universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Improved Algorithms for RNA Folding Paths“

verfasst von / submitted by

Maximilian Faissner, Bakk.techn.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 875
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Bioinformatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Phys. Dr. Ivo Hofacker

Acknowledgements

First of all, I would like to express my gratitude to my supervisor, Ivo Hofacker. His support, constructive feedback, and guidance were instrumental in the successful completion of this thesis. I am grateful for his vast and extensive knowledge in the field of RNA bioinformatics, which constantly guided the direction of my research.

I am grateful for the support of Stefan Badelt at every stage of the project. In the countless discussions and meetings we had, he provided a clearer view of the bigger picture, which was invaluable in advancing this thesis. His feedback greatly improved the quality of this document and directly influenced the content of related presentations.

Special thanks go to Ronny Lorenz, whose knowledge of the ViennaRNA framework proved invaluable in the completion of this thesis. I would also like to acknowledge the support of Christoph Flamm and Michael Wolfinger, who provided assistance and guidance at various stages of this project. I would also like to thank all the student colleagues of the group, including Irene, Maria, Michael, Milan, Thomas, Yuliia, and many others. Our stimulating atmosphere and the discussions spanning a spectrum of subjects we had during the breaks contributed to the overall quality of this thesis.

Finally, I would like to thank my parents for their support and encouragement, which helped me to overcome challenges throughout this journey.

Abstract

RNA research has established itself as a prominent cornerstone at the forefront of scientific discovery in the field of biology. In recent decades, there has been a notable increase in advancements that have enhanced our understanding of RNA molecules and their roles in various biological processes within the cell. This progress was enabled by a paradigm shift, no longer considering RNA solely as an intermediary for genetic information, but recognizing its ability to adopt functional structures. This shift is particularly evident in the field of therapeutics, as demonstrated by the development of COVID-19 mRNA vaccines. Understanding structural stability is crucial in addressing the challenges and limitations associated with RNA-based therapeutics, such as mitigating the cellular degradation of exogenous RNA molecules.

RNA bioinformatics has an extensive history spanning now over two decades, focused on the modelling and prediction of RNA structures. This field of research is particularly valuable due to the abundance of known RNA sequences that lack a known structure. Initially, thermodynamics-based algorithms were developed to identify minimum free-energy structures with reasonable accuracy. Advancements have led to ensemble methods, which explore the dynamic behaviour of multiple structures within the energy landscape of a given RNA molecule.

This thesis has its focus on the dynamics of RNA folding. It aims to model and predict the dynamic behaviour and conformational changes of a single sequence capable of adopting multiple known alternate structures. The primary focus revolves around estimating RNA folding paths and identifying energy barriers between two alternate structures. There are multiple algorithmic approaches for estimating folding paths, which will undergo assessment and comparison. Several options will be discussed, aiming to enhance one of the existing algorithms significantly. Considering the established occurrence of alternative RNA conformations in numerous instances across all domains of life, this theoretical research in the field of RNA bioinformatics will contribute to advancing our comprehensive understanding of the RNA ensemble.

Kurzfassung

Die RNA-Forschung hat sich als ein bedeutender Bestandteil an der Basis der wissenschaftlichen Entdeckungen im Bereich der Biologie etabliert. In den letzten Jahrzehnten gab es eine beachtliche Zunahme von Forschungsergebnissen, die unser Verständnis von RNA-Molekülen erheblich verbessert haben. Dieser Erfolg wurde durch einen Paradigmenwechsel begünstigt, bei dem die RNA nicht mehr nur als Intermediär zur Übertragung genetischer Informationen betrachtet wird, sondern ihre Eigenschaft, strukturelle Funktionen zu übernehmen, berücksichtigt wird. Dieser Wandel zeigt sich besonders deutlich im Bereich der Therapeutika, wie beispielsweise bei der Entwicklung von COVID-19 mRNA-Impfstoffen. Entscheidend ist das grundlegende Verständnis der molekularen RNA-Stabilität, die so beispielsweise die Abbaugeschwindigkeit exogener RNA-Moleküle steuern kann.

Die RNA-Bioinformatik blickt auf eine mehr als zwei Jahrzehnte lange Geschichte zurück, die sich auf die Modellierung und Vorhersage von RNA-Strukturen konzentriert. Dieser Teil der Forschung ist insbesondere deshalb relevant, weil es eine Vielzahl bekannter RNA-Sequenzen gibt, deren Struktur unbekannt ist. Zunächst wurden Algorithmen zur Identifizierung von Strukturen mit minimaler freier Energie und guter Präzision auf der Grundlage der Thermodynamik entwickelt. Die Weiterentwicklung führte zu Ensemble-Methoden, die das dynamische Verhalten mehrerer Strukturen innerhalb der Energielandschaft eines bestimmten RNA-Moleküls analysieren.

Diese Arbeit befasst sich mit der Dynamik der RNA-Faltung, konkret mit der Modellierung und Vorhersage von Konformationsänderungen einer einzelnen Sequenz, die mehrere bekannte alternierende Strukturen annehmen kann. Das Ziel besteht darin, die RNA-Faltungspfade zu bestimmen, um Energiebarrieren abzuschätzen. Verschiedene verfügbare Algorithmen zur Abschätzung von Faltungspfaden werden bewertet und verglichen, einer dieser wird anschließend verbessert. In Anbetracht des nachgewiesenen Auftretens alternativer RNA-Konformationen in allen Bereichen des Lebens soll diese theoretische Forschung auch dazu beitragen, unser umfassendes Verständnis des RNA-Ensembles zu verbessern.

Contents

CHAPTER 1

Introduction

Both RNA and DNA are biopolymers comprised of elementary nucleotide building blocks. Nucleotides consist of a sugar molecule (ribose for RNA), a phosphate group, and a base. The four elementary nucleotides for RNA are adenine, cytosine, guanine, and uracil. In nature, DNA primarily consists of two linked strands forming a double helix, while RNA is always single-stranded. This characteristic of RNA allows for the formation of distinct three-dimensional structures, where a specific fraction of nucleotides can form base pairs to create helices.

Similar to protein structures, RNAs exhibit primary, secondary, and tertiary structures. The primary structure refers to the sequence of nucleotides, while the secondary structure represents the arrangement of base pairs. The tertiary structure of RNA determines its ultimate three-dimensional confirmation and is the foundation for any interaction with other molecules.

In RNA, base pairs at the secondary level predominantly consist of Watson-Crick-type base pairs (canonical base pairs), such as G-C and A-U, and also include the G-U wobble pairing. Additionally, various energetically weaker non-canonical base pairs may form. The incorporation of modified nucleotides, like inosine or pseudouridine, extends the range of possible base pairings.

The RNA folding process involves the formation of secondary and tertiary structures from a given RNA sequence. Various factors, including environmental variables and chaperone proteins, can influence the folding process. However, it is primarily the hydrophobic effects that are attributed as the main driver of folding, which is primarily mediated through base-pairing. From the current scientific perspective, RNA folding can be abstracted as a simplified hierarchical process (Figure 1.1), since the tertiary structure is heavily dependent on the secondary folding level

(Tinoco and Bustamante, 1999). The initial stage involves the establishment of nested base pairs accompanied by various related structural motifs, such as hairpin loops, internal loops, and bulges. Stacked Watson-Crick-type base pairs play the most significant role in contributing to the overall folding energy and stability. Subsequently, overlapping pseudoknots (non-nested bonds) and other three-dimensional motifs are formed during a later stage. This simplification greatly aids our understanding of the folding process, important for various fields like synthetic biology or RNA-based therapeutics.

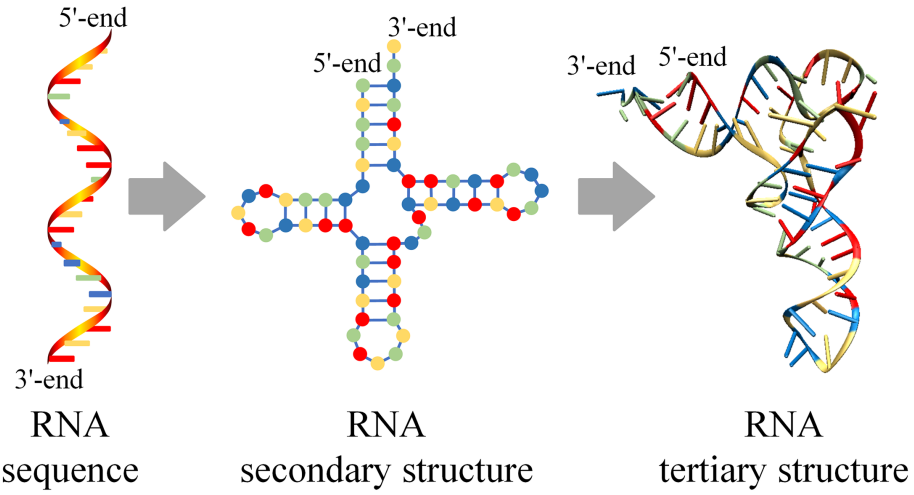


Figure 1.1: Simplified Hierarchical Folding Process of an RNA. The formation of secondary structure motifs precedes the formation of tertiary bonds. Adapted from Zhao et al. (2021) under the Creative Commons Attribution 4.0 License (CC BY 4.0).

Similar to computational biology research focusing on proteins, key areas of interest in RNA research include thermodynamic structure prediction for non-coding RNAs (ncRNAs) and regulatory structures in messenger RNAs (mRNAs), as well as the prediction of RNA-based interactions. This field also encompasses the development of kinetic models to capture the structural changes that RNA molecules may undergo throughout their biological lifespan. Predictive, in silico models in these areas hold significant research value, particularly due to the unresolved structures of the majority of sequenced ncRNAs, which are not feasible to experimentally validate using techniques such as NMR spectroscopy or X-ray crystallography (Zhao et al., 2021).

After discussing the fundamental aspects of RNA thermodynamic folding, RNA kinetics will be the focus of this thesis. This allows investigating of alternative structural conformations that exist within the energy landscape of RNA. Structures

of interest typically have comparable free energies to the most thermodynamically favorable structure, known as the minimum free energy (MFE) structure. This characteristic is a result of the inherent properties of nucleotide building blocks. The limited diversity of nucleotides means that multiple combinations of differently stacked base pairs are often interchangeable in terms of stability. Additionally, suboptimal combinations of stacked pairs have been found to be biologically relevant due to tertiary interactions, further stabilizing the molecule (Wu and D'Souza, 2020).

Well-documented examples are abundant in scientific literature where RNA structures are influenced by dynamic signals. For instance, riboswitches exhibit two distinct conformations depending on the presence or absence of specific ligands, but they can also undergo changes as a response to a change in temperature, pH, or metal ion concentrations. These different conformations often function as switches, playing a crucial role in gene expression. (Wu and D'Souza, 2020) Another example that highlights the relevance of multiple conformations can be observed during transcription. The rate of synthesis with cotranscriptional folding paths can strongly impact the folding process, leading to scenarios where the thermodynamically optimal MFE structure may not form. Instead, kinetically trapped structures may arise, which persist in energetically suboptimal conformations throughout the lifetime of the RNA molecule in vivo (Xayaphoummine et al., 2006).

RNA bioinformatics has made significant advancements in the development of various tools for predicting and simulating the dynamic behaviour of structural ensembles within RNA energy landscapes. The primary goal of this thesis is to enhance an existing heuristic algorithm, aimed at generating near-optimal folding paths with related energy barriers between two RNA structures. This algorithm builds upon the `Findpath` algorithm (Flamm et al., 2001a), which is a freely available folding path algorithm as part of the ViennaRNA package (Lorenz et al., 2011). The proposed improvements will refine the accuracy and efficiency of predicted RNA folding paths and provides valuable insight for follow-up projects.

To introduce the concept of folding paths, this thesis will begin by explaining the mathematical formal definition of RNA secondary structures. This will be followed by an exploration of the thermodynamic and kinetic aspects of RNA folding, which will lay the foundation for the introduction of different approaches to estimate RNA folding paths. The subsequent discussions will primarily focus on the secondary structure level, as it is often adequate for functional analysis due to the assumed hierarchical nature of folding. In contrast, the precise three-dimensional arrangement of atoms at the tertiary structure level is a relatively minor area of research in comparison (Zhang et al., 2022).

CHAPTER 2

Background

2.1 RNA Structure

2.1.1 Secondary Structure Notations

In this first section of this chapter, a formal notation for RNA secondary structures is introduced. Various publications in the field of RNA bioinformatics, such as the RNA folding review article by Ponty and Reinharz (2022), employ similar notations. In the subsequent section that discusses the introduction to folding paths, further definitions will be presented.

This framework will be used consistently throughout this thesis. As discussed in the introduction chapter, the folding process can be simplified as a hierarchical process, with the secondary structure being adopted first. The main focus of this thesis will be on RNA folding paths, restricted to the secondary structure level. This is in line with many related studies within the field of RNA bioinformatics. The introduced notation reflects this approach with additional typical restrictions, such as only considering the standard Watson-Crick base pairs formed between unmodified nucleotides and the omission of pseudoknot-based structures.

1. A **sequence** $\sigma = (N_1, ..., N_n)$ is defined as ordered list of letters with length n with a limited alphabet of nucleotides $\Sigma = A, G, C, U$. A sequence always starts at the $5'$ end.

2. A (secondary) **structure** S is a set of base pairs (i, j) in $[1, n]$ with the following constraints:

a) $1 \leq i < j \leq n$ for every base pair (i, j)

b) only canonical base pairs are valid: $(N_i, N_j) \in \{(A, U), (U, A), (C, G), (G, C), (G, U), (U, G)\}$

c) a nucleotide can form only a single base pair

d) all base pairs are nested or parallel, crossing pairs are not permitted: if $(i, j), (i', j') \in S$ and $i < i'$, then $i < k < l < j$ or $i < j < k < l$.

e) hairpin loops have no sterically impossible turns, the minimum amount of unpaired bases in turns is 3: $j - i > 3$

3. **pseudoknots** are crossing base pairs which invalidate nested secondary structures, and are therefore not permitted.

2.1.2 Secondary Structure Visualizations

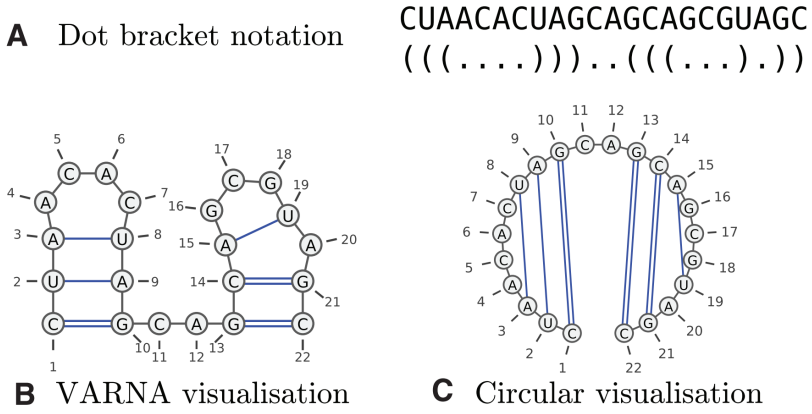


Figure 2.1: Illustrations of a short RNA sequence and its corresponding secondary structure. In such planar drawings of secondary structures, there are no intersecting base pairs when pseudoknots are absent. Adapted from Golden et al. (2019) under the Creative Commons Attribution-NonCommercial-NoDerivates 4.0 International License (CC BY-NC-ND 4.0).

Since permitted secondary structures are always perfectly nested, visualizations can always be planar without the need for a third dimension. The following plots, as seen in Figure 2.1, will be used throughout this thesis:

The **Dot-bracket notation** is a compact structure representation where opening and closing parenthesis indicate base pairs. Only a single type of bracket is sufficient for nested structures. Unpaired nucleotides are represented with a dot. The dot-bracket notation can be simply processed by algorithms as **input string**, which subsequently convert the string notation into adjacency matrices or adjacency lists.

The **circular representation** places the RNA backbone on a circle, canonical base pairs can be seen as links within. Such drawings can be created with simple algorithms without user input.

Sophisticated secondary structure graphs, as typically seen in publications, visualize structure motifs such as bulges, hairpins, and internal loops (Figure 2.2). These require complex drawing algorithms which aim to find visually pleasing representations without intersections on a plane, but may also require user input for final touch-up. Typical algorithms, which calculate nucleotide coordinates on a plane, are NAVIEW (Brion and Westhof, 1997) and RNApuzzler (Wiegreffe et al., 2018).

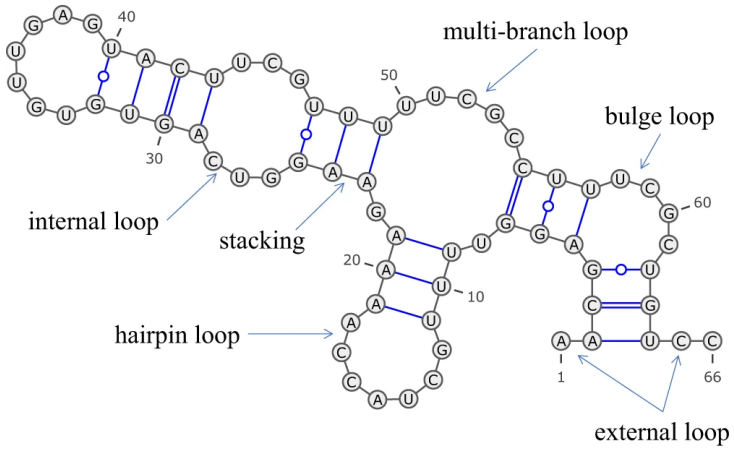


Figure 2.2: Illustration of Regions and Loop classifications in RNA secondary structures. Source: Adapted from Sato et al. (2021) under the Creative Commons Attribution 4.0 License (CC BY 4.0).

2.2 Thermodynamic Folding

2.2.1 Energy Models and MFE structures

The introduction of an energy model is a fundamental concept for assessing the stability of an RNA structure. One of the most popular energy models in RNA bioinformatics is the **nearest neighbor model**. This model, which does not consider pseudoknots, simplifies the calculation of the free energy (or thermodynamic stability ΔG_s) of a secondary structure by summing its loop contributions, as depicted in Figure 2.3. The "Turner"-free-energy model developed by Mathews et al. (2004) is a well-accepted set of energy parameters for the nearest neighbor model.

The basic premise is that the stability of a given motif only depends on its directly adjacent Watson-Crick base pairs in the context of its sequence. Experimentally determined energy parameters for smaller motifs allow to build up all contributing free energy factors for any given structure.

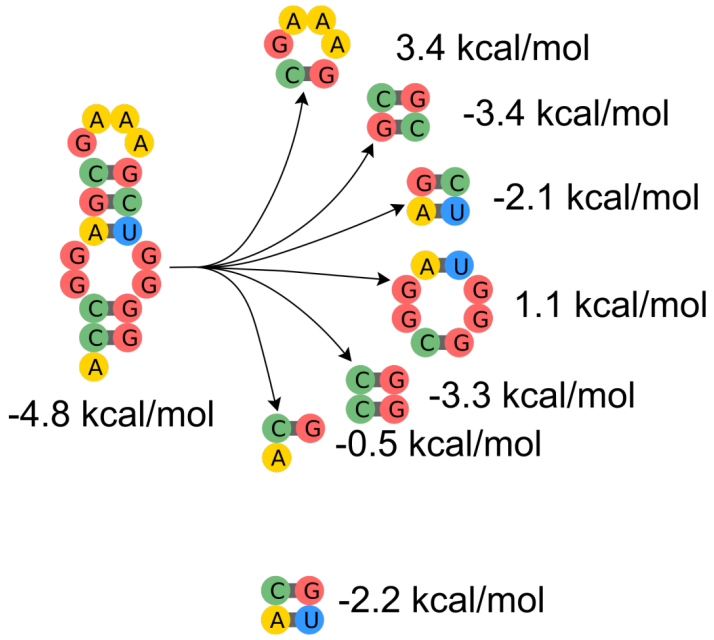


Figure 2.3: Illustration of RNA Secondary Structure Decomposition Based on the Nearest Neighbor Model. Adapted from Wu (2018) under the Creative Commons Attribution 4.0 License (CC BY 4.0).

Out of all possible secondary structures of a sequence, the **minimum free energy structure** (MFE) has the most optimal conformational stability at thermodynamic equilibrium. Assuming the molecule folds into the lowest energy structure during

its lifetime, the prediction of a folded structure becomes an optimization problem, enabled by a solid energy model (Zhao et al., 2021).

Most traditional folding algorithms are based on this paradigm: A single MFE structure is computed with sequence and additional applied energy parameters as input. Typical algorithms, such as RNAfold (Hofacker, 2003) or Mfold (Zuker, 2003), employ dynamic programming as an elegant and fast solution to break down candidate structures in a recursive fashion. Dynamic Programming algorithms are not limited to MFE computations, typical other estimates include MEA (maximum expected accuracy) or ranges of suboptimal structures below the thermodynamic equilibrium. The accuracy of predicted MFE-structures without pseudoknots tends to plateau at around 80% (Seetin and Mathews, 2012; Fu et al., 2021). The computation of MFE structures with pseudoknots is not possible with DP algorithms, as this problem is considered NP-complete (Lyngsø and Pedersen, 2000).

In contrast to energy minimization-based methods **machine learning methods** are data-driven and don't require assumptions about the in vivo folding mechanism. Various algorithms, for example UFold (Fu et al., 2021), show promise, especially under the aspect of predicting structures with pseudoknots. Notably, deep learning-based folding algorithms remain problematic due to training data biases and related overfitting concerns: Only a low number of available ground-truth secondary structures (e.g. via X-ray crystallography or NMR) are available which inevitably makes it challenging to assess capabilities or shortcomings (Zhao et al., 2021; Flamm et al., 2022).

2.2.2 Finding alternative conformations

Shifting focus from the task of identifying a single optimal secondary structure, it becomes necessary to establish **energy landscapes** and **structure ensembles**. Even if a predicted structure at equilibrium is well-defined, cellular environments often require the identification of multiple representative structures. There are various reasons that can limit the accuracy of MFE predictions, such as the presence of pseudoknots, environmental factors, or dominant folding kinetics, which means that the native state may not be at equilibrium of the energy landscape.

If considering all possible structures of a given sequence under a **Boltzmann distribution**, the probability that any structure S exists in the ensemble of all possible structures $\mathcal{S}$ depends on the free energy $E(S)$ and system temperature T:

$$\mathbb{P}(S|\sigma) = \frac{e^{-\frac{E(S)}{RT}}}{\mathcal{Z}}$$

R refers to the Boltzmann constant $1.987 \times 10^{-3} kcal.mol^{-1}K^{-1}$, and $\mathcal{Z}$ refers to the **partition function**, which is the sum of all structure energy contributions:

$$\mathcal{Z} = \sum_{S \varepsilon \mathcal{S}} e^{-E(S)/(RT)}$$

For a given sequence σ, the aforementioned MFE structure has the highest ensemble probability. With increasing sequence lengths, the probability of being at equilibrium gets smaller in absolute terms, which inherently reduces the accuracy of MFE structure predictions.

Ensemble approaches were initially developed as an alternative to MFE predictions by computing average properties either by explicitly computing $\mathcal{Z}$ or via stochastic sampling. One of the first practical algorithms was introduced by McCaskill (1990), who developed a dynamic programming algorithm that allows computing **base pairing probabilities** for all possible pairs. Despite the exponential nature of potential structures beginning from the lowest energy MFE structure, the partition function can be computed over all nested structures in $O(n^3)$. Confidence estimates for base pairs extend equilibrium predictions and help to assess the reliability of predictions by highlighting reliable and unreliable regions. Other related ensemble approaches compute average properties which aim to predict representative structures as MFE alternatives, e.g. centroid or maximum expected accuracy (MEA) structures. DP algorithms can also be used to compute suboptimal structures, e.g. by enumerating all structures within a given energy range relative to the equilibrium energy (Wuchty et al., 1999).

Essentially, all the approaches discussed so far offer a static perspective of RNA landscapes. However, since RNAs can exhibit dynamic structural behaviour, these methods are limited in terms of accurately predicting relevant structures in cellular environments. In other words, an RNA molecule may not have enough time to reach its energetic minimum within the free energy landscape before it degrades, making it difficult to study dynamics. Folding dynamics and its implications will be covered in the next section.

2.3 Kinetic RNA Folding

2.3.1 RNA folding as dynamic process

For RNA folding, out-of-equilibrium states can be of importance in various scenarios inside cellular environments, as the function of an RNA molecule is not only determined by the sequence but also by the folding process. RNA molecules have an inherent tendency to be **kinetically trapped**: the structure at thermodynamic equilibrium is not necessarily identical to functional alternate structures (Tinoco and Bustamante, 1999). Trapped states are relevant for all but the smallest RNA molecules. Thermodynamic modelling does not consider the amount of energy and time required to leave such trapped states.

Understanding the dynamic folding process requires knowledge of how RNA explores the vast energy landscape, i.e. estimating likely states with energy barriers in between. Energy landscapes of RNAs are often described as "rugged", as molecules have to overcome potentially high energy barriers to (re-)fold into a different state. Compared to other biopolymers like proteins, the stability gap in RNA ensembles (free energy difference between alternate states) is comparatively small (Lin et al., 2012; Thirumalai and Hyeon, 2005) This can be attributed to the "homopolymer" (single building block) nature of RNA structures - RNAs consist of nucleotide building blocks with limited chemical diversity. An RNA molecule is, therefore, more likely kinetically trapped than a "typical" protein, which enables the presence of alternate folding states. As an outcome, this increases the complexity of computational kinetic modelling and design (Thirumalai and Hyeon, 2005).

Another aspect of the study of RNA kinetics is how RNA molecules change their confirmations during the process of transcription. If considering that the folding process starts early, the strand production rate heavily influences final structures. As a result, multiple alternate, potentially trapped structural states can be present at any given time. Structural changes within these states are common, which are for example induced by environmental triggers like small molecule ligands.

2.3.2 Kinetic Modelling and Move Sets

Before introducing various approaches for modelling RNA folding kinetics, it is important to understand the in silico environment of the folding process. For modelling, the energy landscape is abstracted as a directed graph, where the nodes represent secondary structure conformations. The connections between these nodes are defined with **move sets**, which determine how a structure can navigate through the landscape.

Excluding molecular dynamics approaches, a widely used microscopic strategy, as described in Flamm et al. (2000), is to represent edges as **elementary moves** or step-wise transitions which correspond to the formation or removal of individual base pairs. However, it is important to note that even this extensive model is a simplification. For example, it neglects cooperative mechanisms, which are known to exist from e.g. molecular dynamics simulations (Bian et al., 2015). The step-wise model can be further expanded by incorporating additional features, such as base-pair shifting or nucleotide elongation for cotranscriptional folding modelling. In contrast, more macroscopic approaches, as discussed in Xayaphoummine et al. (2005), consider the addition or deletion of entire helices as a single transition.

The final modelling ingredient is the energy model, which assigns fitness values to confirmations and moves with respective rates between valid transitions.

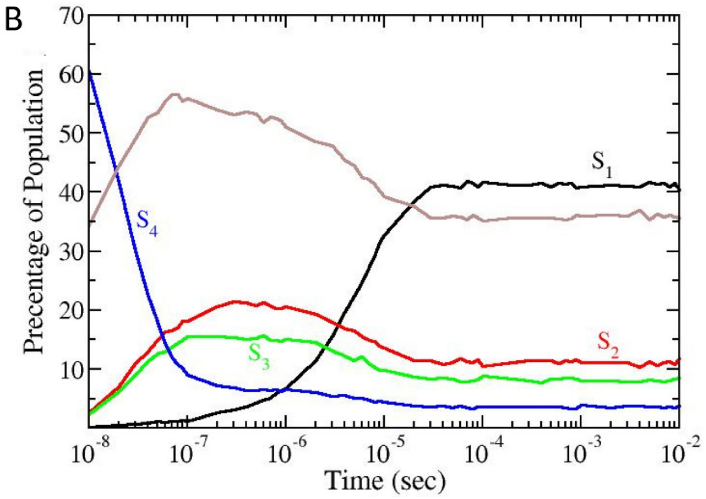


Figure 2.4: Following the Gillespie algorithm, RNA kinetics are modelled as a continuous time Markov chain. This allows the simulation of structures with respective folding times, i.e. how populations of conformations evolve in time. Example: Sample sequence with 20 nucleotides, the three dominant low energy structures are labelled S_1 to S_3, S_4 is the open chain configuration. Simulated with the KFOLD algorithm (Dykeman, 2015), which is also the source of this illustration (Creative Commons Attribution 4.0 License CC BY 4.0).

These abstraction layers allow representing the kinetics of a folding process as a continuous-time Markov process. This enables the examination of conformation populations as they change over time, as demonstrated in the example shown in Figure 2.4. Here, the energy landscape has to be strongly connected with reversible edges. The resulting Markov chain is both ergodic and irreducible. Additionally, the transition rates between states must adhere to the fundamental principle of

the detailed balance condition, leading to a stable equilibrium condition (Flamm and Hofacker, 2008).

One method to mathematically describe the Markov process is via the chemical master equation, which evaluates the time evolution of probability distributions:

$$\frac{dP_x(t)}{dt} = \sum_{y \neq x} (P_y(t)k_{xy} - P_x(t)k_{yx})$$

where $P_x(t)$ is the probability of state x at time t, k_{xy} and k_{yx} are influx and outflux rates, respectively, referencing two states x and y which are directly kinetically connected. The master equation can also be expressed in the vector form

$$\frac{d}{dt}P(t) = KP(t)$$

where K is the transition rate matrix, and $P(t)$ is the probability distribution vector. This vector equation can be solved to determine the time evolution of system states with

$$P(t) = e^{tK}P(0)$$

where P(0) is the initial probability distribution vector, and e^{tK} is a matrix exponential (Flamm and Hofacker, 2008; Ge and Qian, 2013).

This allows exact folding simulations for short molecules, however, for almost all practical applications finding predominant confirmations over time becomes infeasible due to the number of possible confirmations. In general, computational approaches for RNA kinetic modelling fall into 3 main categories, which all aim to overcome this inherent limitation.

Simulation approaches model folding kinetics with stochastic methods, such as the rejection-less Monte-Carlo simulation variant (Gillespie algorithm in chemistry), allow the computation of equilibrium time and states. Starting conformations for simulations can be empty secondary structures or any other relevant low-energy state. The Gillespie algorithm requires tracking large numbers of folding trajectories since each structure transition is evaluated per its transition probability (rejection-less approach).

One of the first published Gillespie-based RNA folding simulation algorithms is `Kinfold` (Flamm et al., 2000), which implements microscopic transition at the base-pair level in conjunction with the Turner energy model (Mathews et al.,

1999). Other algorithms trade precision with computational runtime by employing more macroscopic move sets, as described earlier. In general, simulation-based approaches are often considered the gold standard for RNA kinetics (Senter and Clote, 2015) but are time-consuming and thus limited in practice to short sequence lengths, assuming the starting simulation point is an open-stranded RNA molecule.

A completely different approach is to only model **macro states** in the energy landscapes with a **coarse graining** method. One such algorithm is `BARRIERS` (Flamm et al., 2001b), which defines local optima in the energy landscape. Subsequently, the master equation can be solved with `Treekin` (Wolfinger et al., 2004).

Finally, RNA kinetics can also be modelled via energy barriers of **folding pathways**. The most basic example is a folding pathway from an empty structure to the MFE structure of a sequence. Rates are related to the minimum energy barrier of all folding pathways between two structures. Folding pathways will be introduced in detail in the next section.

2.4 Folding Paths

This section explores the modelling of RNA folding, which is the process by which complex secondary and tertiary structures are formed or change over time. A basic example of a folding path involves the transformation from an open-stranded RNA molecule to a specific native structure that is biologically significant. Additionally, the folding paths of interest often involve transitions between two **local minima** within the folding landscape, commonly referred to as **refolding paths**. The primary focus of folding paths revolves around the rates of folding events. Typically, the assembly of helices can take place within microseconds, while the complete folding of a native structure, including the formation of tertiary interactions, may involve traversing multiple energy barriers, resulting in folding rates ranging from milliseconds to hundreds of seconds (Woodson, 2010).

Numerous experimental studies have been conducted that enhance our understanding of various aspects related to RNA folding, which form the basis of the development of folding algorithms. For instance, studies conducted by different research groups on the pathways of the *Tetrahymena* ribozyme have revealed the presence of both slow and fast folding pathways (Treiber and Williamson, 1999), along with the identification of certain intermediates that act as kinetic traps Mitchell and Russell (2014). Here, notable differences have been observed between in vivo and in vitro folding pathways, indicating that RNA chaperones assist in overcoming energy barriers within the folding landscape.

From a bioinformatics perspective, the modelling of folding paths involves a sequential approach to capture the succession of structural intermediates. As discussed earlier, the selection of a suitable move set is critical. The folding path algorithms discussed in this context specifically target the microscopic level of folding, involving elementary base pair additions and deletions, as initially introduced by Flamm et al. (2000).

All path models assume the notation of perfectly nested secondary structures introduced previously. These paths represent structural changes within a single RNA sequence σ, between two predefined secondary structures. Figure 2.5 illustrates a basic example of a sample path represented in dot-bracket notation. It is important to note that this thesis does not delve into folding path algorithms for tertiary structures that involve pseudoknots.

The primary goal of folding path algorithms is to determine a sequence of elementary moves that minimizes the energy barrier along the path. The height of the barrier directly impacts the folding path probabilities between two structures, and knowledge of these energy barriers allows for the calculation of transition rates and kinetic studies. Following this, the framework of folding paths will be formally

introduced, and various pathway algorithms will be explored and discussed.

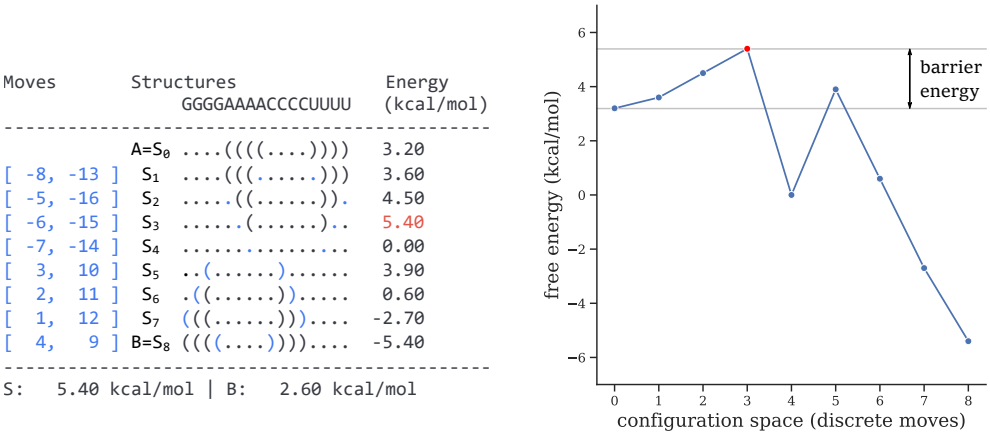


Figure 2.5: A sample folding path connecting two structures A and B of a small toy-sequence σ. On the left, intermediates of the folding path are depicted in dot-bracket-notation, including elementary moves (positive base-pair additions and negative deletions), and their respective free energies (kcal/mol). The graph on the right illustrates the relationship between free energy and folding progression, clearly displaying the energy barrier.

2.4.1 Preliminaries

1. A **folding path** $P(A, B)$ between two secondary structures A and B of a given sequence σ is defined as sequence of intermediate structures $A = S_0, S_1, \ldots, S_n = B$, if consecutive intermediates S_i, S_{i+1} differ by a single base-pair addition or deletion (**elementary move**).

2. $P(A, B)$ has a distance n, which is equal to the number of elementary moves.

3. n has a well-defined minimum length, which is equal to the **base-pair distance** (or Hamming distance) between two structures: $n_{min} = d(A, B)$. Such minimum-lengths paths, which only contain the addition and removal of distinct pairs, are defined as **direct path**. If a path contains temporary base pairs ($n > d(A, B)$), a path is defined as **indirect**.

4. The free energy of each intermediate has the notation $E(S_i)$. The **saddle energy** of a path (or simply path energy) is $E_S = \max(E(S_0), E(S_1), ..., E(S_n))$. The **barrier energy** is $E_S - E(A)$.

5. **Optimal** folding paths have the lowest possible saddle or barrier energy. This definition references either *direct* or *indirect* paths, i.e. a given optimal direct path may have a higher saddle energy than a optimal indirect path.

6. Folding paths **with repeats** may contain multiple identical elementary moves, which are required for certain optimal paths. From a graph theoretical perspective, these are defined as walks (repeating edges), folding paths without repeats are proper paths.

2.4.2 Direct and Indirect Paths

Optimal folding paths between two structures are often *indirect* pathways which contain detours. This extends the minimum path length, defined by the base-pair distance, for an arbitrary amount of additional elementary moves. Only indirect paths allow for the formation of additional helices (not present in both A and B) and for the dissociation of constant base pairs. Even for small toy examples (see sample paths in Figure 2.6), indirect paths may contain detours such that path lengths become significantly longer.

```
Path A      GGGGGGCCCCCC
            .((.....)).. -0.10
[ -2, -10 ] ..(.....)...  3.20
[ -3,  -9 ] ..........    0.00
[  4,  10 ] ...(.....)..  3.20
[  3,  11 ] ..((.....)). -0.10
[  5,   9 ] ..(((...))). -2.20
S:   3.20 kcal/mol | B:   3.30 kcal/mol
```

```
Path B      GGGGGGCCCCCC
            .((.....)).. -0.10
[  4,   8 ] .(((...))).. -2.20
[ -2, -10 ] ..((...))...  1.10
[  2,  11 ] .(((...)).).  1.60
[  1,  12 ] ((((...)).)) -0.70
[ -4,  -8 ] (((.....).)) 1.40
[ -3,  -9 ] ((........))  0.70
[  4,  10 ] ((.(.....))) 1.40
[  5,   9 ] ((.((...)))) -0.70
[ -1, -12 ] .(.((...))).  1.60
[ -2, -11 ] ...((...)).. 1.10
[  3,  11 ] ..(((...))). -2.20
S:   1.60 kcal/mol | B:   1.70 kcal/mol
```

Figure 2.6: Direct and indirect sample folding paths for the sequence GGGGGGC-CCCCC between 2 structures. Direct path A consists of 2 base-pair deletions and 3 additions, the total base-pair distance is 5. Indirect Path B has 6 extra elementary moves which results in a lower barrier energy.

The optimal energy barrier among all *indirect* paths is lower or equal to the best possible *direct* path barrier since the latter are often forced into high-energy intermediates in a small corridor within the total energy landscape. Such intermediates frequently contain energetically unfavourable loop motifs, leading to an overestimation of energy barriers (Lorenz et al., 2009).

For both variants, the exhaustive enumeration of all optimal paths is only possible with exponential runtime algorithms: The simpler *direct* path problem without pseudoknots in RNA energy landscapes was proven NP-complete by Manuch et al. (2009). The solution to the direct path problem can be considered as a less precise approximate solution to the global energy barrier problem. Practically, the computation of near-optimal direct paths for any (reasonable) sequence and path

length can be performed by fast heuristic algorithms. Computing indirect paths is more challenging due to its vastly increased energy landscape size.

2.4.3 Computation of Direct and Indirect Paths

Various algorithms are available to determine both optimal and suboptimal folding paths. Presented algorithms estimate free energies with the Turner energy model (Mathews et al., 2004) or can be adapted for it.

Optimal Pathway Algorithms

`BARRIERS` (Flamm et al., 2001b) is a flooding algorithm for computing local minima in RNA landscapes. It relies on `RNAsubopt` (Wuchty et al., 1999) (Lorenz et al., 2011) to enumerate all secondary structures within an energy band (referenced by the MFE structure) to compute barrier heights between minima. It can be used to compute optimal folding paths, sometimes referred to as ground-truth paths, if both structures are local optima. Since `RNAsubopt` is an exponential runtime algorithm, path computations are only feasible if the difference between path saddle energies and the MFE is not too large. In practice, this only works for rather short sequences.

Another exponential time algorithm by Takizawa et al. (2020) utilizes Dijkstra's algorithm to compute optimal solutions for direct paths. In a similar vein, for practical purposes, it is limited to structure pairs with base-pair distances not larger than 20.

Greedy Folding Path Heuristic

The Morgan-Higgs algorithm (1998) was the first simple greedy algorithm for finding valid direct paths of length n. In the context of folding paths, a greedy choice refers to selecting the elementary move which points towards the lowest energy state from the neighbourhood of a given intermediate. Its intention is to provide a first estimate (or upper bound) for energy barriers. Naturally, resulting energy barriers are unlikely to be equal to optimal path barriers.

In contrast to all other presented algorithms, the Morgan-Higgs heuristic was originally intended for the simplified Nussinov energy model (Nussinov and Jacobson, 1980). The original algorithm intends to produce "reasonable" paths by simply balancing the formation and removal of helices (conflict-driven balancing). For example, instead of 3 consecutive deletion moves (d) and 3 consecutive add moves (a), the heuristic would yield the result d-a-d-a-d-a, assuming no incompatible moves. Due to identical energy contributions (-1/+1) of all base-pair additions and

deletions (Nussinov energy model), the algorithm often yields randomized results. The authors, therefore, recommended to run the algorithms multiple times until convergence.

The selection of each greedy choice per intermediate structure involves a validity check. Not all remaining elementary moves can be considered as potential candidates, as some moves may introduce base pairs that are incompatible with the considered intermediate. This straightforward approach of disregarding invalid moves guarantees that the final structure B remains reachable, as there are no valid move permutations in folding paths that lead to dead ends. This holds true even when adopting the pathfinding process to the Turner energy model (Mathews et al., 2004).

Algorithm 2.1: Greedy Folding Path Heuristic (Input: σ, A, B)

```
1  S ← A; E_s ← E(A)                          ▷ initial intermediate and saddle energy
2  L ← List of elementary moves (i,j) of direct path P(A, B)
3  for n ← 1 to d(A, B) do
4      E'_s ← ∞
5      foreach (i,j) in L do
6          Perform candidate move and validity Check S --(i,j)--> S'
7          if (i,j) is valid move for S and E(S') < E'_s then
8              E'_s ← E(S'); S* ← S'           ▷ Save S' with lowest energy as S*
9      E_s ← max(E_s, E'_s); S ← S*            ▷ Update saddle energy
10 return E_s
```

The presented greedy pathfinding algorithm in 2.1 is an energy model agnostic variant, which is often cited as Voss et al. (2004) method. It only returns the saddle energy E_s. The method can also be adapted to return $P(A, B)$ by keeping track of all visited intermediates. Every iteration yields a new intermediate structure with one move closer to B. Available moves are checked for validity: certain base pairs may need to be removed first before attempting to add base pairs. The candidate intermediate with the lowest energy is selected. This procedure is repeated until the current intermediate structure is equal to B.

Findpath Algorithm

The `Findpath` algorithm (Flamm et al., 2001a) is a bounded breadth-first search (or beam search) folding path heuristic which potentially produces near-optimal direct paths. It improves upon the Morgan-Higgs algorithm in many ways. This heuristic is only intended for the Turner energy model, and is part of the current ViennaRNA framework (Mathews et al., 2004; Lorenz et al., 2011). Since `Findpath` is integral to this thesis, this algorithm will be explained in detail.

The algorithm can be seen as an extension of the idea coined by the Morgan-Higgs heuristic. It introduces an additional parameter, the **search width** m, which regulates the search by retaining all m best paths for every distance step. The underlying principle is that optimal direct paths will almost never be the result of purely greedy decisions, however, often the second or third best move from a given intermediate may be sufficient for finding the best possible path. Also, if all potential intermediates for the next iteration are sorted by energy: there are diminishing returns if retaining all structures with unfavourable energies.

Practically, when the search width is set to $m = 1$, the resulting path is computed greedily and yields the same outcome as algorithm 2.1. On the other hand, assuming there are n_p possible valid paths, setting $m = n_p$ would lead to an exhaustive search. Therefore, the selection of the appropriate value for m involves finding a balance between obtaining near-optimal folding paths and considering computational runtime constraints, which ultimately rests with the user's decision.

Implementation-wise, `Findpath` builds upon algorithm 2.1 but keeps track of m paths $P_m(A, B)$ with an associated list of remaining moves L_m and saddle energy E_{s_m}. In each iteration, candidate paths $P'_m(A, B)$ are branched off all $P_m(A, B)$, which are one elementary move closer to B. m candidate paths with the lowest saddle energy are retained for the next iteration. Notably, candidate paths are retained independently of their origin, i.e. all retained paths might come from a single $P_m(A, B)$ from the last iteration. Additionally, a threshold energy E_{max} is introduced: intermediates with free energies above E_{max} are ignored. The presented pseudocode in algorithm 2.2 is a simplified `Findpath` variant which only returns a saddle energy E_s, no previously visited intermediates are tracked.

It is important to adjust the search width m based on the length of the folding path n (or base-pair distance). This is because longer paths require higher search widths to remain close to the optimal saddle energy. To enable performance comparisons across different path lengths, this thesis often references m as **search width multiplier** m^*, which is defined as $m^* = m/n$.

`Findpath` employs various optimization techniques to speed up the computation. **Search passes** are performed from both directions $A \to B$ and $B \to A$, since

Algorithm 2.2: Findpath (Input: σ, A, B, m, E_{max})

```
1  S_0 ← A; S = [S_0]                    ▷ list of intermediates S with single initial entry
2  E_{s_0} ← E(A);  E_s ← [E_{s_0}]      ▷ initial list of saddle energies E_s
3  L ← List of elementary moves (i,j) of direct path P(A, B)
4  for n ← 1 to d(A, B) do
5      E'_s ← []; S' ← []                ▷ empty lists for branch paths
6      foreach S_m in S  and  E_{s_m} in E_s do
7          foreach (i,j) in L do
8              Perform candidate move and validity Check S_m --(i,j)--> S'_m
9              if (i,j) is valid move for S_m and E(S'_m) < E_max then
10                 E'_{s_m} ← max(E_{s_m}, E(S'_m))      ▷ Update saddle energy
11                 E'_s.insert(E'_{s_m})
12                 S'.insert(S'_m)
13     E_s ← m best E'_s   ▷ Sorting Step: Retain m best lowest energy paths
14     S ← m best S'
15 return min(E_s)
```

near-optimal paths may be easier to find from only one direction. Secondly, the search width m is increased gradually, which significantly increases the overall performance. Early passes with low m yield an initial energy barrier estimate, which is employed to set E_{max} for the following iterations with higher m. Thirdly, duplicate intermediate structures with associated paths are removed during the sorting step (line 13 in algorithm 2.2).

The current ViennaRNA implementation (Lorenz et al., 2011) uses the following search strategy, where a total of i search passes are performed:

1. Set current search width $m_i = 1(greedy)$, direction = forward ($A \rightarrow B$), $E_{max} = \infty$
2. Perform a Findpath pass with m_i with given direction and limiting E_{max}
3. E_{max} is adjusted to the maximum saddle energy of the last pass
4. Double the current search width: Set $m_i = 2 \times m_{i-1}$, if $m_i > m$ set $m_i = m$ (last iteration)
5. Reverse the direction for next pass
6. Goto step 2 if $m_i \neq m$

Chapter 3 presents an asymptotic runtime analysis of the `Findpath` algorithm. This analysis reveals that the algorithm exhibits a nearly quadratic scaling with the path length n (Figure 2.7) and a linear scaling with the chosen search width m (Figure 2.8). As m increases, the runtime tends to reach a plateau when approaching exhaustive search scenarios.

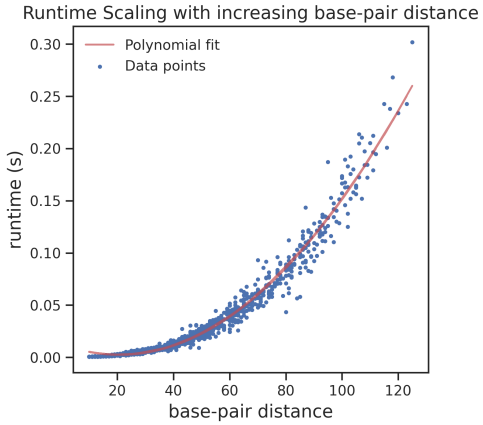


Figure 2.7: Findpath shows nearly quadratic scaling with respect to n. Data points are random folding paths, both search width and sequence length are kept constant.

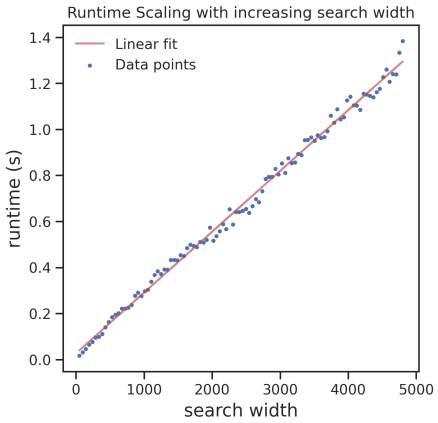


Figure 2.8: Findpath scales linearly with m. Data points are from a single folding path, only the search width is adjusted.

Indirect Path heuristic algorithms

`RNAtabupath` (Dotu et al., 2010) is a heuristic algorithm for indirect folding paths. An adaptive fitness function determines the most promising move from a given structure to a potential intermediate. The algorithm aims to balance free energy vs. distance to the target structure. Ideally, detours from the direct paths are only permitted if not steering too far away from the shortest possible path. This deviation from the direct path is linked to a user-defined input. The construction of resulting folding paths follows a semi-greedy approach, where k lowest energy intermediates are randomly selected and a specific number of "trajectories" are computed. To evenly explore the search space and prevent cycles, a tabu-list is utilized, which also restricts the use of recently added intermediates.

Another indirect path heuristic is `RNAEApath` (Li and Zhang, 2012): The algorithm treats base-pair stacks as consecutive units, where each elementary move is associated with a stack that is either formed or degraded. Starting from a few initial folding pathways, offspring paths are created with various mutation

strategies. Indirect moves are incorporated by adding stacks which do not conflict with intermediates in existing paths. New offspring paths are also generated by rearranging existing moves within paths. The selection of different actions is frequently made through random decisions, making the algorithm non-deterministic.

A completely different class of indirect folding path heuristics are detour-based **mesh-point** approaches. Such algorithms compose indirect paths via intermediate structures as **checkpoints** between the initial and final structure. These additional intermediate structures, referred to as mesh-points, must be positioned outside the direct-path corridor of the energy landscape.

Determining these checkpoints and the computation of folding paths between them are completely different, unrelated tasks. The Shape Triples approach (Bogomolov et al., 2010) falls into this category: `RNAshapes` (Giegerich et al., 2004) is used to compute representative structures ("shreps"), designated as detour checkpoints. A-meta heuristic approach is employed to construct a folding path network, where not all paths between checkpoints are computed exhaustively. In similar vain, the `Pathfinder` algorithm (Lorenz et al., 2009) scans the κ, λ neighbourhood of both initial and final structures via `RNA2dfold` for minimum energy structures, which are subsequently used as checkpoint candidates. Only a maximum deviation from the direct path is allowed, which limits the number of checkpoints. In addition, a variant of the 2D-landscape approach with improved runtimes was developed by Senter et al. (2014). All presented algorithms use direct path heuristics to connect mesh-points (`Findpath`).

Algorithm Comparison

All the approaches described in this section have significant drawbacks. Exponential runtime algorithms can only compute optimal folding paths for short sequences. Direct path heuristics can compute an upper bound on energy barriers at the cost of precision. The existing indirect path heuristics should be considered proof-of-concept algorithms, as they suffer from poor scalability when computing folding paths for longer sequences. In Figure 2.9, the tested indirect pathway algorithms take several minutes to compute a single path for a sequence of 100 nt, depending on the chosen settings. In contrast, `Findpath` computes near-optimal direct paths in less than a second, even with high search-width settings. The estimated energy barriers from indirect path heuristics are often worse than direct path estimates provided by `Findpath`.

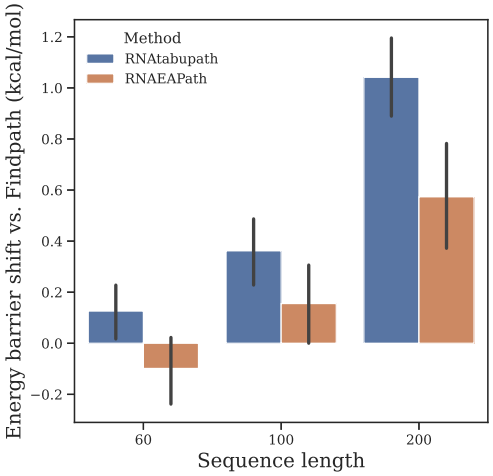


Figure 2.9: Performance tests comparing the average energy barrier shifts of indirect-path heuristics `RNAtabupath` (Dotu et al., 2010) and `RNAEAPath` (Li and Zhang, 2012) to convergent direct-path barriers computed with `Findpath`. The tests involved randomly generated folding paths of defined sequence lengths (60, 100 and 200 *nt*) and varying base-pair distances. The expected result was a negative energy shift, indicating lower energy barriers for computed indirect paths. However, both heuristics showed inferior performance in comparison to `Findpath` for paths of longer sequences.

Thesis Motivation

This master's thesis is motivated by two factors. Firstly, the direct path heuristic `Findpath` has the potential for multiple improvements, both in terms of computational runtime and energy barrier precision. Secondly, the `Findpath` algorithm can be adapted for indirect folding paths. While this adaptation incurs a runtime cost, it has the benefit of preserving the current worst-case precision. In other words, if no indirect paths are found, the energy barrier of the direct path serves as an upper bound.

CHAPTER 3

Advancing Findpath: Algorithm Modifications and Extensions

3.1 Background

The `Findpath` algorithm for RNA folding paths, introduced in the previous chapter, is a freely available software included in the ViennaRNA package (Lorenz et al., 2011) and implemented in ANSI C.

This chapter discusses the modifications and extensions that have been developed during this thesis to enhance the original algorithm. As a new baseline, an optimized variant of the algorithm, implemented in modern C++ (ISO, 2012), has been introduced for all subsequent tests. All discussed extensions are built upon this new variant, which forms the foundation for further development. The details of these optimizations will be discussed in a later section. It is important to note that the breadth-first search algorithm with a user-defined search width m remains the fundamental core component of the heuristic pathfinding approach.

The focus of this thesis revolves around three extensions, which significantly reduce the search space while finding near-optimal paths. These extensions aim to minimize both runtimes and identify paths with lower energy barriers. The first extension, the divide-and-conquer method, will be covered in the upcoming section.

3.2 Dataset Generation: Sample Input RNA Folding Paths

Creating a sample dataset is essential for testing and verifying heuristic algorithms, as it allows this thesis to investigate the effects and modifications to the original `Findpath` algorithm on resulting energy barriers and runtimes. The ViennaRNA implementation was verified to converge to optimal energy barriers within the class of direct RNA folding paths without repeats. In other words, if high search widths m are chosen, the results are identical to those obtained using optimal pathway algorithms, for example, approaches based on Dijkstra's algorithm. For any subsequent modifications, it is essential to identify if adaptations maintain this property and if not, to identify potential error rates.

To address these concerns, the sample folding paths were designed to be as unbiased as possible, aiming to identify any algorithmic flaws. However, it is important to note that these empirical tests do not validate the correctness of any specific implementation. While the focus is not on practical applications, efforts were made to create paths that resemble biologically relevant input scenarios.

Each dataset in this study consists of fixed-length nucleotide sequences ranging from 100 nt to 600 nt, along with two respective secondary structures. Similar datasets with consistent basic settings were used throughout the thesis. The majority of sample datasets utilized in this thesis were generated using the following strategy:

1. Generate a random sequence string σ of letters $\{A, G, C, U\}$ and length N
2. `RNAsubopt`(Lorenz et al., 2011) was used to generate two Boltzmann-weighted sample secondary structures A and B ("`-p 2`" option)
3. Perform gradient walks for both structures A and B such that they correspond to local minima (using the `ViennaRNA` api)
4. Structure pairs (A, B) with a base-pair distance $d(A, B) < 10$ were filtered out since solutions to short folding paths are trivial

3.3 Findpath Extension: Divide-and-Conquer

3.3.1 Theory

`Findpath` with its original runtime complexity of $O(mn^3)$ is an effective folding path heuristic which yields approximate solutions within reasonable runtimes. This thesis will investigate several promising approaches that aim to enhance its performance, resulting in shorter runtimes, enabling folding path computations for longer sequences, and allowing the computation of more accurate solutions for time-dependent computations.

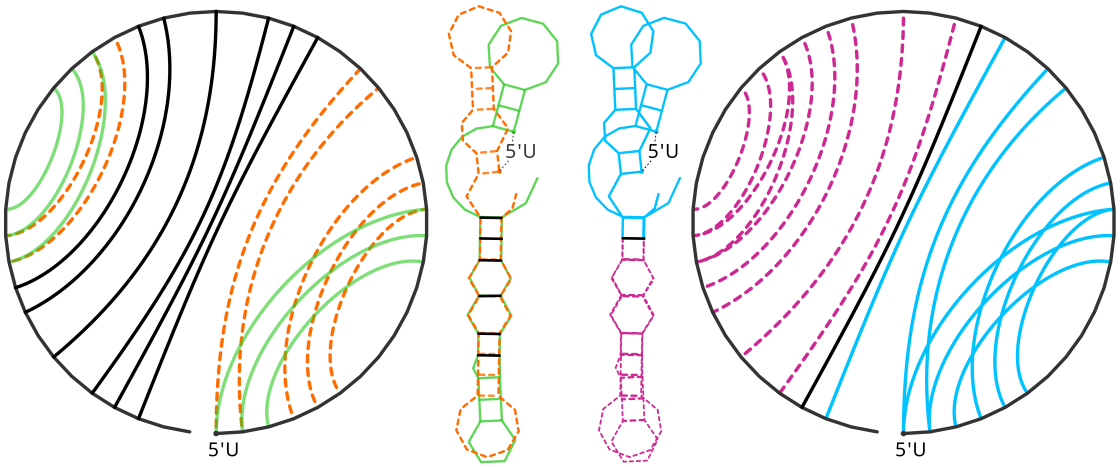
The following part of this thesis focuses on the **divide-and-conquer** extension. It involves dividing *direct* RNA folding paths into smaller subpaths whenever the path can be split at constant base pairs. The underlying assumption is that a combined energy barrier can be inferred from these subpaths. Unfortunately, simply concatenating these subpaths does not yield an optimal solution. The reasoning behind this phenomenon will be discussed in subsequent sections.

The first description of this idea can be attributed to Morgan and Higgs (1998), who coined the term "frozen pair" which denotes constant base pairs which are present in both initial and final structures of a *direct* folding path. Constant pairs can split both structures into independent sections, i.e. subpaths within these sections are geometrically confined. Morgan and Higgs concluded that concerning the energy barrier, regions act independently and only the region with the largest sequence length has to be considered. This conclusion, however, only considers the simplified Nussinov energy model (Nussinov and Jacobson, 1980).

Breaking down the folding path problem into multiple more manageable subproblems can lead to significant efficiency improvements, given that any additional overhead from the process is insignificant. In addition to lower computational runtimes per section, algorithms like `Findpath` also have the tendency of being much more accurate for shorter paths, which results in more accurate estimations of combined energy barriers. Divide-and-conquer algorithms also tend to scale well with parallelization.

Formally, a constant base pair (i, j) induces an independent loop section $i + 1$ to $j - 1$, the remaining section forms a combined loop from 1 to $i - 1$ and $j + 1$ to N, as visualized in Figure 3.1. Formed sections are mutually inaccessible, no base pairs (i, j) will cross constant base pairs during the folding process. Any constant pair (i, j) is a viable position for a splitting process in a recursive fashion, assuming all subpaths have a base-pair distance > 1.

In this section, the viability and pitfalls of Morgan and Higgs' approach in conjunction with the Turner energy model (Mathews et al., 2004) will be examined. The



UCGGACAGAAACGGUUGAGGGGCGGCGGGAAGCGAUUGUUCUAGGCGCGG
A ((.(((.......)))..)).((.(.((((.......)))).).))..
B (((........)))........((.(.((.(((....))))).).))..

Figure 3.1: Illustration of structures A and B of a sample RNA folding path with applied divide-and-conquer scheme: The initial and final structures are shown in orange (with dashes) and green, respectively (left). Constant base pairs are depicted in black. To split the path into two sections, a constant base pair is selected (right). This constant base pair is present in both sections of their respective subpaths (magenta and blue sections).

divide-and-conquer `Findpath` extension can be used in conjunction with other extensions, which are presented later.

3.3.2 Estimating Energy Barriers

Steven Morgan and Paul Higgs defined the largest independent region in terms of nucleotide distance as N_{eff} with respective folding path and energy barrier. This relies on the simplification that the largest variation in terms of base-pair distance occurs in this region. Within the Nussinov energy model, it was predicted that the overall energy barrier should usually be related to this largest section (Morgan and Higgs, 1998).

For the divide-and-conquer `Findpath` extension, it is inevitable to merge subpaths recursively into a combined folding path, i.e. the combined energy barrier cannot be inferred greedily or by simply concatenating elementary moves from subpaths (Figure 3.2). The merging procedure is an optimization problem with required assumptions about its input paths. The most basic assumption is that one optimal

(lowest possible energy barrier) subpath per section is sufficient to merge all of them into a globally optimal folding path.

Summarized, this basic assumption does not hold for all folding paths: Not all optimal subpaths are equal, and can be merged into the best possible path. In the context of `Findpath`, this means that there are instances where such a merged path cannot converge to optimal energy barriers at high search widths. However, in select instances, even suboptimal energy paths would be required as merging input to reconstruct optimal direct paths as output. Detailed results and the developed merging procedure will be discussed in the following sections.

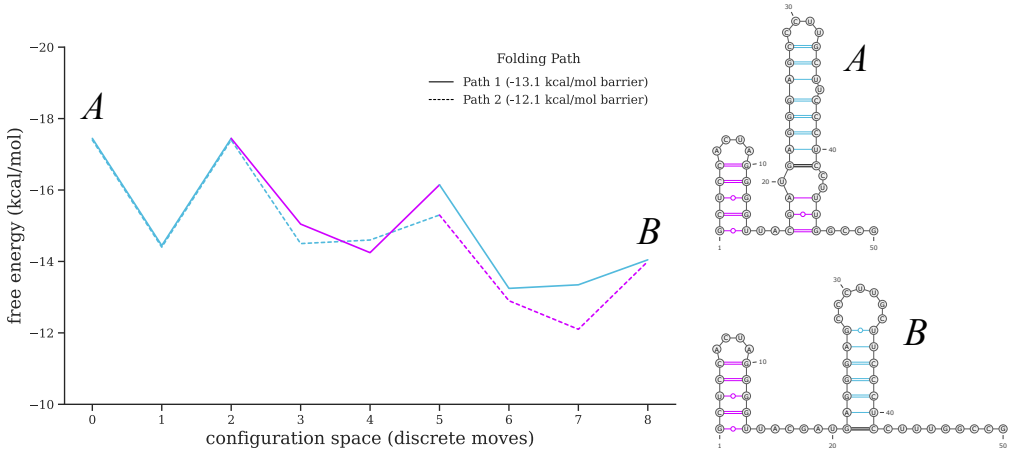


Figure 3.2: Merging subpaths as an optimization problem. Sample folding path with magenta and blue sections where the total energy barrier can be inferred by merging both optimal subpaths. Merging paths is an optimization problem: the greedy solution (not shown) and the concatenated solution (dashed path 2 with highest energy structure at move 7) have higher energy barriers than the optimal solution (path 1). Folding path see appendix Fig. 1.

3.3.3 Identifying Path Sections

As the first step of the `Findpath` version with the divide and conquer scheme, its independent sections with respective subpaths have to be defined. For this, a recursive section scanner was developed. A section boundary can only occur at base-paired nucleotides if both structures A and B have at position (i, j) identical (constant) pairs. Within these defined boundaries may be multiple nested helices and loop regions, which can also be the starting point of nested sections.

It proceeds as follows: If a constant base pair (i, j) is found a new inner loop section is defined from position i to j, if this section has a distinctive (non-zero)

set of elementary moves compared to moves within the remaining positions (outer section). Sections are not allowed to consist of only unpaired bases and constant base pairs.

Implementation-wise (Algorithm 3.1), the initial function call is SectionScanner(A, B, $k = 1$, $l = N$). A' and B' are cut structures from the inner section i to j, such that all base pairs outside i, j are removed. A'' and B'' corresponds to structures from the remaining outer section. To guarantee a valid split, it is necessary for both sections to contain at least one elementary move within their geometrically confined regions, i.e. base-pair distances $d(A', B') > 0$ and $d(A'', B'') > 0$. Nested sections are resolved with a recursive function call. A typical output notation can be seen in Figure 3.3.

Algorithm 3.1: SectionScanner (Input: A, B, k, l)

```
1  Sections ← [k, l]                    ▷ initial list of sections is [1, N]
2  for i ← k to l do
3      if pos. i forms base-pair with same pos. j in both A and B then
4          d_inner = d(A', B') (structures restricted to pos. i to j)
5          d_outer = d(A'', B'') (structures restricted to pos. k to i and j to l)
6          if d_inner > 0 and d_outer > 0 then
7              Sections' ← SectionScanner(A, B, i, j)
8              Sections.insert(Sections') at position 2
9                                   ▷ [k, l] becomes [k, [i, ..., j], ..., l]
10             i ← j + 1
11 return Sections
```

This procedure greedily maximizes the number of nested sections by choosing to split the first constant base pair. A completely different approach would be to balance resulting sections, e.g. splitting such that inner and outer sections ideally have equal-length subpaths. Such a procedure would be preferable if the subsequent merging would be time sensitive. However, since the implemented merging has a negligible runtime, the idea of balanced sections was discarded.

Similarly, sections can be also split at the exterior loop level. This split does not rely on the presence of constant base pairs present in both structures. For this variant, all loops at the base level form initial sections, assuming subpaths of distance > 0. Each base section is then subsequently processed by algorithm 3.1. In cases such as the example of Figure 3.3, this split can result in an increased number of sections.

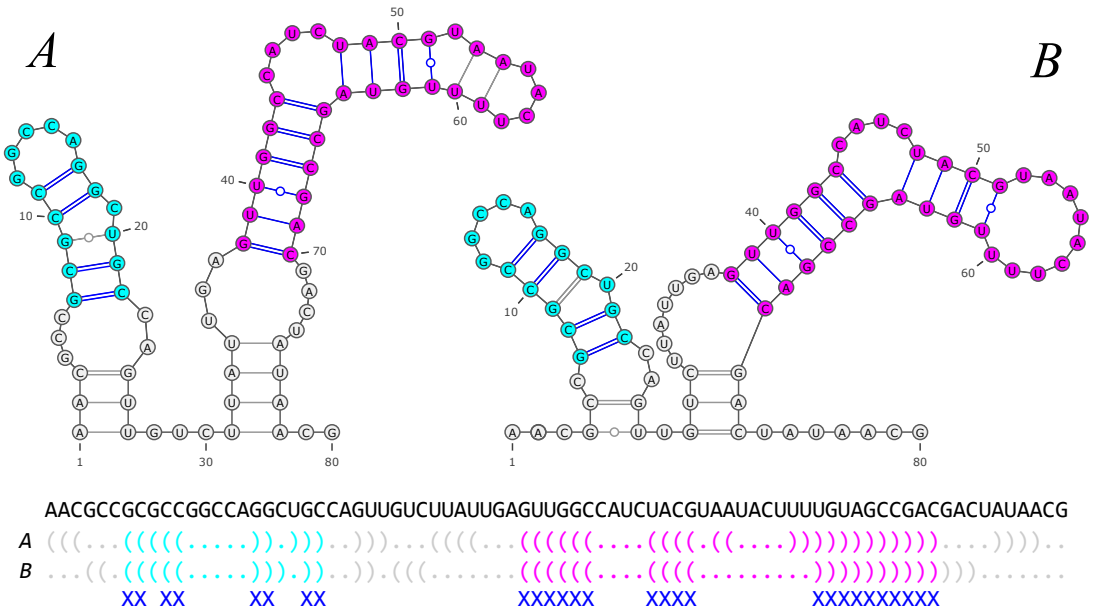


Figure 3.3: Exemplary folding path, which can be split into 3 independent subpaths with the recursive section scanner approach. This split corresponds to the notation [1, [7, 22], [38, 70], 80]. All constant base pairs are marked in blue and with an X in the dot-bracket notation. In addition, a split between nucleotides 27 and 28 is only possible if the splitting procedure also considers the exterior loop level, yielding 4 subpaths for this example. This visualization was created with VARNA (Darty et al., 2009).

3.3.4 Results: Occurrence of Sections within Random Paths)

Using the recursive section scanner on the initial set of random sequence datasets gives us a first indication of which scenarios the divide-and-conquer method could be beneficial. Any additional merging overhead will not be considered here. For these results, the polynomial `Findpath` runtime has to be considered: even comparatively small path length reductions have large potential benefits.

In the synthetic test dataset, it is noteworthy that on average one large section dominates the section-wise base pair distances, i.e. a 9:1 split for subpath distances is much more likely than a 1:1 split, assuming short 100 nt sequence paths (Figure 3.4C). The subpath length of the largest section dominates the total runtime due to Findpath's near-polynomial scaling with respect to the path length n. For the synthetic dataset, which may not reflect the typical use case, the divide-and-conquer method is unlikely to improve the runtime for short sequence folding paths (< 100 nt) on average.

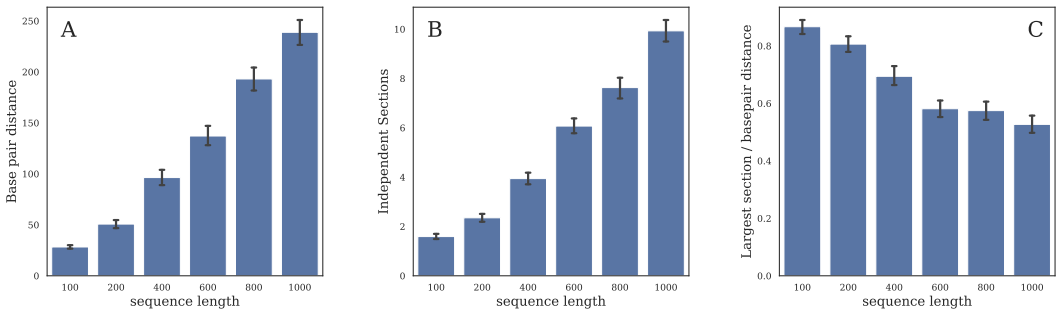


Figure 3.4: Section occurrence of randomized folding path inputs, synthetic datasets with 1000 paths per length: The number of independent sections and total base-pair distances increases with n on average (see left Figures A and B). The largest section in terms of relative subpath distance, which dominates the final `Findpath` runtime, is comparably long in paths of short sequences (see right Figure C).

3.3.5 The Subpath Merging Problem: Path Permutations

As discussed earlier, the overall energy barrier cannot be simply inferred from subpath barriers, a merging process had to be developed which accepts two folding paths as input. Nested sections have to be merged recursively. The overall energy barrier has to be inferred from the final merged folding path.

The first step is solving the most basic task: two sections with one optimal subpath per section. The permutation of elementary moves within sections shall remain unchanged during the merging process. The optimization problem involves finding the optimal sequence of n consecutive decisions between moves from both subpaths. No cycles are permitted, the joined path length n is equal to combined base pair distances $d_i + d_o$ (shortened notation for inner and outer path distances). Merging two subpaths is a combinatorial problem with

$$\binom{d_i + d_o}{d_i} = \frac{(d_i + d_o)!}{d_i! d_o!} = \frac{n!}{d_i! d_o!}$$

possible ways to combine subpath moves: A merged path has d_i positional choices for one path and d_o choices for the other path. Thus, the combined path chooses d_i (or d_o) choices from $d_i + d_o$ choices.

The number of possible combinations grows exponentially with n: If both input paths have a path length of 10, there are already 184,756 ways to merge both paths. A naive approach would result in an exponential runtime, and a greedy approach is unlikely to yield the optimal path.

3.3.6 Merging Subpaths: the Dynamic Programming approach

Two subpaths can be merged efficiently since the merging problem can be broken down into smaller sub-problems, which is a typical application for a dynamic programming approach. The key concept is that with increasing path lengths, the saddle energies can only ever grow. The solution to the optimal merged path is recursively nested in smaller problems, which can be computed by gradually increasing the path size by step-wise addition of elementary moves. The smallest possible problem is a single decision between elementary moves of both subpaths, where the only solution is to take the lowest energy move.

The path merging problem falls in line with basic problems with the **optimal substructure property**. Typical similar well-known problems within the literature are the **Longest Common Subsequence** problem, or the **Manhattan Tourist Problem** (Figure 3.5). Various more complicated problems have a long-lasting history in bioinformatics, starting with the global sequence alignment problem by Needleman and Wunsch (1970).

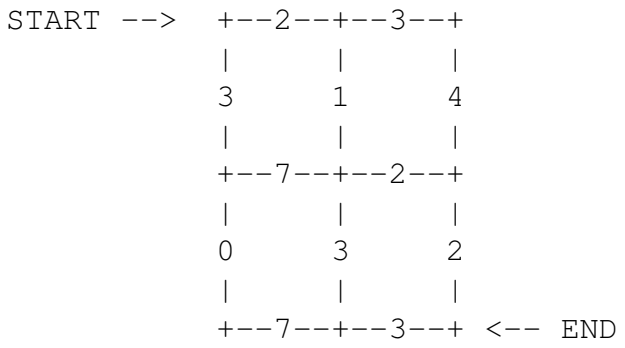


Figure 3.5: The Manhattan Tourist Problem: A tourist may only move east and south along city blocks. The goal is to maximize the number of attractions along the route (numbers along the roads), without computing all possible path permutations. Dynamic programming can be used to find the best possible path with a $O(mn)$ runtime.

A dynamic programming table M is created of size $d_i \times d_o$, which keeps track of maximum saddle energies. Indices i, j in M refer to taken moves from both subpaths, such that $M_{0,0} = E(A)$ (free energy of initial structure). $S_{i,j}$ is an intermediate structure with i and j moves from both subpaths. The recurrence relation only considers the best possible next move of either subpath and the free energy of the current intermediate:

$$M_{i,j} = max \begin{cases} E(S_{i,j}) \\ min(M_{i-1,j}, M_{i,j-1}) \end{cases}$$

The optimal saddle energy will be the final matrix cell M_{d_i,d_o}. The best possible path can then be recovered via backtracking (example in Table 3.1).

0	3	3	3	3	3
2.4	5.4	3	5.3	5.2	4.5
3.2	6.2	3.2	6.1	6	5.3
3.2	4.3	3.2	4.2	4.2	4.2

Table 3.1: Calculating the combined energy barrier by merging 2 subpaths using dynamic programming. This Table displays the DP matrix M of sample subpaths of Figure 3.2 with dot-bracket notation in Appendix Figure 1 with a final energy barrier of 4.2 kcal/mol. Note that in this example M has barrier and not saddle energies, as described earlier. The optimal combined folding path obtained by backtracking from the last cell is highlighted in yellow.

3.3.7 Validity of Input Subpaths

Optimal input paths

For the presented dynamic programming merging approach one optimal direct subpath per section was selected per section. However, **multiple optimal folding paths** may be present between two structures since not all move permutations affect the resulting energy barrier. Given the fact that path permutations grow exponentially with respect to n, the amount of optimal paths has to grow similarly. Therefore, heuristic algorithms cannot track all optimal folding paths.

A natural assumption would be that any optimal subpath is sufficient to derive the best overall folding path, but this can be refuted with a simple example which is presented in Figure 3.6: Here, the combined folding path can be reconstructed by merging 2 subpaths as input, however, only subpath A1 merges with B (see appendix 2) into an optimal combined path. Both subpaths A1 and A2 have identical individual energy barriers, reached after a single move. A1 has a local minima intermediate (move 4, 1.6 kcal/mol), which is vital in the merging process.

Optimality criteria for folding paths are an important concept in the divide-and-conquer method. Besides finding paths with the lowest energy barriers, there are cases where barriers are less important than **local minima**. The presented sample path should be considered as a toy example: for larger sequences with longer folding paths, these local minima are not straightforward to define. Optimality criteria

for merging depend on the other subpath which is unknown to the pathfinding heuristic.

Since not all optimal folding paths are trackable, the convergence to the lowest energy barrier may not be achieved by **any** heuristic folding algorithm that implements a merging procedure. The primary objective should always be to find subpaths with the lowest energy barrier, with all other optimality criteria considered secondary objectives. An analogous concept can be seen in the design of two mechanical cogs, where they are independently designed without knowledge of each other's dimensions. In such a case, there is no guarantee that the teeth will mesh properly in the end.

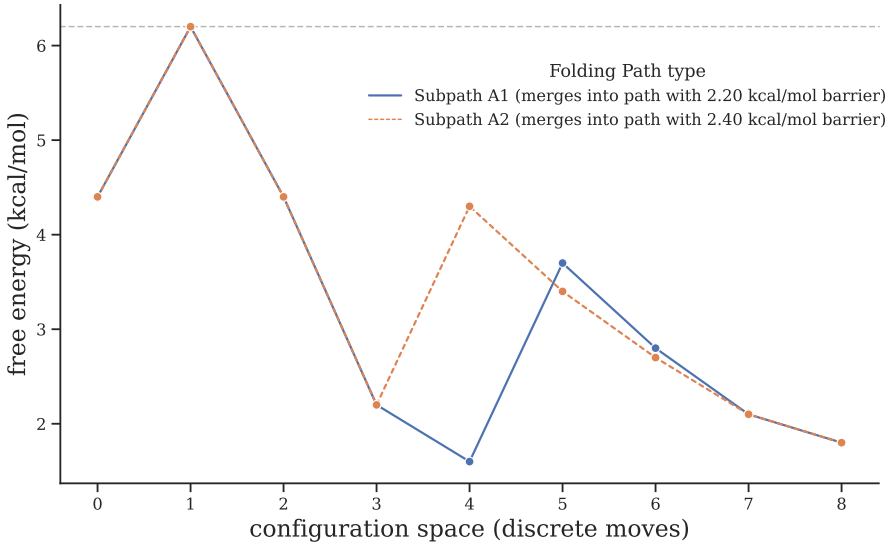


Figure 3.6: Subpaths as merging input. In this example, A1 and A2 from section A are both optimal (6.2 kcal/mol saddle energy), yet only A1 can be merged with subpath B (section B see appendix 2) into an optimal combined path. A2+B merges into a path with a 0.2 kcal/mol higher energy barrier. In total, section A has 156 optimal subpaths, while section B has only 1 optimal subpath. Only 4 optimal subpaths from section A merge optimally with the single path from B.

Suboptimal input paths

In rare occurrences within the sample dataset, subpaths with optimal saddle energies are not sufficient as merging input. Various sample paths were exhaustively tested by enumerating all optimal input paths, and none of them merged into an optimal

combined path. In such cases, suboptimal paths are sometimes required, typically with one subpath being non-optimal. For these instances, the presence of a local energy minimum at a precise location supersedes the importance of a low energy barrier.

These findings emphasize the importance of a large number of randomly generated folding path inputs: Within the sample dataset, this affects less than 1% of all input paths. It is important to distinguish between the problem `Findpath` is supposed to solve from the practical applications where folding path heuristics are applied.

In general, there are no known indications in which scenarios these suboptimal input paths are required. As described earlier, the validity of a single input path depends on the second path. `Findpath` can be modified to yield paths predominantly with local minima. In a few affected examples, a `Findpath` variant produced local minima paths that were optimal for merging. However, the changes made to the algorithm were not effective as expected, as this modified version often produced paths with higher saddle energies compared to the original. Any adaptation inevitably affects the main objective of finding paths with low saddle energies.

Therefore, the idea of modifying Findpath's optimality criterion was abandoned. Notably, when dealing with longer sequence inputs, paths are frequently merged recursively, and it should be noted that not all merging outcomes will have an impact on the final combined energy barrier.

3.3.8 New Input Path Strategy

The original `Findpath` algorithm without divide-and-conquer strategy will converge to an optimal result by increasing the search width m such that an exhaustive search is performed. With a merging approach, however, regardless of which path heuristic is used for subpaths, there can be no guarantee of convergence.

Nevertheless, the strategy of merging subpaths is still promising from a heuristic point of view since computations are often limited by runtime constraints, meaning that obtaining the theoretical optimal lowest possible energy path can be of no practical value. The divide-and-conquer `Findpath` variant has to produce lower energy barriers on average with typically used search widths m. Scenarios, where this approach is worse than the original heuristic, should be kept to a minimum.

Efficient Method for Finding Multiple Input Subpaths

As noted before, searching for specific optimal paths or even suboptimal paths conflicts with the optimality criterion to find paths with the lowest saddle energy. Searching for additional paths is therefore always a trade-off. In various test datasets, the following effect was observed: Suboptimal paths, which are potentially optimal for merging, are often found in earlier `Findpath` passes (see Algorithm 2.2 search strategy where `Findpath` gradually ramps up m in multiple passes). Therefore, the overarching strategy is to retain folding paths from all passes with the merging procedure described in the next section. Due to the developed DAG-processing step, this strategy is not computationally expensive.

The effects of this input path strategy can be seen in Figure 3.7, where in *strategy A* a single optimal subpath is merged with the divide-and-conquer approach, and *strategy B* makes use of multiple input paths with the merging procedure described in the next section. Notably, even the single input approach shows a shift to lower average saddle energies on average, but a significant fraction of sample folding paths had higher saddle energies compared to the original Findpath implementation. The search width m is constant in both cases.

Additional tests were conducted at higher search widths: Both presented variants converge to the optimal result with a good success rate. The multiple input paths variant is a valuable improvement which further reduces the fraction of non-convergent paths (see later results).

3.3.9 Merging Subpaths: the Heuristic Approach

The first initial prototype with dynamic programming as the merging procedure showed promising results for various input datasets. However, this method suffers from the inherent problem of only accepting two input paths: Processing all available input paths step-by-step is computationally infeasible, especially considering the recursive nature of subsequent merging operations. Additionally, the backtracking process (DP) has to be considered, as it potentially produces a large number of optimal paths which have to be recursively processed.

This shortcoming led to the development of a graph-based breadth-first search merging algorithm, which works in a very similar fashion to the original Findpath algorithm. The key difference from the original algorithm is that while the original algorithm performs a move validation for each tracked intermediate structure, the merging procedure in this variant always tracks two concurrent nodes. These nodes have a sparse number of outgoing edges, and each edge corresponds to a valid move. Notably, this algorithm has the same heuristic design with a defined search width

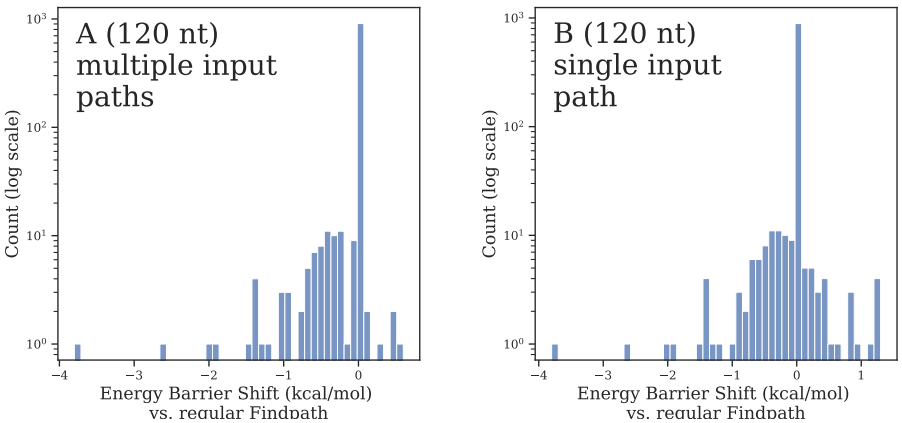


Figure 3.7: Divide-and-Conquer `Findpath` (Multiple input paths vs. single path as merging input), compared with regular `Findpath`. $m^* = 2$, 1000 sample paths with nucleotide lengths of 120. Mean saddle energy shift vs `Findpath` are -0.047 kcal/mol (A) and -0.033 kcal/mol (B), respectively. In strategy A, single input paths were used for testing the divide-and-conquer merging procedure, which was implemented to generate a single merged path via backtracking. If there are more than two sections, the recursive merging procedure shown in Figure 3.8 was applied. Strategy B uses the heuristic merging procedure with multiple input paths, as described in the section 3.3.9.

parameter, thereby no guarantee that the merging process will be optimal for all instances.

Heuristic Merging Method: DAG-processing Step

Since the idea of using more than one subpath per section seemed promising, a method was developed that reliably merges multiple paths at once. The first step in this procedure is the DAG-processing step: All path redundancies (intermediate structures which occur more than once) are removed while merging them into a single directed acyclic graph (DAG) with nodes and edges, corresponding to intermediate structures and elementary moves, respectively.

The resulting graphs are sparse with nodes having on average less than 2 outgoing edges in observed sample paths: Optimal paths are mostly similar with only minor deviations. These minor deviations, however, should not be disregarded as these are often the key component for optimal merging outcomes. Suboptimal paths, which are added to the graph, are often dissimilar to optimal paths with fewer shared intermediates.

Formally, each DAG $G = (N, E)$ consists of a set of nodes N and a set of edges E. Each node N corresponds to an intermediate structure and edges E connect two nodes with an elementary move (i, j). In the merging process, two DAGs are always considered, and they are referred to as G_1 and G_2 (as shown in the merging algorithm in the following section). Each DAG has a defined base-pair distance $d(G) = d(A, B)$. Implementation-wise, all intermediate structures $S_0, ..., S_n$ from all input paths $P_i(A, B)$ per Findpath call are identified with a unique hash-value and associated node. Each node has a simple adjacency-list-based set of outgoing and ingoing edges.

The current version retains all available input paths as the computational runtime for the whole merging process is negligible compared to the runtime of all combined Findpath calls. The DAG-processing step only proceeds if required for subsequent merging, as seen in Figure 3.8.

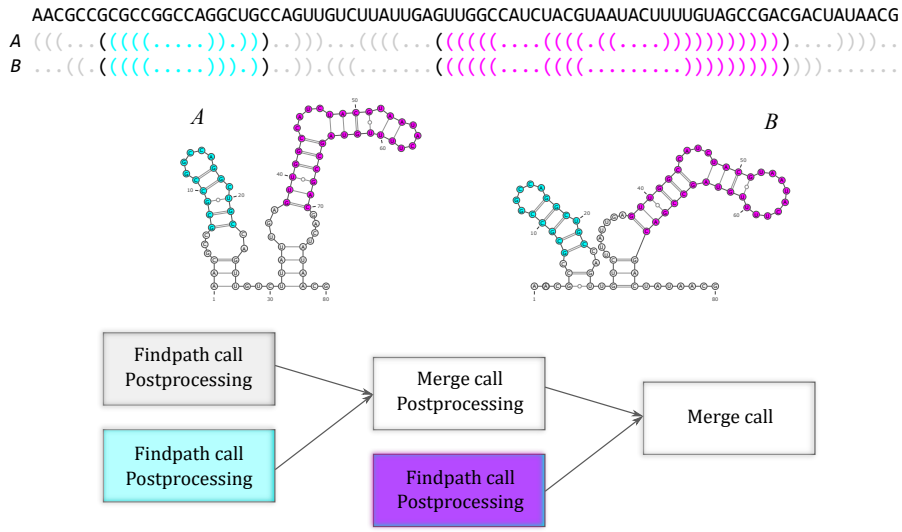


Figure 3.8: Recursive Nature of the Merging Procedure. The example from Figure 3.3 has 3 interior loop sections with associated Findpath and DAG-processing calls. Each merging step is limited to two input sections.

Heuristic Merging Method: Breadth-first Search Algorithm for Merging

As previously indicated, the best-performing merging procedure tested is a heuristic breadth-first search algorithm, which will be covered in this section. Here, the algorithm (pseudocode see 3.2) takes two DAGs G_1, G_2 with respective nodes N_1, N_2 and edges E_1, E_2 from two sections as input and generates folding path outputs identical to those of the `Findpath` algorithm. A search width parameter m_{merge} is required which has the same functionality as in `Findpath`.

The current structural intermediate in a path corresponds to a node tuple, e.g. $(A1, A2)$, as seen in Figure 3.9. In each iteration, the algorithm moves one step closer to the final merged structure B. For every retained intermediate structure, new candidate intermediates are generated which are outgoing edges from both nodes of the tuple. The following sorting step is identical to `Findpath`.

The same overarching search strategy is used as the original `Findpath` algorithm as described earlier in section 2.4.3: Starting with greedy search passes, the search width m_{merge} is gradually increased while updating the threshold energy E_{max}. Both forward and backward passes are computed. The merging procedure generates identical output paths as `Findpath`, which necessitates a DAG-processing step for subsequent recursive merging steps (see Figure 3.8).

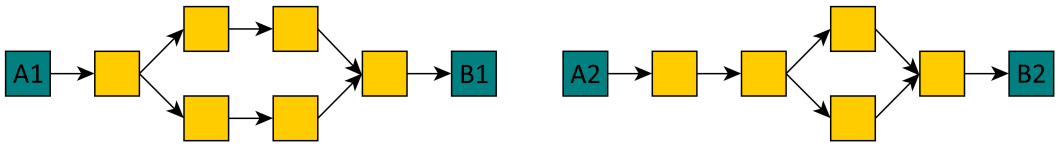


Figure 3.9: Merging two folding paths with Breadth-first Search by processing two DAGs. Any intermediate structure corresponds to one node per DAG as a tuple, e.g. $(A1, A2)$. Candidate moves are generated from the edges of both DAGs. Each candidate move per iteration moves one step closer to either B1 or B2.

3.3.10 Results: Performance of the Divide-and-Conquer extension

For subsequent results, the search width m is automatically calculated, referenced by the search width multiplier $m^* = m/n$. This approach is advantageous for the divide-and-conquer variant, as the section-wise base-pair distances, which collectively add up to n, are unknown to the user. Therefore the search width is dynamically adjusted per section. Similarly, for the BFS-merging procedure, the merge search width m_{merge} is adjusted with a defined m^*_{merge}.

The optimal energy barrier for most tested direct paths is unknown, as it cannot be computed within a reasonable timeframe for longer sequence paths. Therefore, a convergent energy barrier is defined as the lowest tested energy barrier among all obtained results from a single path, including all tested `Findpath` variants at various search widths.

It is important to consider the heuristic nature of `Findpath` when interpreting the presented results. In the majority of tested cases, all variants of the developed extensions yield identical outcomes at high search widths. A higher search width results in potentially lower energy barriers. However, this does not imply that

Algorithm 3.2: DAG-Merging (Input: $\sigma, G_1, G_2, m, E_{max}$)

$G_1 = (N_1, E_1)$; $G_2 = (N_2, E_2)$
$n_{final} = d(G_1) + d(G_2)$ ▷ number of iterations (both base-pair distances)
$S_0 \leftarrow (N_{1_0}, N_{2_0})$ ▷ Starting intermediate S_0 refers to both initial nodes
$S \leftarrow [S_0]$ ▷ List of intermediates with single entry
$E_{s_0} \leftarrow E(S_0)$; $E_s \leftarrow [E_{s_0}]$ ▷ initial list of saddle energies E_s
for $n \leftarrow 1$ **to** n_{final} **do**
  $E'_s \leftarrow []$; $S' \leftarrow []$ ▷ empty lists of candidate structures
  foreach S_m *in* S *and* E_{S_m} *in* E_s **do**
    $S_m = (N_1, N_2)$ ▷ Each intermediate refers to two nodes
    foreach *outgoing edge (i,j) in* N_1 *and* N_2 **do**
      Perform candidate move $S_m \xrightarrow{(i,j)} S'_m$
      if $E(S'_m) < E_{max}$ **then**
        $E'_{s_m} \leftarrow \max(E_{s_m}, E(S'_m))$ ▷ Update saddle energy
        E'_s.insert(E'_{s_m})
        S'.insert(S'_m)
  $E_s \leftarrow m$ best E'_s ▷ Sorting Step: Retain m best lowest energy paths
  $S \leftarrow m$ best S'
return $\min(E_s)$

certain variants are inherently superior at any given search width m, as shown in the left Figure of 3.10. In rare instances, increasing the search width m can lead to adverse effects. The divide-and-conquer variant exhibits this behaviour, along with additional inconsistencies stemming from the recursive nature of the merging process, as seen in the right Figure of 3.10.

Shift in Energy Barrier

Tests conducted with $m^* = 2$ reveal minimal changes between both variants for short sequences, as only a fraction of path inputs can be split into subpaths. However, for larger sequences, significantly larger energy barrier shifts are observed. These findings are illustrated in the barrier shift histograms shown in Figure 3.11, and average energy barrier shifts are documented in Table 3.2. Since $m^* = 2$ is a relatively low search width setting, instances, where the divide-and-conquer variant yields higher energy barriers, can be mostly attributed to scaling inconsistencies, as discussed earlier and shown in Figure 3.10.

In the majority of sample paths, where the divide-and-conquer variant results in higher energy barriers at $m^* = 2$, the energy barriers will eventually converge to

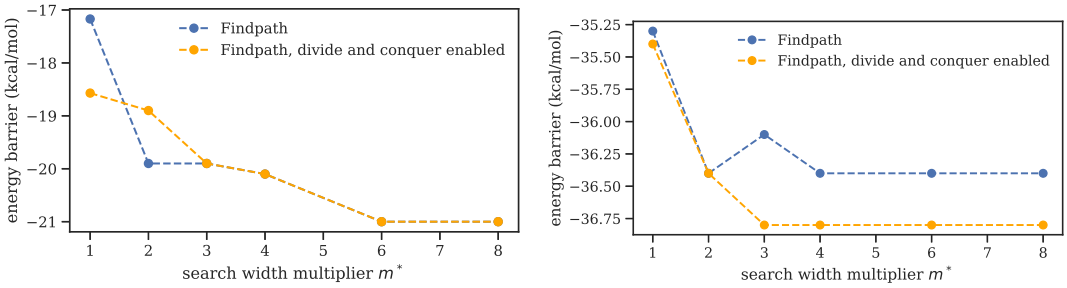


Figure 3.10: Two distinct sample input folding paths, with and without enabled divide-and-conquer extension. Depending on the chosen search width, either variant has the potential to outperform the other. In rare cases (right), where suboptimal input paths may be necessary to achieve convergent energy barriers, the divide-and-conquer variant does not produce optimal outcomes or only under specific conditions.

the optimal result when the search width is increased further. However, there will always be a small fraction, with percentages depending on the dataset, that never reaches the optimal result when the extension is enabled. As mentioned earlier, the multiple input subpath strategy with BFS-based merging is a noticeable improvement to the merging scheme, but it is not without flaws.

Dataset	100 *nt*	200 *nt*	300 *nt*	400 *nt*	600 *nt*
DC extension	-0.01	-0.21	-0.52	-1.11	-2.65

Table 3.2: Average energy barrier shift in kcal/mol of the divide-and-conquer extension vs. regular `Findpath`. Tested in sample datasets ranging from 100 to 600 *nt* sequence lengths (1000 paths per length) at search width $m^* = 2$.

Notably, this extension is optimized to generate near-optimal results with reduced computational demand. At any given search width, the divide-and-conquer variant has a much higher chance to produce convergent results, as depicted in Figure 3.12. Generally, the findings suggest that the BFS-based `Findpath` heuristic is not optimal for efficiently computing convergent energy barriers. To compute convergent energy barriers, a modified version of `Findpath` that does not discard any structure per iteration would be necessary. This invalidates the employed memory optimization adaptations, which rely on a defined search width m to preallocate memory. An optimized pathfinding algorithm based on Dijkstra's algorithm for optimal convergent paths is likely to be a more suitable choice.

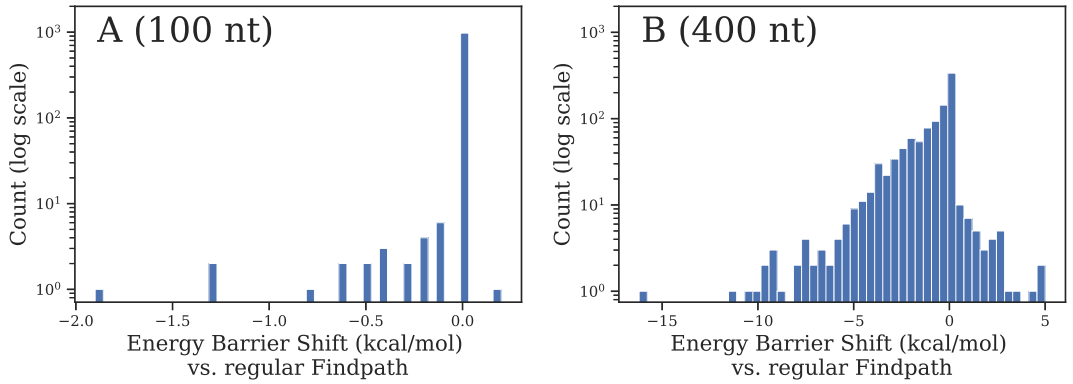


Figure 3.11: Sample folding paths with sequence lengths 100 *nt* (A) and 400 *nt* (B), 1000 paths per dataset, tested at $m^* = 2$. Shown are energy barrier shift histograms of the divide-and-conquer variant vs. regular Findpath. Average energy shifts see table 3.2.

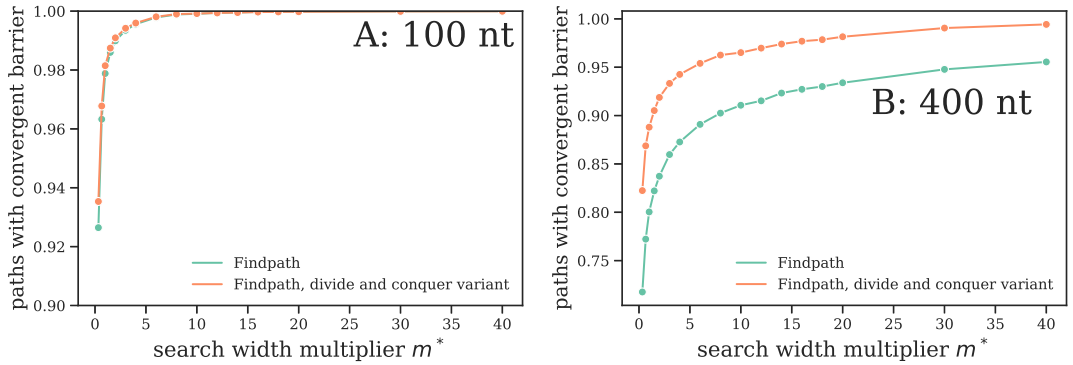


Figure 3.12: Convergence analysis of datasets with sequence lengths of 100 *nt* (A) and 400 *nt* (B), consisting of 1000 paths per dataset. The y-axis represents the percentage of paths within each dataset that achieve optimal energy barriers at various search widths (x-axis).

BFS-Merging Efficiency

As previously touched upon, the merging process with two DAGs as input has comparably insignificant computational demands compared to all combined `Findpath` calls per section. As seen in Figure 3.13, required merging search widths are comparably low: At $m^*_{merge} = 0.8$ already 99.9% of all input paths converge. This supports the assumption that the merging algorithm works fast and reliably.

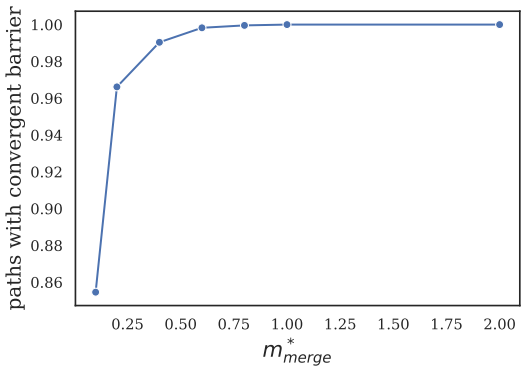


Figure 3.13: Required merging search width for convergence. This test only involves a dataset of length 400 nt with 1000 input paths. The regular search width is kept constant ($m^* = 2$). A convergent energy barrier is defined as the outcome with the lowest obtained energy barrier.

Runtimes

Figure 3.14 displays runtime comparisons between the regular `Findpath` algorithm and the divide-and-conquer variant. In these results, the optimized `Findpath` algorithm serves as the baseline. Details of these general optimizations will be discussed in a later section. The divide-and-conquer scheme exhibits a substantial speedup for longer sequences, with the break-even point occurring around 200 nt. This extension synergistically complements all other developed extensions, which will be in introduced subsequent sections. A runtime comparison of all the developed extensions is presented in the final chapter.

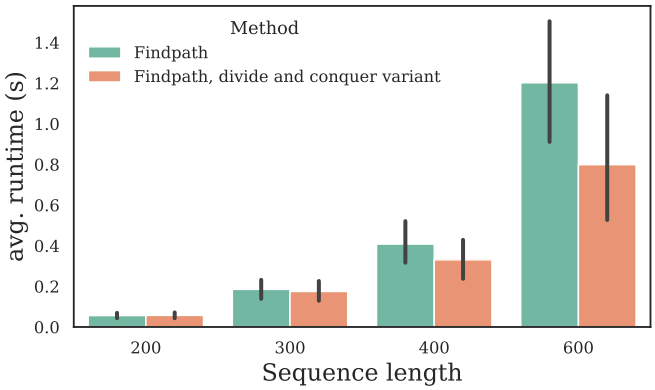


Figure 3.14: Runtime comparison with and without divide-and-conquer extension. Random folding paths of defined sequence lengths, with 1000 paths per length. Typically, the extension enables a speedup at sequence lengths above 200 nt.

3.4 Findpath Extension: Constrained MFE State

3.4.1 Theory

This extension to the Findpath heuristic draws inspiration from the work of Bulteau et al. (2021), who developed multiple exponential-time algorithms for calculating optimal RNA folding paths. These approaches are related to prior research by Thachuk et al. (2010), where direct paths were computed between pseudoknot-free structures, utilizing the simplified Nussinov energy model (Nussinov and Jacobson, 1980). Among these algorithms, the best runtime was achieved by a parameterized algorithm, which requires $O(n^{\rho+1})$ in space and $O(n^{\rho+2})$ in time. ρ corresponds to the energy range between the maximum allowed energy and minimum free energy.

One of the algorithms introduces a **MFE state**: A restricted minimum free energy structure, where base pairs are limited to base pairs from A and B (initial and final structures in a folding path). If the MFE state $\neq A$ and $\neq B$, it is defined as **mixed MFE state**. A sample path with MFE state as a checkpoint, which also represents the lowest energy state, is depicted in Figure 3.15.

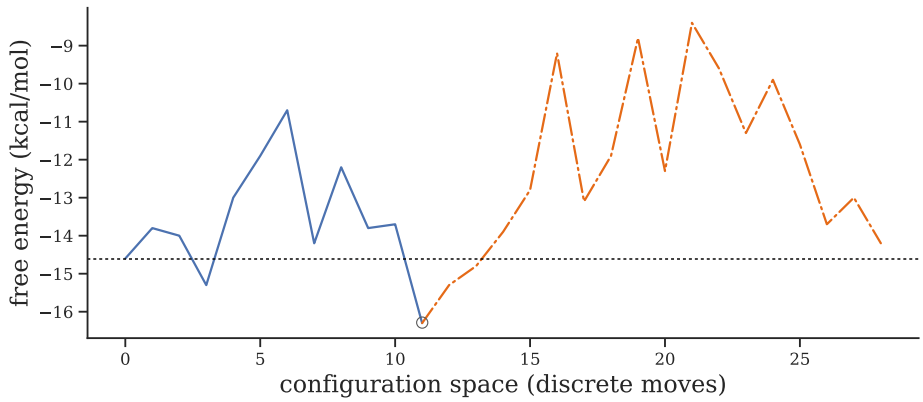


Figure 3.15: Single RNA folding path with mixed MFE state (lowest energy intermediate) as a checkpoint. The computation of both folding paths $A \rightarrow$ MFE (blue) and MFE $\rightarrow B$ (orange) with subsequent concatenation is an efficient way to compute a combined folding path. This method is only applicable when the MFE state has a lower energy than either A or B (mixed MFE state).

According to the introduced separation lemma (Bulteau et al., 2021), a mixed MFE state is one of the *direct* path intermediates of optimal folding paths ($A \rightarrow$ MFE state $\rightarrow B$), without loss of generality. In the energy landscape of the Nussinov energy model, the MFE state is characterized by the maximum number of base pairs in the direct-path corridor, resulting in the lowest energy.

An open question was whether this procedure can be adapted for folding path heuristics based on the Turner free energy model (Mathews et al., 2004). A mixed MFE state can be computed for any direct folding path using the ViennaRNA package, which includes hard and soft constraints based on an updated folding grammar (Lorenz et al., 2016). In the global MFE computation, hard constraints are applied, allowing only base pairs that occur in structures A and B, while prohibiting all other pairs outside the direct-path corridor. Multiple MFE states can exist, such as the initial structure and an additional mixed state with the same free energy.

3.4.2 Implementation

The implementation of this method involves a simple preprocessing step that computes the MFE state through constraint folding and verifies if the MFE state is mixed (not equal to A or B). If possible, both split paths are computed independently as separate Findpath calls. No further recursions are possible. The divide and conquer extension can be used to compute both split paths more efficiently.

3.4.3 Results

The separation lemma does not apply to all direct folding paths in the Turner energy model environment, indicating that the mixed MFE state is not always a required intermediate state in optimal paths. To verify this, a variant of the `Findpath` algorithm was tested against sample folding paths (as described in the dataset generation section) with nucleotide lengths ranging from 100 to 400. The results indicate that less than 1% of all folding paths per dataset fail to converge to optimal energy barriers.

A representative example of a path which fails to converge is shown in Figure 3.16, and its corresponding dot-bracket notation can be found in Appendix path 3. For this and other folding paths of shorter sequences, an optimal direct path algorithm based on **Dijkstra's algorithm** was used to confirm the existence of optimal paths $A \rightarrow$ MFE state $\rightarrow B$. If using the MFE state as a required checkpoint, there exists no concatenated path with the same optimal energy barrier as the direct path $A \rightarrow B$. Due to computational limitations, this verification could not be extended to longer folding paths in larger sequences.

Due to the intricacies of the RNA ensembles, multiple nearly isoenergetic structures can exist with similar low energies as the predicted MFE state. Even a slight model setting adjustment within the Turner energy model, such as increasing the temperature setting, can result in a completely different state. In some tested

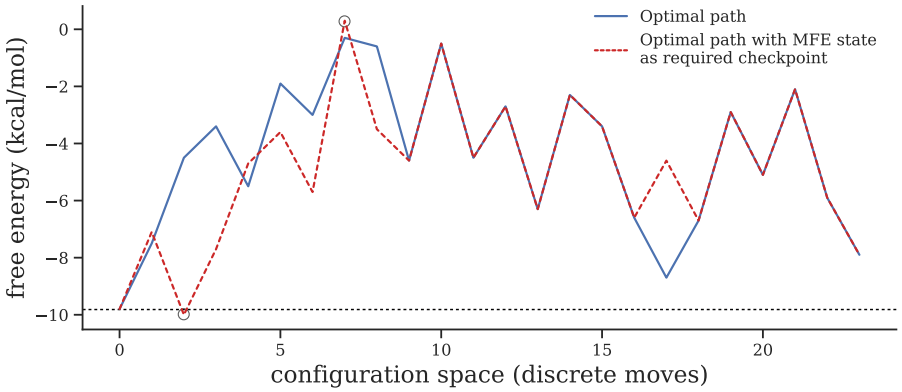


Figure 3.16: Sample path where the separation lemma does not hold for the Turner free energy model. The dashed line corresponds to the optimal path if using a mixed MFE state as a checkpoint (bottom marker). The optimal folding path (blue line) avoids the mixed MFE state as an intermediate and has a lower energy barrier (top marker). Corresponding paths see appendix path 3.

examples, minor model setting modifications led to a different MFE state, which then served as a valid checkpoint for the optimal path with the lowest energy barrier.

Occurrence of mixed MFE states

Within the sample dataset, a significant proportion of folding paths include mixed MFE states as intermediates, allowing for the computation of split paths (see percentages in Table 3.3). Longer paths tend to have a higher occurrence of mixed MFE states. It is important to note that these findings reflect the characteristics of the sample dataset, where A and B are local minimum structures in the energy landscape and often have similar energies. In completely different scenarios, such as when the initial structure has significantly higher energies than the final structure, the likelihood of encountering a mixed MFE state is inevitably lower.

100 *nt*	200 *nt*	300 *nt*	400 *nt*	500 *nt*	600 *nt*
51%	81%	88%	94%	98%	99%

Table 3.3: Occurrence of mixed MFE states within tested sample datasets, ranging from 100 to 600 *nt* sequence lengths, 1000 paths per length. In longer folding paths, the chance of a MFE state being not A or B is much higher.

Shift in Energy Barrier

The MFE state variant yields results in a similar manner to the divide-and-conquer scheme. If the MFE state is a required checkpoint for all resulting paths, the accuracy of the obtained results is good, but not perfect. In the tested sample datasets, a considerable proportion of input paths include mixed MFE states, leading to enhanced precision at lower search widths and improved runtimes. The average energy barrier shifts are comparable to those of the divide-and-conquer extension (see Table 3.4). Notably, for paths of any size, there will be always a certain percentage of paths with higher energy barriers than regular `Findpath` at typical low search widths, as seen in Figure 3.17. Convergence tests (see Figure 3.18) show that, on average, the MFE variant generates lower energy barriers across a wide range of search widths.

Dataset	100 *nt*	200 *nt*	300 *nt*	400 *nt*	600 *nt*
MFE extension	-0.02	-0.24	-0.53	-1.00	-2.18
DC extension	-0.01	-0.21	-0.52	-1.11	-2.65
DC+MFE extension	-0.03	-0.30	-0.71	-1.38	-2.91

Table 3.4: Average energy barrier shift in kcal/mol of extension variants vs. regular `Findpath`. Tested in sample datasets ranging from 100 to 600 *nt* sequence lengths (1000 paths per length) at search width $m^* = 2$.

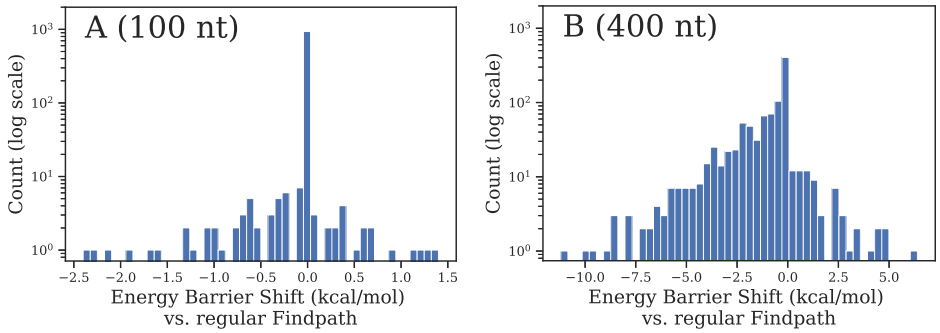


Figure 3.17: Sample folding paths with sequence lengths 100 (A) and 400 nt (B), 1000 paths per dataset, tested at $m^* = 2$. Shown are energy barrier shift histograms of the MFE state variant vs. regular Findpath. Average shifts see Table 3.4.

The utilization of this variant is recommended when using low search widths (e.g. $m^* = 2$) for folding paths of any length. Similar to the divide-and-conquer extension, it is important to consider that a small fraction of input paths may not converge to optimal energy barriers at high search widths.

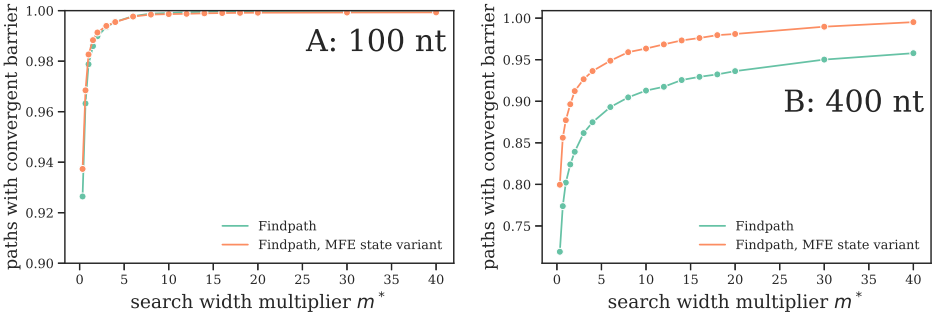


Figure 3.18: Convergence analysis of datasets with sequence lengths of 100 nt (A) and 400 nt (B), consisting of 1000 paths per dataset. The y-axis represents the percentage of paths within each dataset that achieve optimal energy barriers at various search widths (x-axis). The MFE state variant shows a nearly identical behaviour as the divide-and-conquer variant (Figure 3.12): if the extension is enabled, a higher percentage of optimal energy barriers is computed for any tested search width.

Runtimes

In terms of runtime, the calculation of the MFE state through constraint folding has negligible runtime impact on sequences of any length. The resulting runtimes, shown in Figure 3.19, are consistently lower when the extension is enabled. Further details on the performance of this variant, in combination with the divide-and-conquer extension, can be found in the concluding chapter.

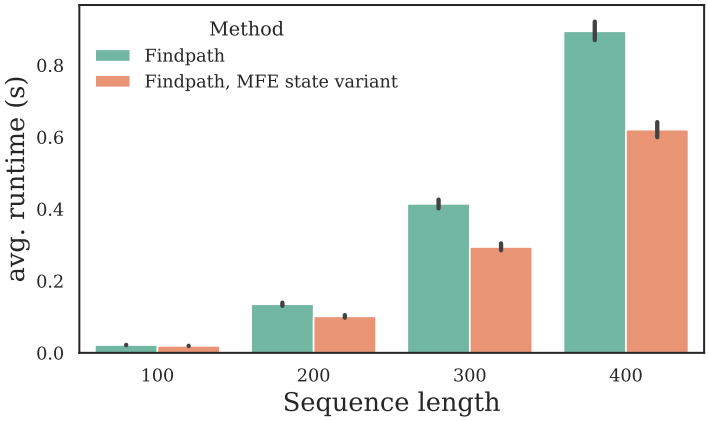


Figure 3.19: Runtime comparison with and without applied MFE state extension. Random folding paths of defined sequence lengths, with 1000 paths per length.

3.5 Findpath Extension: Move Restrictions

3.5.1 Theory

The original `Findpath` algorithm explores all possible permutations of elementary moves when the search width m approaches ∞. A modification to this fundamental model is to restrict the search to probable moves. This part of the thesis specifically investigates probable moves within helices, starting with their formal definition. Here, consecutive moves exhibit exploitable patterns that do not significantly compromise resulting folding paths and respective saddle energies.

The evaluation of possible moves per intermediate enables a modification to the algorithm, allowing for the restriction of the search to the most likely moves within helices. Consecutive moves within helices adhere to identifiable patterns (see Figure 3.20), although these patterns do not extend across different helices. This section sheds light on these patterns and explores how they can be utilized in the `Findpath` algorithm. Furthermore, these adaptations can be implemented alongside any RNA pathfinding heuristic.

For the following model, a **helix** is defined to only consist of base pairs which are either added or deleted during the folding path. Constant base pairs are ignored. A move (i, j) is associated to a helix if an adjacent move $(i+1, j-1)$ or $(i-1, j+1)$ exists.

3.5.2 Greedy and Adjacency restriction

Two distinct types of restrictions were assessed and examined individually, although they can also be used in conjunction.

Greedy Restriction

In the case of a **single helix**, consecutive moves for helix assemblies are typically based on greedy choices, meaning that the energetically lowest option is selected as the optimal next move, as seen in Figure 3.21. In the process of a typical helix assembly, the assumption is that only the initial base pair contributes to an increase in free energy, therefore defining the energy barrier. In contrast, the single helix disassembly has an optimal energy barrier which is rarely the result of greedy choices.

This single helix assembly model, which provided the basis for the later discussed greedy restriction, underwent extensive testing with a large dataset of random folding paths. The model demonstrated typical error rates below 0.1% in cases where the optimal path cannot be determined greedily. Noteworthy examples, like

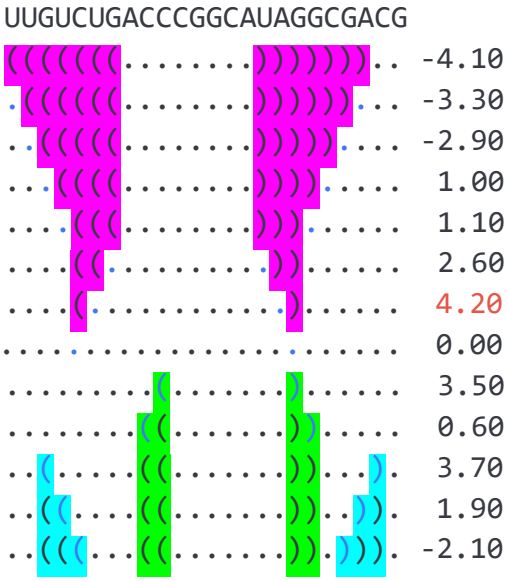
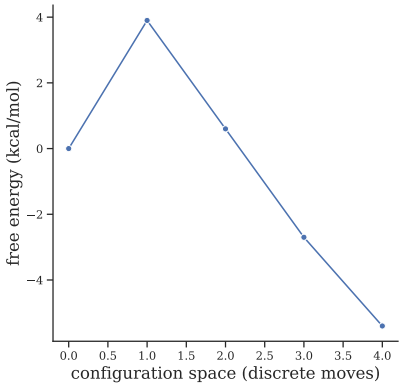


Figure 3.20: A sample folding path illustrating elementary moves organized into helices represented by different colours. Notably, patterns concerning the addition and removal of base pairs emerge within these helices. The adjacent assembly model stands out as the simple approach, characterized by the sequential addition or removal of base pairs exclusively at the end of existing helices (excluding the initial assembly move).

appendix path 4, often involved the presence of multiple G·U wobble base pairs within a helix. The energy model (Mathews et al., 2004) encountered challenges in determining energy contributions in such circumstances. The validity of the greedy assembly model is limited to a simplified energy model that does not include more complex tetraloop or dangling end exceptions. These results notably highlight the results obtained using the current ViennaRNA energy model, which may undergo changes and refinements in the future.

Due to the NP-complete nature of the folding path problem (Lyngsø and Pedersen, 2000), it is infeasible to compute optimal folding paths based on greedy choices for typical scenarios that involve the addition and deletion of multiple helices throughout the entire path. However, it is still possible to utilize the advantages of the greedy assembly model for single helices. The introduced **greedy restriction** assigns every available elementary move per intermediate into helices, and only the energetically best move per helix is considered in the **sorting step** of the `Findpath` algorithm. For instance, in Figure 3.20, the intermediate structure without base pairs would consider only one green and one light-blue assembly move in the sorting step, respectively. Ultimately, the goal of this restriction is thus to




```
Moves        Structures          Energy
             GGGGAAAACCCCUUUU    (kcal/mol)
-----------------------------------------
                 ................    0.00
[  3,  10 ]  ..(......)......    3.90
[  2,  11 ]  .((......)).....    0.60
[  1,  12 ]  (((......)))....   -2.70
[  4,   9 ]  ((((....))))....   -5.40
-----------------------------------------
S:   3.90 kcal/mol
```

Figure 3.21: Greedy helix assembly model. Within a simplified energy model, consecutive moves for an optimal path of a single helix as an isolated unit can be greedily chosen. However, when it comes to the dissociation moves from an existing helix, optimal paths may not be computed based on greedy choices.

act as a filter before selecting the m best candidates per iteration, with the hope of removing candidate structures which have no potential for optimal folding paths. It's important to note that this restriction does not apply to moves across helices, and is not valid for the removal of base pairs.

In terms of implementation, the restriction is unidirectional and ignores disassembly moves. `Findpath` always computes passes in both forward and backward directions. Consequently, even if a folding path includes a helix for which the optimal solution cannot be determined greedily, the computation of corresponding disassembly moves should ensure convergent saddle energies, regardless of whether the greedy restriction is applied. However, it is important to note that extremely rare cases involving problematic helices from both directions may still arise. Comprehensive results and performance metrics will be provided after discussing the next restriction.

Adjacency Restriction

Typically, the optimal permutation of helix assemblies reveals a sequential addition of base pairs adjacent to already existing pairs (path A in Figure 3.22). The in vivo occurrence of fragmented assemblies (path B), where the assembly appears to begin from two separate base pairs that are later merged, remains unknown.

Nevertheless, according to the Turner energy model from the current ViennaRNA release, fragmented assemblies for single helices are found to be required for optimal paths in approximately 0.2% of all tested cases. In a similar manner to the previously introduced greedy assembly model, the strictly adjacent assembly of helices is not able to reproduce all optimal folding paths since loop contributions

of the energy model are not always strictly positive. In contrast, the Turner energy model incorporates a complex set of additional rules. However, for most practical purposes, the strictly adjacent assembly model proves sufficient from the perspective of a heuristic pathfinding algorithm.

Thus, a `Findpath` adjacency restriction variant was tested which limits the dissociation of helices to only base pairs at the margin. This means that at most two elementary moves per helix of a given intermediate are considered in the sorting step of the algorithm. Since the greedy restriction already limits moves to one per helix, the adjacency restriction is not applied to helix assemblies and is therefore also unidirectional.

Figure 3.22: Sample helix assembly: Path A has moves which are strictly adjacent to previously existing base pairs, path B shows a fragmented assembly.

3.5.3 Results

Both tested restriction variants exhibit minor deviations in terms of folding path saddle energies, as seen in the convergence test in Figure 3.23. Specifically, at low search widths, the variant without restrictions performs slightly better in terms of saddle energies. Since both restrictions are only applied unidirectionally, it is expected that both variants will converge to the same result. For the 100 *nt* dataset (1000 sample paths per set), all variants converged to the same result at high search widths. However, for the 300 *nt* dataset, this could not be confirmed due to the computational cost at the required search widths. As mentioned earlier, folding paths should exist where problematic helices hinder convergence to the optimal result from both directions. However, despite an extensive testing dataset, such a sample folding path was not found.

Specifically, the adjacency restriction shows significant runtime improvements for comparably longer folding paths (Figure 3.24): Runtime improvements are caused by a decreased amount of structure energy evaluations, which have per structure a $O(1)$ runtime but still can't be disregarded. Results indicate that this restriction has no evident drawback.

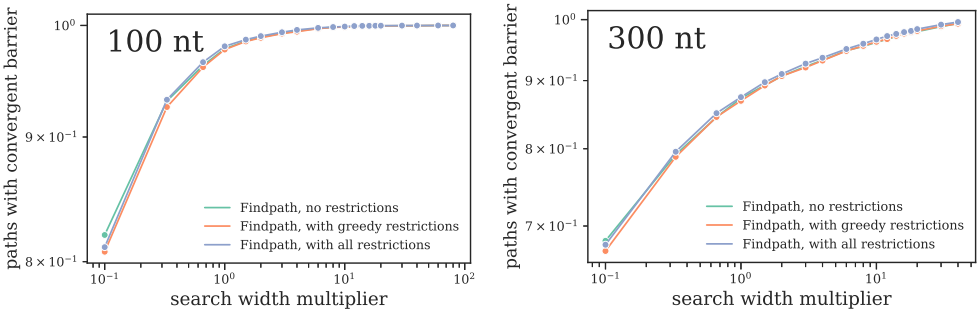


Figure 3.23: Convergence tests for different `Findpath` variants with and without restrictions, focusing on folding paths of sequence lengths 100 and 300 *nt* (1000 sample paths per dataset). The plot does not display results exclusively with the adjacency restriction due to visibility concerns, as this variant exhibits the same behaviour as the results when both restrictions are applied simultaneously. Across all tested variants, the differences in computed saddle energies are minimal and only noticeable in this logarithmic scale plot.

In comparison, the greedy restriction neither affects resulting saddle energies nor runtimes in a significant manner. To find the greedy choice per helix, it is still necessary to evaluate all potential elementary moves per intermediate. For the subsequent search step in the `Findpath` algorithm, results indicate that at typical search widths, no non-trivial choices are removed in actuality.

Implementation-wise, both restrictions have minimal additional computational overhead and can be computed in $O(1)$. The usage of the adjacency restriction is recommended for typical low search widths, while the effectiveness of the greedy restriction is debatable in contrast.

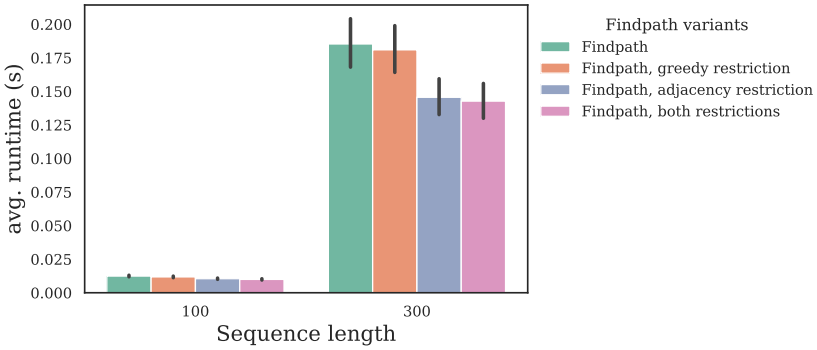


Figure 3.24: Measured runtimes for all restriction variants at sequence lengths 100 and 300 *nt* (1000 sample paths per dataset). Tested at $m^* = 2$, runtime differences of all variants are similar at higher search widths.

3.6 Findpath Optimizations

3.6.1 Sorting the best intermediates

One of the most time-consuming steps in the `Findpath` algorithm is the **sorting step** (algorithm 2.2): As mentioned previously, for each of the m retained paths per iteration, a maximum of n candidate paths are generated. These mn paths (upper bound) are all one elementary move closer to the final structure B, of which m are retained for the next iteration with the lowest saddle energies. This step also filters out duplicate terminal intermediate structures: only the best candidate path per terminal structure is retained.

In the realm of implementation, there are multiple ways to accomplish this task. The current ViennaRNA version (Lorenz et al., 2011) uses two consecutive sorting steps: The candidate paths are sorted by the terminal structure such that duplicate paths are removed (with worse saddle energies), all unique paths are sorted additionally by saddle energy and m best paths are retained.

To improve the runtime of the `Findpath` algorithm, a new sorting step was implemented that uses a **partial sorting algorithm**. These sort the first k elements in a defined order, while leaving all remaining elements in arbitrary order. In the context of the path sorting problem, k is unknown at first ($m \leq k \leq mn$). If k elements are sorted by saddle energy, this ensures that a fraction of paths within k corresponds to the best unique paths with unique terminal structures. Therefore, k has to be adjusted in multiple iterations such that m paths within k are unique, which ensures an identical result as the previous method.

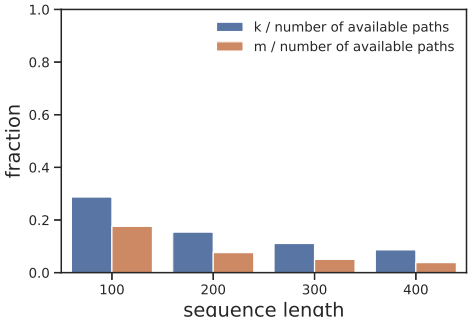


Figure 3.25: Fractions of k and m vs. the number of available paths at typical search widths $m^* = 2$.

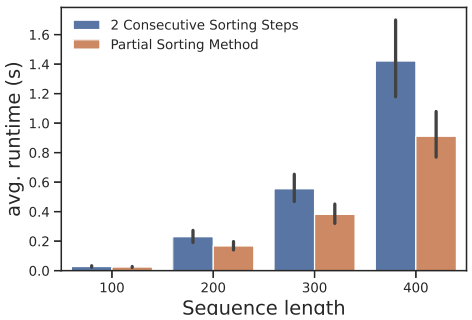


Figure 3.26: Runtime comparison of both sorting methods at fixed search width. one more line.

In Figure 3.25 (left) the relationship between k and m vs. the number of candidate paths available (bounded by mn) is shown. These tests were conducted using a

search width multiplier $m^* = 2$ on the sample dataset. In such scenarios where selected search widths are low, only a small fraction needs to be sorted. With longer sequences, the likelihood of encountering duplicate structures decreases proportionally, primarily because of the larger RNA landscape. Thus, there is a further reduction in the fractions of m and k. However, any possible advantage of the partial sorting variant is null and void if `Findpath` is used to exhaustively collect all path permutations.

Figure 3.26 shows runtimes of the implemented partial sorting variant in comparison to the old version. This approach has no perceived disadvantages and is faster in all tested scenarios, although runtime advantages are expectedly not as pronounced at higher search widths.

3.6.2 Identifying unique structures via hashing

As explained in the previous section, it is necessary to identify unique paths with unique terminal structures. Ideally, each structure should be represented by a unique small data type to minimize memory usage, i.e. 32-bit or 64-bit integers. Any structure representation which exceeds 64 bits (array-based representations) would result in an increased time complexity of the sorting step.

The most common strategy for this task is to utilize dot-bracket strings, which can be represented by a condensed variant with only 3 bits per sequence length to lower the memory footprint. In the context of RNA folding paths, there is a **binary move representation** available that has an even lower memory footprint. This representation is based on the list of used moves, as shown in Figure 3.27. A single processed move can be represented by a single bit, which means that a 64-bit data type can encode all possible structures from a path with lengths up to $n = 64$.

For the general case including longer sequences ($n > 64$), a practical approach is to compute a **hash value** per terminal structure. Since the computation of hash key values is generally faster for shorter inputs, the binary move representation is a good input for any hash function.

Two different hashing variants were tested subsequently. The first **slow hashing variant** utilizes the built-in hash function of the C++ standard library to compute unique 64-bit integers from the minimal representation. The input x of the hash function $h(x)$ is a string where up to 8 moves are stored per 8-bit character. Note that the C++ standard (ISO, 2012) does not define a specific implementation for the used function `std::hash`, but modern compilers ensure that the resulting hash values are computed quickly and are uniformly distributed among all possible hash values (2^{64} values for 64-bit data types).

```
Moves           Structures          move
                GGGGAAAACCCCUUUU    representation
-------------------------------------------------
                ((((....))))....    00000000
[  -4,   -9 ]   (((......)))....    00010000
[  -1,  -12 ]   .((......))....    10010000
[  -2,  -11 ]   ..(......).....    11010000
[  -3,  -10 ]   ..............      11110000
[   7,   14 ]   ......(......)..    11110010
[   6,   15 ]   .....((......)).    11110110
[   5,   16 ]   ....(((......)))    11111110
[   8,   13 ]   ....((((....))))    11111111
```

Figure 3.27: Minimal binary move representation of structures in a typical sample folding path. The binary move representation has a much lower memory footprint than the dot-bracket notation but is only valid in the context of folding paths if all potential moves are known beforehand.

The second **fast hashing variant** assembles a hash value from all previously used elementary moves. Each new move updates the hash value of a given structure based on the previous structure. All hash operations have to be permutation invariant such that all identical structures get identical hash values, i.e. if (i, j) and (k, l) correspond to elementary moves, the following must hold:

$$h((i, j), (k, l)) = h((k, l), (i, j))$$

First, hash values for all elementary moves (i, j) have to be computed: A straightforward approach here is to convert each move (i, j) into a string and use the inbuilt C++ hash function to compute $h(i, j)$. Permutation invariance for all subsequent moves can be achieved by either computing the sum of all moves, or better, with XOR operations:

$$h((i, j), (k, l)) = h(i, j) \oplus h(k, l)$$

This section will now examine the feasibility of both hashing variants. The subsequent section, focusing on the asymptotic runtime analysis, will also present an additional runtime analysis of both hashing variants. The **slow hashing variant**, where hash values are generated from strings, is guaranteed to have good statistical and avalanche properties. This means that two inputs with single-bit offset generate hash values where, on average, 50% of all bits are different. This raises the question of how well the second variant performs, specifically, whether there is a realistic chance of producing hash collisions.

The exact implementation of the **fast hashing variant** was inspired by the "frozenset" datatype in Python, which is an immutable set with a hash value from its components. According to the Python authors (Hettinger, 2014), XOR operations with specific additional bit-shift operations to compute $h(x)$ are optimal in such scenarios, according to empirical observations. In scenarios with typical low search widths (up to sequence lengths of 600 nt), it can be assumed that the number of structures compared per iteration is significantly below 10,000. Therefore, under the condition that the implemented hash function behaves as intended, potential hash collisions are extremely unlikely to occur in such scenarios. In simulations, no hash collisions could ever be reproduced, even if the number of compared structures is much larger. The implemented hash function behaves properly with no observed avalanche effect with uniform bit-distributions.

At longer sequence lengths, the fast hashing variant significantly enhances the overall `Findpath` runtime. The optimal runtime is achieved when using 32-bit hash values (as shown in Figure 3.28). However, it is recommended to use 64-bit hash values to minimize the likelihood of collisions. Due to computational constraints, it should be noted that this survey had limited scope as it was conducted only using low search widths.

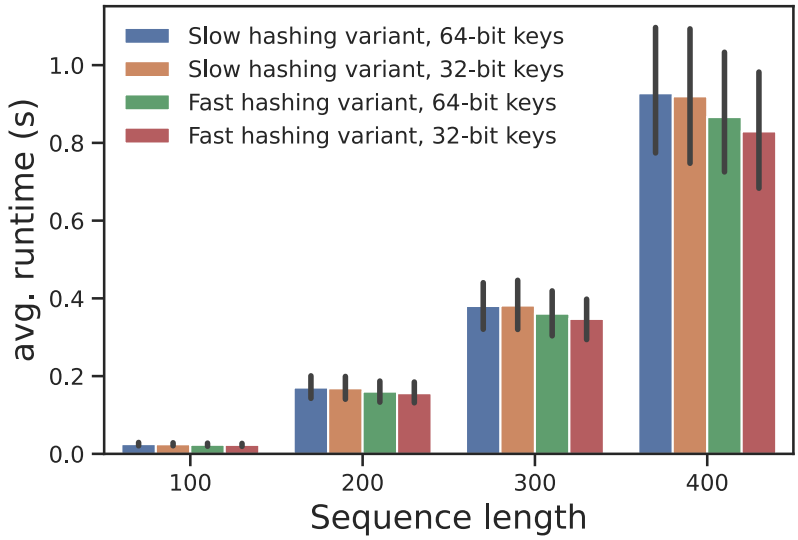


Figure 3.28: Runtime comparison of the `Findpath` algorithm with implemented slow and fast hashing function variants.

3.7 Findpath Computational Runtime Analysis

3.7.1 Current ViennaRNA Implementation

The runtime of the `Findpath` algorithm (see pseudocode in algorithm 2.2) depends on the path length n and the user-defined search width m, as explained previously. The first step to be discussed is the **candidate generation**, in which a maximum of mn candidate terminal structures (or paths) are generated in each of the n iterations. Following that, the **sorting step** will be explored, where the best candidates are selected.

The current implementation of the `Findpath` algorithm in the ViennaRNA package utilizes an ANSI-C implementation. The implementation utilizes pairing table arrays of size n to identify each candidate structure. In each of the n iterations, up to mn arrays are dynamically allocated using the *malloc* function. The asymptotic runtime cost of memory allocation is not clearly defined in the literature, as it depends on the environment of the program. However, in general, it can be assumed that the individual runtime costs are very low, and the size of the array does not impact the time taken by the memory allocator ($O(1)$). When considering all the allocations together, they are inefficient. In the new C++ implementation of `Findpath`, all the required memory is preallocated at the beginning, avoiding this problem altogether.

The computation of free energy and related tasks per candidate structure are efficiently performed in $O(1)$ in the current ViennaRNA (Lorenz et al., 2011) distribution.

The next subsequent step that also consumes a significant amount of time involves two consecutive sorting steps using *Quicksort*, which have an average runtime complexity of $O(mn \log(mn))$ per iteration. The remaining tasks of the algorithm in each iteration can be excluded from the runtime analysis.

In summary, the asymptotic runtime complexity of `Findpath` is

$$O(n(mn + mn \log(mn)))$$

which simplifies to:

$$O(mn^2 \log(mn))$$

3.7.2 Optimized Algorithm Implementation

In contrast to the ViennaRNA version, the new implementation in C++ introduces significant performance improvements in terms of partial sorting and structure hashing, as previously discussed. The analysis of the slow hashing variant's hashing implementation presents challenges in this runtime analysis. This variant utilizes the `std::hash` function provided by the C++ standard library to generate hash values from the minimal structure representation. However, the specific implementation of `std::hash` is not specified in the C++ standard and can vary depending on the compiler. In the worst-case scenario, these operations can potentially have a time complexity of $O(N)$, where N is the length of the input string. A slow hashing operation would result in an increase in the asymptotic runtime of the entire algorithm to $O(mn^3)$.

Specifically for the problem at hand, the minimal structure representation is typically only a few bytes in size, depending on the path length n. Considering that the hashing operation has never been observed to be a significant time-consuming step in the `Findpath` algorithm, even for longer sequences, it should be regarded as a $O(1)$ operation.

Similarly to the original implementation, the sorting step remains the most time-consuming component of the algorithm. The utilization of the partial sorting algorithm alters the runtime as discussed in the following.

In the general case of considering an array of size N with M elements that need to be correctly sorted, the chosen partial sorting algorithm begins by partitioning the array's best elements using `std::nthelement`, which has an average runtime complexity of $O(N)$ (according to the C++ standard ISO (2012)). Following this, the best M elements are sorted using `std::sort`, which has a runtime complexity of $O(M \log M)$, resulting in a combined runtime of $O(N + M \log M)$. It is worth noting that there are different approaches to this task. Another option is to use `std::partial_sort` ($O(N \log M)$), which is only faster when M is significantly smaller than N.

In context of the `Findpath` algorithm, a total of mn items are considered, but only k (where $k \sim m$) elements need to be correctly sorted. Considering n iterations, the overall runtime of the revised `Findpath` algorithm can be expressed as

$$O(n(mn + m \log m)) = O(mn^2 + mn \log m)$$

which simplifies to:

$$O(mn^2)$$

3.7.3 Experimental Runtime Evaluation

Several tests were conducted to compare the actual runtime performance with the theoretical algorithmic complexity. The theoretical analysis does not consider constant factors and makes no hardware-related assumptions. The experimental tests also have their limitations, which need to be acknowledged. The algorithm's behaviour can vary significantly based on the input size. Due to computational constraints, only paths with sequence lengths up to 600 nt were considered for the dataset. Tests were conducted at low search widths ($m^* = 2$). The results are depicted in Figure 3.23.

For this study, a new input dataset was generated. The dataset consisted of random folding paths for defined sequence lengths. In all previously used datasets, most structure pairs have base-pair distances distributed within a narrow range between $n = 50$ to $n = 100$. To avoid any bias caused by fitting and to include longer folding paths, the dataset was designed to have uniformly distributed distances ranging from $n = 40$ to $n = 260$ within the tested 600 nt dataset.

The regression analysis yielded a quadratic polynomial model with an R^2 value of 0.97 for all tested variants. Low p-values from both the quadratic and cubic fit models suggest that the quadratic model is sufficient, the additional complexity of the cubic model does not enhance the model's accuracy. These results emphasize the dominant influence of the quadratic terms in the runtime analysis for the original ViennaRNA implementation. The logarithmic component has a comparatively minor impact.

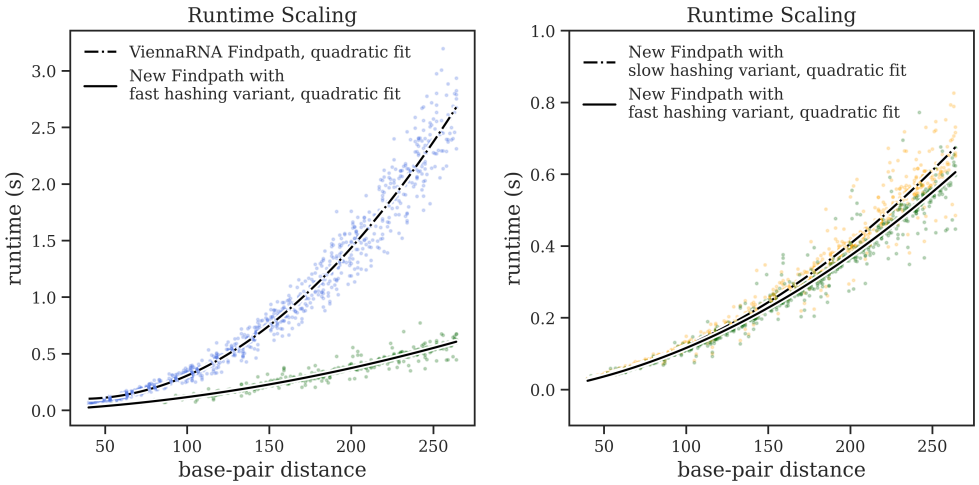


Figure 3.29: Comparative runtime scaling of various `Findpath` variants for folding paths of length 600 *nt* (based on a dataset with 1000 sample paths). The optimized implementation outperforms the original ViennaRNA implementation, resulting in significant speed improvements (left Figure). The fast hashing variant achieves notable gains for longer path lengths (right Figure, see also Figure 3.28). Quadratic regression models ($R^2 > 0.97$) accurately model all variants.

CHAPTER 4

Indirect Folding Path Heuristics

4.1 Background

As mentioned in section 2.4.3 (Algorithm Comparison and Outlook), developing **efficient** algorithms for indirect RNA folding paths remains an unsolved challenge. The indirect-path algorithms currently freely available, such as `RNAtabupath` (Dotu et al., 2010) and `RNAEApath` (Li and Zhang, 2012), are inefficient in terms of runtime and suffer from poor scaling. The resulting barrier energy estimates are often higher than direct-path estimates of `Findpath` (see Figure 2.9). It is worth noting that indirect path heuristics have received very little attention in research papers over the past decade.

Efficient heuristic pathfinding algorithms should be a valuable tool in RNA kinetics since direct paths are known to overestimate energy barriers. Estimating this relative energy shift is a challenging task since optimal indirect paths can only be computed for short sequences. `BARRIERS` (Flamm et al., 2001b) was used to compute a limited set of ground-truth paths for sequence lengths until 120 nt (100 paths per dataset). This workflow relies on `RNAsubopt` Wuchty et al. (1999), and therefore the energy band referenced by the MFE structure had to be limited to 10 $kcal/mol$ due to computational constraints.

If comparing ground-truth paths (`BARRIERS`) with direct paths (high search width `Findpath`) 59% of all tested paths at 70 nt and 73% at 120 nt have lower energy barriers (see the histogram in Figure 4.1 for the 120 nt dataset). The average energy deltas between both variants are 0.47 $kcal/mol$ (70 nt) and 0.58 $kcal/mol$ (120 nt), respectively. It is expected that this energy delta grows for longer sequences due to the increasing number of potential temporary base pairs.

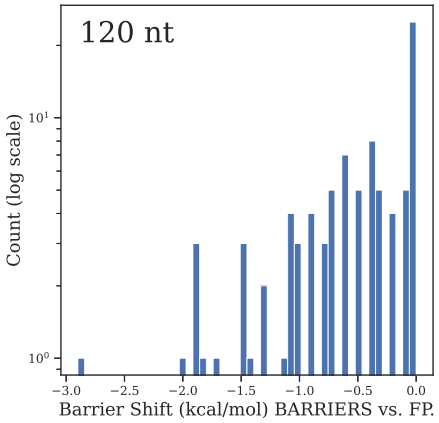


Figure 4.1: Energy barrier shift of optimal ground-truth paths (`BARRIERS`) compared with direct paths of `Findpath`. Limited 120 *nt* dataset with 100 paths.

4.2 Novel Approaches for Indirect Folding Path Heuristics

In addition to exploring enhancements for *direct* pathway algorithms, another objective of this thesis was to investigate potential approaches for adapting the `Findpath` heuristic for *indirect* RNA folding paths. The aim was to develop a workflow that generates indirect paths with minimal deviation from direct paths, where the path length n is only slightly longer than the base-pair distance $d(A, B)$. Paths with only slight detours can achieve significantly lower energy barriers while still maintaining reasonable computational requirements.

Among the discussed approaches of indirect path algorithms in section 2.4.3, the mesh-point approach was identified as the most promising method for achieving minimal detour paths. This approach was identified as particularly promising, as it allows for control over the deviation from the direct path by specifying a maximum detour distance, which will be introduced later. It involves selecting additional intermediate structures which lie outside the direct-path corridor of the energy landscape. One or more of these intermediates, referred to as M, then serve as checkpoints between the initial and final structures (A, B) of the folding path. It is worth noting that existing mesh-point algorithms for indirect RNA folding paths, as found in the literature, are either not freely available for download or require complex setup processes without optimized default settings.

The mesh-point approach has gained significance with the availability of efficient RNA landscape exploration algorithms, as the identification of potential intermediate structures is critical in computing detour paths $A \rightarrow M \rightarrow B$.

Notably, optimal indirect paths may contain **repeats**, meaning that the same base pairs may be added or removed multiple times at the same position. Although in theory, these paths can be established with multiple checkpoints ($A \rightarrow M_1 \rightarrow ... \rightarrow B$), these detour paths were not considered due to computational constraints.

4.3 Concatenated Detour Paths via Mesh-point Structures

Firstly, this section will discuss a novel workflow that utilizes `RNAxplorer` (Entzian et al., 2021) for the identification of mesh-points. The resulting indirect paths are obtained by concatenating two paths $A \rightarrow M$ and $M \rightarrow B$, similar to other mesh-point approaches documented in the literature. Subsequently, the following section will explore an alternative approach that directly computes indirect paths without path concatenation.

In the past few years, several new approaches for exploring the RNA landscape have been published. These heuristic algorithms aim to identify the key landmarks, or locally optimal structures, of the RNA energy landscape using sophisticated sampling methods. Recent algorithms in the field include `RNANR` (Michálik et al., 2017) and `RNAxplorer` (Entzian et al., 2021). For this work, `RNAxplorer` was selected as the exclusive method for acquiring mesh-point structures due to its fast implementation, as it has a polynomial runtime complexity in terms of the sequence length N and number of sampled structures.

Nevertheless, obtaining viable mesh-points for indirect paths is just one step of the developed workflow for indirect RNA folding paths. Here, a crucial aspect that greatly enhances the efficiency of the entire process is to determine the viability of intermediate candidate structures without the need for exhaustive computation of all paths $A \rightarrow M \rightarrow B$. The developed workflow encompasses the following steps:

(A) Generate mesh-point intermediate structures with `RNAxplorer`

(B) Filter structures to only allow slight deviations compared to direct paths

(C) Select a small number of structures based on different optimality criteria

(D) Compute and concatenate direct paths $A \rightarrow M$ and $M \rightarrow B$ with all remaining mesh-point candidates.

The initial step **A** in this baseline implementation utilizes an unmodified API of `RNAxplorer` to generate a minimum of 500 mesh-points. Within the `RNAxplorer`

framework, the exploration of the energy landscape can be fine-tuned using various adjustable parameters, which serve to encourage or penalize specific regions of the landscape. By setting A and B as the reference structures, the resulting mesh-point structures are local minima that ideally lie close to both input structures. Alternatively, it is feasible to adjust `RNAxplorer` such that the entire direct-path corridor serves as the reference. However, for this baseline implementation, which focuses on relatively short folding paths, the current approach is sufficient.

Given that resulting candidate structures may not always be in close proximity to the direct-path corridor or the provided reference structures, filtering step **B** restricts the detour distance in relation to the direct path. This distance, denoted as d_i, is calculated as $d_i = d(A, M) + d(M, B) - d(A, B)$. For the examined dataset consisting of input paths with sequences not exceeding 300 nucleotides, a maximum detour distance of $d_i = 10$ was selected as starting point. Initial steps **A** and **B** are repeated until at least 500 mesh-points are identified with the specified maximum detour distance.

Two different approaches were tested to reduce the number of viable mesh-points in step **C** without resorting to trial-and-error. In **Variant A**, structures with the lowest energy among the discovered structures are simply selected. In **Variant B**, structures are chosen based on the difference in free energy between identified structures M and the corresponding direct-path analogues M_d. The M_d structures are generated by removing base pairs that are not present in any possible intermediate of the direct path. The energy difference between M and M_d, as illustrated in an example in Figure 4.2, serves as a reliable indicator that M is a viable candidate for a mash-point structure.

```
          UGGCACCCCCGCGGAAAGUAGUCUCCUUGUACGCAUAUCGCUACAAAAUCGGGGAACUUC
A         ......(((((.(((........)))(((((.((.....)))))))...)))))......   -14.30 kcal/mol
M         ..((......)).......((((((((((.((.....)))))))....)))))..)).. -10.80 kcal/mol
M_d       ..........................((((((((((.((.....)))))))....)))))..)..   -9.50 kcal/mol
B         ..............((((..((((((((((.((.....)))))))....))))))))).   -10.80 kcal/mol
```

Figure 4.2: Finding potential mesh-points along the path $A \rightarrow M \rightarrow B$. Typical mesh-points M are close to the direct-path corridor ($d_i = 4$) and have comparably low free energies in relation to direct-path analogues (M vs. M_d). In this presented example, direct and indirect paths (via M) have barrier energies of 13.6 $kcal/mol$ and 12.5 $kcal/mol$, respectively.

Step **D** computes paths $A \rightarrow M$ and $M \rightarrow B$ of all candidate mesh-points with `Findpath`. A high search width ($m^* = 100$) is selected such that all computed direct paths are likely to be optimal paths. The final energy barrier is the result of the lowest energy concatenated path.

Results

Tests conducted on the dataset introduced earlier, which includes ground-truth paths (BARRIERS), indicate that the RNAxplorer-based mesh-point approach achieves energy barriers comparable to optimal paths for typical short sequences. When considering all 500 mesh-points without applying filtering step C, the workflow yields lower energy indirect paths in 90% of cases where the optimal path is also indirect, indicating that finding any detour path is not nearly as challenging compared to reconstructing the ground-truth path.

Estimated energy barriers cannot be as low as the ground-truth path since only one mesh-point is permitted along the path. Also, the detour-distance d_i affects the result. Notably, increasing d_i has a comparably negligible impact in terms of energy barriers. Mesh-point heuristics (variants A and B), which restrict the number of mesh-points to 20, indicate that such procedures have the potential to work well. These approaches have optimization potential, as it was observed that these 20 best structures are often very similar.

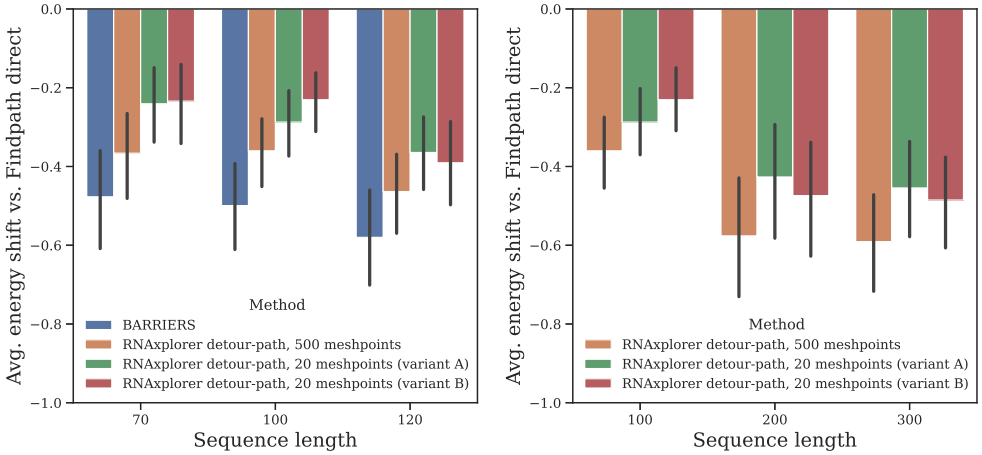


Figure 4.3: Average energy shift of indirect path variants vs. convergent direct paths. The Figure on the left displays datasets that include ground-truth paths (BARRIERS), whereas the Figure on the right demonstrates longer sample paths for which BARRIERS is not capable of generating such paths. Expectedly, all tested indirect-path variants show a negative energy shift compared to direct paths. RNAxplorer-based concatenated detour paths ($A \to M$ and $M \to B$) perform comparably well to optimal paths, depending on the number of examined mesh-point structures (Variant A and B). Limited datasets with 100 paths per length.

Nevertheless, the presented mesh-point approach consistently computes indirect paths with energy barriers that are, on average, 0.6 $kcal/mol$ lower than convergent `Findpath`-results in the datasets ranging from 120 to 300 nt, as depicted in Figure 4.3. These energy barriers are significantly lower compared to all other existing indirect-path heuristics. Notably, the energy difference between detour paths and direct paths appears to converge at approximately 0.6 $kcal/mol$. This indicates that detour paths, which contain a single mesh-point intermediate $A \rightarrow M \rightarrow B$, are imprecise estimations of optimal paths for longer sequences. However, since no ground-truth paths are available for such lengths, this remains an unconfirmed hypothesis.

4.4 Findpath with additional Indirect Moves

Rather than concatenating two paths $A \rightarrow M$ and $M \rightarrow B$, an alternative approach is to extract base pairs from mesh-point intermediates while ignoring the structure itself. The objective is to calculate an alternative path $A \rightarrow B$ using both direct and indirect elementary moves. This path may involve adding or removing additional helices and temporarily removing constant base pairs found in both A and B. Figure 4.2 provides an example of these additional indirect moves, which are highlighted in green within mesh-point M.

The identification and **extraction of indirect moves** improves the computational feasibility of the indirect path problem since there are orders of magnitude more potential mesh-point structures than indirect moves. For the discussed algorithms in this section all elementary moves may only occur once during the folding path, computed paths fall therefore into the category of indirect paths **without repeats**.

The aim was to assess various approaches for indirect heuristic algorithms, that rely on initial and final structures, A and B, as well as additional indirect elementary moves as input. Given that any such approach yields distinct optimal paths compared to ground-truth paths using `BARRIERS`, the development of a new method to generate optimal paths incorporating indirect moves was required.

Although the RNA pathfinding problem is a min-max optimization problem, it is possible to compute optimal solutions using **Dijkstra's algorithm** (Takizawa et al., 2020). Despite being an efficient algorithm in principle, with a best-case asymptotic runtime of $O(|E| + |V| \log |V|)$ (edges and nodes in a general graph), its complexity becomes exponential in the RNA pathfinding context since up to $n!$ nodes have to be considered even for direct path graphs. This limitation restricts its applicability, but it remains a valuable tool as it reliably computes exact solutions for short, indirect paths. Inspired by Takizawa et al. (2020), the newly developed Dijkstra's pathfinding algorithm generates optimal paths without repeats when

provided with a defined set of additional indirect moves. These additional moves are only included if a detour path results in a lower saddle energy path.

In similar vein, the `Findpath` algorithm was modified to allow for additional provided indirect moves. The BFS-heuristic treats direct and indirect moves equally, similar to the Dijkstra-implementation. However, unlike the direct-path `Findpath` variant, structural intermediates that are now compared in every iteration are not always equidistant to the final structure B, since e.g. indirect base pair additions increase the distance in relation to the initial structure A. This invalidates the algorithm's optimization efforts since duplicate intermediates may not be removed properly, as intermediates can appear multiple times during different iterations. Furthermore, this modification makes it difficult to verify the algorithm's correctness, and attempts to verify this behaviour resulted in comparatively worse performance metrics.

The developed indirect path workflow is similar to the previously discussed concatenation approach:

(A) Generate mesh-point intermediate structures with `RNAxplorer`

(B) Filter structures to only allow slight deviations compared to direct paths

(C) Select a small number of structures based on different optimality criteria

(D) Extract indirect elementary moves from all remaining structures

(E) Compute detour paths $A \rightarrow B$ with all supplied indirect moves using Dijkstra's pathfinding algorithm and using the indirect `Findpath`-variant

Theoretically, the indirect `Findpath` variant allows for quick computations of indirect paths. Unfortunately, even for short 100 nt paths, excessive search widths are required to generate paths with lower saddle energy than optimal direct paths, as seen in Figure 4.4. Even at $m^* = 100$, the heuristic algorithm fails to produce the same outcome as the Dijkstra implementation in a few instances (100 nt dataset). Nonetheless, the `Findpath` variant will eventually converge to the optimal Dijkstra result at even higher search widths for all tested cases. Due to these limitations, this `Findpath` variant should be considered as a proof-of-concept. A BFS-based algorithm is unlikely to be the best option for a detour pathfinding heuristic.

Notably, move-based detour paths tend to have lower saddle energies on average than concatenated mesh-point paths: `RNAxplorer`-sourced mesh-points often have the optimal set of indirect moves, but are often suboptimal intermediates. A concatenated path $A \rightarrow M \rightarrow B$ could be theoretically always computed as

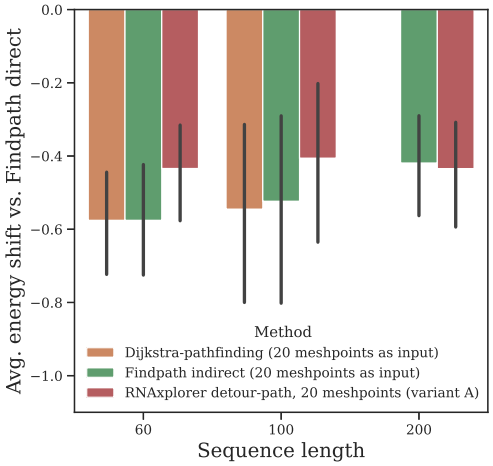


Figure 4.4: Average energy shift of indirect path variants vs. direct paths. 20 mesh-points per path (`RNAxplorer`-based approach) were used to extract all present indirect moves, which were subsequently used as input for the Dijkstra- and `Findpath`-based indirect pathfinding algorithms. The performance of the `Findpath` heuristic deteriorates rapidly for longer input paths, even at high search widths ($m^* = 100$). Limited datasets with 100 paths per sequence length.

path with indirect moves, as long as only one mesh-point is considered (repeats otherwise). The indirect `Findpath`-variant is not always able to compute known detour paths in longer paths (200 *nt* and above), indicating its limitation as heuristic in such scenarios.

In principle, it should be possible to develop an efficient heuristic algorithm that does not rely on breadth-first search (BFS) and takes into account both direct and indirect elementary moves, overcoming the inherent limitations of the presented indirect `Findpath` variant. This would enable significantly more efficient computations of indirect paths in comparison to the concatenation approach. However, such a novel algorithm would require extensive optimization efforts, which were beyond the scope of this thesis. It is important to note that even with a more effective heuristic, the inherent limitation of elementary moves with repeats, which are commonly observed in ground-truth paths, would remain.

CHAPTER 5

Concluding Remarks and Future Perspectives

The original `Findpath` heuristic is a well-documented breadth-first search algorithm that calculates direct folding paths between two RNA secondary structures with low energy barriers, close to optimal barriers. The algorithm is frequently referenced in other papers and is often the preferred heuristic algorithm to evaluate folding paths in the field of RNA bioinformatics. Several future perspectives for folding paths were evaluated, leading to the development of multiple enhancements and modifications of the original algorithm. These improvements aim to enhance `Findpath` in terms of computational runtimes and energy barrier accuracy.

The primary vision of this thesis was to increase the efficiency of folding path computations, particularly in scenarios where a significant number of energy barriers need to be calculated using limited computational resources and within reasonable runtimes. These objectives are relevant for applications such as the synthetic design of RNA switches or cotranscriptional folding heuristics, where folding path computations may be a small but time-consuming element of a bigger problem.

This work has proven that, even under the aspect that no significant progress in the field of *direct* path heuristics were published in the last two decades, `Findpath` and other related algorithms can be improved in a manifold of ways. Work focused on models which reduce the search space of move permutations, for example, the in-depth analyzed divide-and-conquer scheme. While the individual enhancements may not result in groundbreaking improvements in runtime or energy barriers, their cumulative effect enhances the overall algorithm and contributes to our understanding of the folding process.

Figure 5.1 displays a runtime comparison chart showcasing all fully developed enhancements to the `Findpath` algorithm, as extensively discussed in the extensions chapter. The use of a logarithmic scale enables comprehensive comparison across a wide range of input sequence lengths. Here, the move restriction (both adjacency and greedy restriction) proves to be the most effective enhancement in terms of speedup for short sequences up to a length of 200 *nt*. For longer sequences, the divide-and-conquer extension becomes the primary contributor to runtime improvements.

This comparison does not take into account the increased precision achieved by the MFE and divide-and-conquer extensions, which tend to compute lower energy barriers at the same search width. Additionally, the general optimization of `Findpath` significantly enhances runtimes even without activating the extensions. All extensions can be used in conjunction to enable synergistic effects for further runtime improvements.

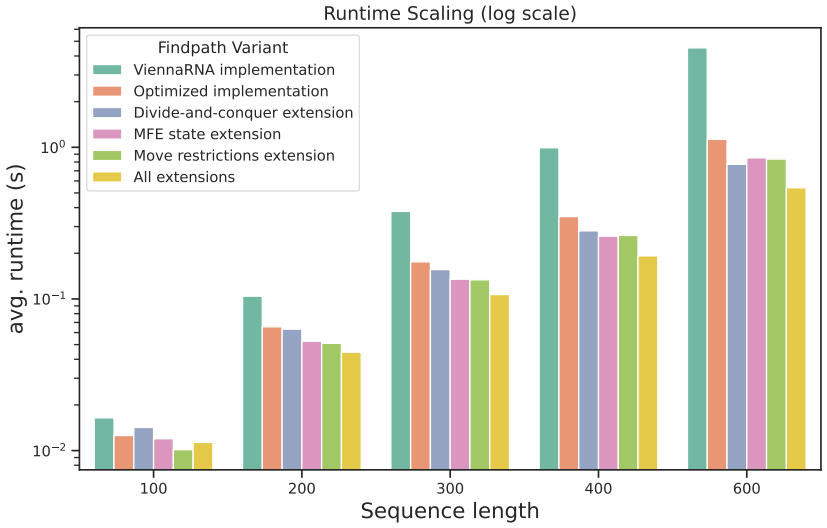


Figure 5.1: Runtime comparison chart showcasing the developed optimized `Findpath` implementation as well as all developed extensions for the algorithm, using random folding paths of varying sequence lengths (1000 paths per dataset). The logarithmic scale enables a clear comparison across the tested lengths. The new implementation, incorporating all extensions, has significant runtime improvements compared to the original ViennaRNA implementation, with up to 8x faster runtimes observed in the 600 *nt* dataset. The synergistic effects of all extensions together contribute to the performance improvements.

The newly developed folding path algorithm was specifically designed to serve as substitute for the original `Findpath` algorithm, complete with additional APIs. For users, it is important to take note of certain concerns regarding the implemented extensions, such as the potential computation of non-convergent optimal (direct) energy barriers at high search width settings.

In this concluding chapter, it is important to emphasize the utilization of the ViennaRNA framework in all the software developed within this thesis. The framework incorporates free energy calculations based on the Turner energy model for determining the energy of pseudoknot-free structures. In the context of RNA folding pathways, conformational changes are approximated using elementary base pair additions and deletions. It is worth noting that all currently published RNA folding path algorithms follow this framework, focusing on perfectly nested secondary structures. This simplification helps the folding path problem to be computationally manageable.

Nevertheless, it is important to acknowledge the limitations of the utilized framework. One such example is the omission of pseudoknots within the folding landscape, as discussed in Kucharík et al. (2015), which adds a significant additional level of complexity to folding pathways if they are considered. Depending on loop conformations and environmental conditions, pseudoknot-assisted pathways may have lower energy barriers but could also introduce additional kinetic traps that prevent efficient folding.

Once moving beyond the relatively narrow direct-path corridor in the energy landscape of a sequence to explore indirect folding paths, the developed mesh-point approach significantly improves upon existing indirect heuristics that have been published. While the tested mesh-point approach shows promise, it does have limitations when it comes to longer sequence lengths. This limitation is shared by all other existing heuristics. Nevertheless, this approach can be expanded in future revisions, such as considering multiple intermediates as pathway checkpoints. However, taking into account the computational cost, continued development of direct path algorithms should be also of value to the scientific community.

The future development of heuristic pathway algorithms relies on advancements within the energy model framework, and on finding the ideal balance between computational cost and energy barrier precision. The development of improved algorithms for both direct and indirect pathways represents a notable improvement compared to existing approaches. Fundamentally, these advancements serve as stepping stones, enabling further progress in the field of RNA folding pathway algorithms.

Appendix

This Appendix chapter provides dot-bracket notation representations of multiple folding paths, which serve as illustrations of the behaviour of the `Findpath` algorithm in the context of the tested extensions.

```
Outer Section GCUCCACUAGGGGUUACGAUGAGGGAGCCCUUGCUUCCCUCCUUUGGCCG
[   0,    0 ] (((((....)))))..(((.(...................)..))))....    -1.30
[ -19,  -44 ] (((((....)))))..((..(...................)...))....     1.10
[ -18,  -45 ] (((((....)))))..(...(...................)....)....     1.90
[ -17,  -46 ] (((((....)))))......(...................).........     0.00
S:   1.90 kcal/mol | B:   3.20 kcal/mol

Inner Section GCUCCACUAGGGGUUACGAUGAGGGAGCCCUUGCUUCCCUCCUUUGGCCG
[   0,    0 ] ....................(((((((....))).))))..........    -8.20
[ -26,  -35 ] ....................((((.((....))..))))..........    -5.20
[  26,   36 ] ....................(((((((....)).)))))..........    -8.20
[ -28,  -33 ] ....................((((((.....).))))).........    -5.30
[ -27,  -34 ] ....................(((((........)))))..........    -5.40
[  27,   35 ] ....................((((((.......))))))..........    -6.10
S:  -5.20 kcal/mol | B:   3.00 kcal/mol

Merged Path   GCUCCACUAGGGGUUACGAUGAGGGAGCCCUUGCUUCCCUCCUUUGGCCG
[   0,    0 ] (((((....)))))..(((.(((((((((....))).))))))..))).....   -17.40
[ -26,  -35 ] (((((....)))))..(((.(((((.((....)).))))))..))).....   -14.40
[  26,   36 ] (((((....)))))..(((.(((((((....)).)))))).))).....   -17.40
[ -19,  -44 ] (((((....)))))..((..(((((((....)).)))))..)).....   -15.00
[ -18,  -45 ] (((((....)))))..(...(((((((....)).)))))....).....   -14.20
[ -17,  -46 ] (((((....)))))......(((((((....)).)))))..........   -16.10
[ -28,  -33 ] (((((....)))))......((((((.....).))))).........   -13.20
[ -27,  -34 ] (((((....)))))......(((((........)))))..........   -13.30
[  27,   35 ] (((((....)))))......((((((.......))))))..........   -14.00
S: -13.20 kcal/mol | B:   4.20 kcal/mol
```

Figure 1: Sample Folding path for the divide-and-conquer extension with two sections with respective subpaths. In this typical example, the merged path is an optimal direct path.

```
A1              GUAAACGUUUUGGAGGAACUCGAGCUCUCCUAGACCUUUCGGAUAGUAGC
[   0,    0 ] ...........(.................).....................    4.40
[   7,   35 ] ......(....(.................)....).................    6.20
[   8,   34 ] ......((...(.................)...)).................    4.40
[   9,   33 ] ......(((..(.................)..))).................    2.20
[  10,   32 ] ......((((.(.................).)))).................    1.60
[   5,   37 ] ....(.((((.(.................).)))).).............    3.70
[   4,   38 ] ...((.((((.(.................).)))).)).............    2.80
[   3,   39 ] ..(((.((((.(.................).)))).))).............    2.10
[   1,   40 ] (.(((.((((.(.................).)))).))))...........    1.80
S:   6.20 kcal/mol | B:   1.80 kcal/mol

A2              GUAAACGUUUUGGAGGAACUCGAGCUCUCCUAGACCUUUCGGAUAGUAGC
[   0,    0 ] ...........(.................).....................    4.40
[   7,   35 ] ......(....(.................)....).................    6.20
[   8,   34 ] ......((...(.................)...)).................    4.40
[   9,   33 ] ......(((..(.................)..))).................    2.20
[   5,   37 ] ....(.(((..(.................)..))).).............    4.30
[   4,   38 ] ...((.(((..(.................)..))).)).............    3.40
[   3,   39 ] ..(((.(((..(.................)..))).))).............    2.70
[  10,   32 ] ..(((.((((.(.................).)))).))).............    2.10
[   1,   40 ] (.(((.((((.(.................).)))).))))...........    1.80
S:   6.20 kcal/mol | B:   1.80 kcal/mol | E[start]:  4.40 E[end]:  1.80

B               GUAAACGUUUUGGAGGAACUCGAGCUCUCCUAGACCUUUCGGAUAGUAGC
[   0,    0 ] ...........(((((.........)))))......................   -4.60
[ -16,  -26 ] ...........((((.........))))......................   -3.90
[ -15,  -27 ] ...........(((.............)))......................   -0.60
[  16,   27 ] ...........(((.(.........))))......................   -0.20
[  17,   26 ] ...........(((.((........)))))......................   -2.60
[ -14,  -28 ] ...........((..((........)).))......................    0.40
[  15,   28 ] ...........((.(((........)))))......................   -1.70
S:   0.40 kcal/mol | B:   5.00 kcal/mol

A1+B merged     GUAAACGUUUUGGAGGAACUCGAGCUCUCCUAGACCUUUCGGAUAGUAGC
[   0,    0 ] ...........(((((.........)))))......................   -4.60
[   7,   35 ] ......(....(((((.........)))))....).................   -2.80
[   8,   34 ] ......((...(((((.........)))))...)).................   -4.60
[ -16,  -26 ] ......((...((((.........))))...)).................   -3.90
[   9,   33 ] ......(((..((((...........))))..))).................   -6.10
[ -15,  -27 ] ......(((..(((.............)))..))).................   -2.80
[  10,   32 ] ......((((.(((.............))).)))).................   -3.40
[  16,   27 ] ......((((.(((.(.........)))).)))).................   -3.00
[  17,   26 ] ......((((.(((.((........)))).)))).................   -5.40
[ -14,  -28 ] ......((((.((..((........)).)).)))).................   -2.40
[  15,   28 ] ......((((.((.(((........))))).)))).................   -4.50
[   5,   37 ] ....(.((((.((.(((........))))).)))).).............   -2.40
[   4,   38 ] ...((.((((.((.(((........))))).)))).)).............   -3.30
[   3,   39 ] ..(((.((((.((.(((........))))).)))).))).............   -4.00
[   1,   40 ] (.(((.((((.((.(((........))))).)))).))))...........   -4.30
S:  -2.40 kcal/mol | B:   2.20 kcal/mol
```

Figure 2: Sample Folding path for the divide-and-conquer extension. In this rare example, two similar subpaths A1 and A2 (same section) are optimal with identical energy barriers, yet only A1 can be merged with subpath B into the optimal combined path. In contrast, merging subpath A2 with B yields an 0.2 kcal/mol higher energy barrier.

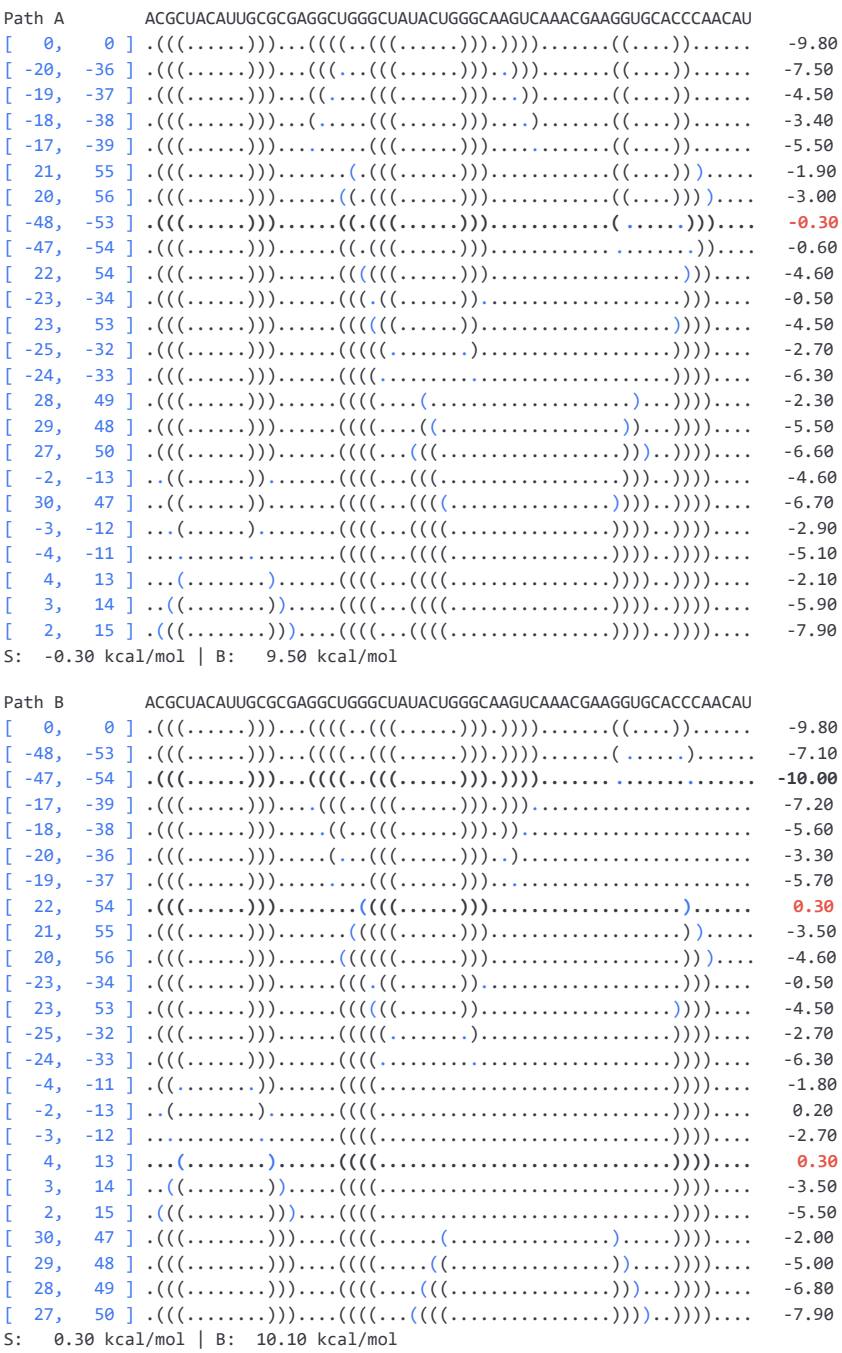


Figure 3: Sample path for the MFE state extension. In this rare example, the mixed MFE state (-10 kcal/mol) is not a required checkpoint of the optimal path (-0.3 kcal/mol saddle energy). Using the MFE state as checkpoint in path B yields a path with 0.6 kcal/mol higher energy barrier.

```
Greedy path   GCAAUGUAUGAAUUGUCACGAGGUAGCGUUAUGCCGGAUCUUACGGGUAAUUCACUUGUGUUC
[   0,   0  ] ..............................................................    0.00
[  17,   46 ] ................(............................)................    4.70
[  16,   47 ] ...............((............................))...............    4.60
[  15,   48 ] ..............(((............................)))..............    5.70
[  14,   49 ] .............((((............................))))............    4.50
[  13,   50 ] ............(((((............................)))))...........    4.00
[  12,   51 ] ...........((((((............................))))))..........    2.80
[  11,   52 ] ..........(((((((............................)))))))..........    1.90
[  10,   53 ] .........(((((((((............................)))))))))........   -1.80
[   9,   54 ] ........(((((((((............................)))))))))........   -2.50
S:   5.70 kcal/mol | B:   5.70 kcal/mol

Optimal path  GCAAUGUAUGAAUUGUCACGAGGUAGCGUUAUGCCGGAUCUUACGGGUAAUUCACUUGUGUUC
[   0,    0 ] ..............................................................    0.00
[  10,   53 ] .........(........................................).........    5.06
[  11,   52 ] .........((......................................)).........    3.61
[   9,   54 ] ........(((......................................))).........    2.91
[  12,   51 ] ........((((......................................)))).........    2.35
[  13,   50 ] ........(((((......................................)))))........    1.09
[  14,   49 ] ........((((((..............................))))))........    0.03
[  15,   48 ] ........(((((((..............................)))))))........   -0.84
[  16,   47 ] ........(((((((((............................)))))))))........    0.30
[  17,   46 ] ........(((((((((............................)))))))))........   -2.50
S:   5.06 kcal/mol | B:   5.06 kcal/mol
```

Figure 4: Sample Folding path showcasing the (mostly) greedy nature of helix assemblies. The energy barrier is obtained after a single elementary move, all subsequent moves lead to lower free energies. In rare examples, such as the presented one, helix assemblies have optimal paths which cannot be computed greedily. Note that the energy barrier of the optimal path is reached after a single move, but this optimal path cannot be computed greedily in rare circumstances.

List of Algorithms

List of Tables

List of Figures

Bibliography

Bian, Y., Zhang, J., Wang, J., Wang, J., and Wang, Y. (2015). Free energy landscape and multiple folding pathways of an h-type RNA pseudoknot. *PLOS ONE*, 10:e0129089.

Bogomolov, S., Raden, M., Voss, B., Podelski, A., and Backofen, R. (2010). Shape-based barrier estimation for RNAs. *German Conference on Bioinformatics*, pages 41–50.

Brion, P. and Westhof, E. (1997). Hierarchy and dynamics of RNA folding. *Annu Rev Biophys Biomol Struct*, 26:113–137.

Bulteau, L., Marchand, B., and Ponty, Y. (2021). A new parametrization for independent set reconfiguration and applications to RNA kinetics.

Darty, K., Denise, A., and Ponty, Y. (2009). VARNA: Interactive drawing and editing of the RNA secondary structure. *Bioinformatics*, 25(15):1974–1975.

Dotu, I., Lorenz, W. A., Van Hentenryck, P., and Clote, P. (2010). Computing folding pathways between RNA secondary structures. *Nucleic Acids Res*, 38(5):1711–1722.

Dykeman, E. (2015). An implementation of the gillespie algorithm for RNA kinetics with logarithmic time update. *Nucleic acids research*, 43.

Entzian, G., Hofacker, I. L., Ponty, Y., Lorenz, R., and Tanzer, A. (2021). RNAxplorer: harnessing the power of guiding potentials to sample RNA landscapes. *Bioinformatics*, 37(15):2126–2133.

Flamm, C., Fontana, W., Hofacker, I. L., and Schuster, P. (2000). RNA folding at elementary step resolution. *RNA*, 6(3):325–338.

Flamm, C. and Hofacker, I. (2008). Beyond energy minimization: Approaches to the kinetic folding of RNA. *Chem. Monthly*, 139:447–457.

Flamm, C., Hofacker, I., Maurer-Stroh, S., Stadler, P., and ZEHL, M. (2001a). Design of multi-stable RNA molecules. *RNA (New York, N.Y.)*, 7:254–65.

Flamm, C., Hofacker, I., Stadler, P., and Wolfinger, M. (2001b). Barrier trees of degenerate landscapes. *Zeitschrift für Physikalische Chemie*, 216.

Flamm, C., Wielach, J., Wolfinger, M. T., Badelt, S., Lorenz, R., and Hofacker, I. L. (2022). Caveats to deep learning approaches to RNA secondary structure prediction. *bioRxiv*.

Fu, L., Cao, Y., Wu, J., Peng, Q., Nie, Q., and Xie, X. (2021). UFold: fast and accurate RNA secondary structure prediction with deep learning. *Nucleic Acids Research*, 50(3):e14–e14.

Ge, H. and Qian, H. (2013). Chemical Master Equation. *Encyclopedia of Systems Biology*, pages 396–399.

Giegerich, R., Voss, B., and Rehmsmeier, M. (2004). Abstract shapes of RNA. *Nucleic acids research*, 32:4843–51.

Golden, M., Murrell, B., Martin, D., Pybus, O., and Hein, J. (2019). Evolutionary analyses of base-pairing interactions in DNA and RNA secondary structures. *Molecular biology and evolution*, 37.

Hettinger, R. (2014). Python frozenset hashing algorithm / implementation. Mathematics Stack Exchange. URL: https://stackoverflow.com/a/20931478 (version: 2023-03-01).

Hofacker, I. L. (2003). Vienna RNA secondary structure server. *Nucleic Acids Research*, 31(13):3429–3431.

ISO (2012). *ISO/IEC 14882:2011 Information technology — Programming languages — C++*. International Organization for Standardization, Geneva, Switzerland.

Kucharík, M., Hofacker, I. L., Stadler, P. F., and Qin, J. (2015). Pseudoknots in RNA folding landscapes. *Bioinformatics*, 32(2):187–194.

Li, Y. and Zhang, S. (2012). Predicting folding pathways between RNA conformational structures guided by RNA stacks. *BMC bioinformatics*, 13 Suppl 3:S5.

Lin, J.-C., Hyeon, C., and Thirumalai, D. (2012). RNA under tension: Folding landscapes, kinetic partitioning mechanism, and molecular tensegrity. *The journal of physical chemistry letters*, 3:3616–3625.

Lorenz, R., Bernhart, S., Höner zu Siederdissen, C., Tafer, H., Flamm, C., Stadler, P., and Hofacker, I. (2011). ViennaRNA Package 2.0. *Algorithms for molecular biology : AMB*, 6:26.

Lorenz, R., Flamm, C., and Hofacker, I. (2009). 2d projections of RNA folding landscapes. *German Conference on Bioinformatics*, pages 11–20.

Lorenz, R., Hofacker, I. L., and Stadler, P. F. (2016). RNA folding with hard and soft constraints. *Algorithms Mol Biol*, 11:8.

Lyngsø, R. and Pedersen, C. N. (2000). RNA pseudoknot prediction in energy-based models. *J Comput Biol*, 7(3-4):409–427.

Manuch, J., Thachuk, C., Stacho, L., and Condon, A. (2009). Np-completeness of the direct energy barrier problem without pseudoknots. 5877:106–115.

Mathews, D. H., Disney, M. D., Childs, J. L., Schroeder, S. J., Zuker, M., and Turner, D. H. (2004). Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proc Natl Acad Sci U S A*, 101(19):7287–7292.

Mathews, D. H., Sabina, J., Zuker, M., and Turner, D. H. (1999). Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *J. Mol. Biol.*, 288(5):911–940.

McCaskill, J. S. (1990). The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119.

Michálik, J., Touzet, H., and Ponty, Y. (2017). Efficient approximations of RNA kinetics landscape using non-redundant sampling. *Bioinformatics*, 33(14):i283–i292.

Mitchell, D. and Russell, R. (2014). Folding pathways of the tetrahymena ribozyme. *Journal of Molecular Biology*, 426(12):2300–2312.

Morgan, S. R. and Higgs, P. G. (1998). Barrier heights between ground states in a model of RNA secondary structure. *Journal of Physics A: Mathematical and General*, 31(14):3153.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453.

Nussinov, R. and Jacobson, A. B. (1980). Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proceedings of the National Academy of Sciences*, 77(11):6309–6313.

Ponty, Y. and Reinharz, V. (2022). RNA Folding. From text to graphs: Discrete structures and methods for Bioinformatics. *ISTE*, pages hal–03614168.

Sato, K., Akiyama, M., and Sakakibara, Y. (2021). RNA secondary structure prediction using deep learning with thermodynamic integrations. *Nature Communications*.

Seetin, M. G. and Mathews, D. H. (2012). RNA structure prediction: an overview of methods. *Methods Mol Biol*, 905:99–122.

Senter, E. and Clote, P. (2015). Fast, approximate kinetics of RNA folding. *Journal of Computational Biology*, 22.

Senter, E., Dotu, I., and Clote, P. (2014). RNA folding pathways and kinetics using 2d energy landscapes. *Journal of mathematical biology*, 70.

Takizawa, H., Iwakiri, J., Terai, G., and Asai, K. (2020). Finding the direct optimal RNA barrier energy and improving pathways with an arbitrary energy model. *Bioinformatics*, 36(Supplement_1):i227-i235.

Thachuk, C., Manuch, J., Rafiey, A., Mathieson, L.-A., Stacho, L., and Condon, A. (2010). An algorithm for the energy barrier problem without pseudoknots and temporary arcs. *Pac. Symp. Biocomput.*, pages 108–119.

Thirumalai, D. and Hyeon, C. (2005). RNA and protein folding: Common themes and variations. *Biochemistry*, 44(13):4957–4970. PMID: 15794634.

Tinoco, I. and Bustamante, C. (1999). How RNA folds. *J Mol Biol*, 293(2):271–281.

Treiber, D. K. and Williamson, J. R. (1999). Exposing the kinetic traps in RNA folding. *Current Opinion in Structural Biology*, 9(3):339–345.

Voss, B., Meyer, C., and Giegerich, R. (2004). Evaluating the predictability of conformational switching in RNA. *Bioinformatics*, 20(10):1573–1582.

Wiegreffe, D., Alexander, D., Stadler, P. F., and Zeckzer, D. (2018). RNApuzzler: efficient outerplanar drawing of RNA-secondary structures. *Bioinformatics*, 35(8):1342–1349.

Wolfinger, M. T., Svrcek-Seiler, W. A., Flamm, C., Hofacker, I. L., and Stadler, P. F. (2004). Efficient computation of RNA folding dynamics. *Journal of Physics A: Mathematical and General*, 37(17):4731.

Woodson, S. A. (2010). Compact intermediates in RNA folding. *Annual Review of Biophysics*, 39(1):61–77. PMID: 20192764.

Wu, M. J. (2018). Convolutional models of RNA energetics. *bioRxiv*.

Wu, M. T.-P. and D'Souza, V. (2020). Alternate RNA structures. *Cold Spring Harbor Perspectives in Biology*, 12:a032425.

Wuchty, S., Fontana, W., Hofacker, I. L., and Schuster, P. (1999). Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers*, 49(2):145–165.

Xayaphoummine, A., Bucher, T., and Isambert, H. (2005). Kinefold web server for RNA/DNA folding path and structure prediction including pseudoknots and knots. *Nucleic Acids Res*, 33(Web Server issue):W605–610.

Xayaphoummine, A., Viasnoff, V., Harlepp, S., and Isambert, H. (2006). Encoding folding paths of RNA switches. *Nucleic Acids Research*, 35(2):614–622.

Zhang, J., Fei, Y., Sun, L., and Zhang, Q. C. (2022). Advances and opportunities in RNA structure experimental determination and computational modeling. *Nature methods*, 19(10):1193—1207.

Zhao, Q., Zhao, Z., Fan, X., Yuan, Z., Mao, Q., and Yao, Y. (2021). Review of machine learning methods for RNA secondary structure prediction. *PLoS Comput Biol*, 17(8):e1009291.

Zuker, M. (2003). Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Research*, 31(13):3406–3415.