Linear Programming and Integer Linear Programming in Bioinformatics

Dissertation zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
vorgelegt von

Sandro Andreotti

am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

February 4, 2015

Datum der Disputation: ***15.12.2014***

Gutachter:
Prof. Dr. Knut Reinert, *Freie Universität Berlin, Deutschland*

Prof. Dr. Gunnar W. Klau, *CWI Amsterdam / Vrije Universiteit Amsterdam, Niederlande*

Da ist das Ding!

Oliver Kahn (15. Juni 1969), ehem. deutscher Fußballtorhüter.*

Abstract

A wide range of important problems related to bioinformatics and computational biology are optimization problems asking for a solution that minimizes or maximizes a certain objective function. Often, these problems are combinatorial optimization problems that can be formulated as integer linear programs. While for some of these problems polynomial time algorithms are known, for many other problems it is unlikely that such algorithms exist. However, much work has been dedicated to develop algorithms that are capable of solving many interesting integer linear programming problems on real live instances with acceptable memory and running time requirements. These algorithms are implemented in a variety of free or commercial solver software packages. In situations where the performance of general purpose solvers is insufficient, often problem specific integer linear programming techniques can be applied that take advantage of knowledge about the particular structure of the integer linear programming formulation to solve the problem in a much more time- or space-efficient way.

In this thesis we present our algorithmic approaches to three relevant bioinformatic problems, each involving certain linear programming and integer linear programming techniques.

The first problem is the *de novo* peptide sequencing problem, which consists in identifying a peptide's sequence solely from its tandem mass spectrum without any additional information stored in genome databases or protein databases. This problem can be formulated as a graph theoretical problem asking for the computation of a longest antisymmetric path in a directed acyclic graph. The particular structure of the associated integer linear programming formulation facilitates the application of a technique called Lagrangian relaxation, which yields an algorithm that outperforms state-of-the-art commercial integer linear programming solvers by orders of magnitude.

The second problem is the isoform inference and abundance estimation problem from RNA-Seq data. This problem consists in predicting a set of expressed RNA isoforms, i.e., full length RNA transcripts corresponding to alternative splice variants, together with an estimate of their individual expression levels. We apply a linear programming technique called delayed column generation, which allows us to increase the search space without explicitly enumerating the potentially huge set of candidate isoforms. As a consequence, our approach allows for the identification of isoforms that otherwise could not be recovered due to incomplete read coverage. A central component of our delayed column generation algorithm is an integer linear programming formulation.

The third problem is the duplication-loss alignment problem, which asks for a labeled alignment of two genome sequences that implies the minimal number of loss and duplication events in the evolutionary history from an unknown nearest common ancestor. In a labeled alignment, every unaligned gene must be labeled either as a loss or as the product of a duplication event. Once an optimal labeled alignment has been computed, a common ancestor genome with minimal implied evolutionary operations can be derived in a straight forward way. In our approach we identified problem specific cutting planes and developed efficient separation algorithms to obtain a branch and cut algorithm that is several orders of magnitude faster than existing approaches based on integer linear programming.

Zusammenfassung

Viele bedeutende Probleme im Forschungsbereich der Bioinformatik sind Optimierungsprobleme, bei denen eine Lösung errechnet werden muss, welche eine gegebene Zielfunktion minimiert oder maximiert. Bei einigen dieser Probleme handelt es sich um kombinatorische Optimierungsprobleme, welche als ganzzahlige lineare Optimierungsprobleme formuliert werden können. Während für eine Teilmenge dieser Probleme effiziente Polynomialzeit-Algorithmen bekannt sind, ist es für viele Probleme unwahrscheinlich, dass solche Algorithmen überhaupt existieren. Infolge kontinuierlicher Forschung wurden jedoch Algorithmen und Methoden entwickelt, die es erlauben viele ganzzahlige lineare Optimierungsprobleme für reale Instanzen mit vertretbarem Rechenaufwand zu lösen. Diese Algorithmen sind in einer Vielzahl von freien oder kommerziellen Löser Software Paketen implementiert. In Situationen, in denen die Effizienz solcher Löser unzureichend ist, kann die Anwendung problemspezifischer Methoden der ganzzahligen linearen Optimierung, welche die spezielle Struktur der Formulierung berücksichtigen, den Zeit- oder Speicherbedarf signifikant reduzieren.

In dieser Arbeit werden algorithmische Ansätze für drei relevante Probleme der Bioinformatik vorgestellt, die jeweils bestimmte Methoden der linearen Optimierung und ganzzahligen linearen Optimierung verwenden.

Als erstes wird das Problem der *de novo* Peptidsequenzierung betrachtet, welches darin besteht, die Sequenz eines unbekannten Peptids aus seinem Tandem Massenspektrum zu rekonstruieren, ohne zusätzliche Informationen aus Genom- oder Proteindatenbanken zu verwenden. Dieses Problem lässt sich als das graphentheoretische Problem des längsten antisymmetrischen Pfads in einem gerichteten azyklischen Graphen formulieren. Die besondere Struktur der Formulierung als ganzzahliges lineares Optimierungsproblem ermöglicht eine effektive Anwendung von Lagrange Relaxierung. Der resultierende Algorithmus ist um mehrere Größenordnungen schneller als leistungsfähige kommerzielle Löser Software.

Das zweite Problem behandelt die Vorhersage exprimierter RNA-Isoformen, welche alternativen Spleißvarianten desselben Gens entsprechen, basierend auf RNA-Seq Daten. Gleichzeitig sollen die individuellen Expressionsniveaus der einzelnen Isoformen bestimmt werden. Der vorgestellte Ansatz verwendet die Methode der verzögerten Spaltengenerierung, die es ermöglicht den Suchraum der Isoform Kandidaten zu erweitern, ohne eine vollständige Aufzählung dieser Kandidaten erforderlich zu machen. Durch die gezielte Erweiterung des Suchraums können auch Isoformen korrekt vorhergesagt werden die zuvor aufgrund unvollständiger Abdeckung durch sequenzierte reads nicht betrachtet werden konnten. Ein zentraler Bestandteil des vorgestellten Algorithmus zur Generierung neuer Spalten ist ein ganzzahliges lineares Optimierungsproblem.

Als drittes Problem wird das Duplikation und Verlust Alignment Problem betrachtet, bei dem für zwei Gensequenzen ein gelabeltes Alignment gesucht wird. In einem gelabelten Alignment ist jedes ungepaarte Gen entweder als Produkt einer Duplikation oder als Genverlust gelabelt. Aus einem berechneten optimalen gelabelten Alignment kann anschließend die Gensequenz eines gemeinsamen Vorfahren konstruiert werden, welches die Anzahl an erforderlichen Verlust- und Duplikationsereignissen in der Evolutionsgeschichte minimiert. Der vorgestellte branch and cut Algorithmus basiert auf neuen, problemspezifischen Schnittebenen und effizienten Separierungsalgorithmen und ist um mehrere Größenordnungen schneller bei der Berechnung eines optimalen gelabelten Alignments als existierende Ansätze.

Acknowledgements

I want to express my gratitude to everyone who supported me throughout the long and exhausting course of this thesis. I thank my supervisor Prof. Knut Reinert for giving me the opportunity to work on several interesting problems, for his continuous support and advices. My thanks also go to Prof. Gunnar W. Klau for his constant support, his valuable scientific input and motivation over the past years and for offering me scientific research visits at the CWI Amsterdam. I also want to thank the IMPRS-CBSC for their financial support of my scientific research visits, for the enjoyable retreats and social events. Special thanks go to my colleagues, first of all to Dr. Stefan Canzar for close and successful collaborations on two interesting research projects and also many interesting discussions about current issues related to FCB and BVB. Further, I want to thank the whole AG-ABI group, especially Stephan, David and Chris for their mental support and several inspiring burger and schnitzel eating orgies. Also special thanks go to Kathrin for many helpful comments on this thesis. Finally, I want to thank my friends, family and especially Michèle for their selfless and unconditional support and their patience during the last months.

CONTENTS

Part I

INTRODUCTION

Chapter 1 Introduction

1.1 About bioinformatics

Bioinformatics and computational biology

Bioinformatics and *computational biology* are two relatively young but rapidly emerging interdisciplinary research fields, studying the solution of computational problems arising from theoretical models of biological processes and systems. While bioinformatics is rather focused on the development of software tools and algorithms to efficiently process large amounts of biological data or complex biological data, computational biology is focused on the development of large scale data-analytical and theoretical models and computational simulation techniques (see exact definitions in [Huerta et al., 2000]). However, as it is hardly possible to find a clear separation between these two research fields, we put no emphasis on a discrimination between them throughout this thesis and use "bioinformatics" as an umbrella term for both.

Research areas

One of the most intensively studied areas in bioinformatics is *sequence analysis*, which addresses problems like the comparison of genomic sequences, identification of genes, efficient search in large sequence databases, or identification of sequence motifs in a collection of functionally related sequences.

At a higher level, *computational evolutionary biology* studies the evolutionary mechanisms in the speciation process, based on available genomic sequence information and large scale genome comparison.

Another research area is focused on the *quantitative analysis* of gene expression or protein expression, which involves algorithms and statistical methods to interpret the experimental data obtained by different experimental workflows. This analysis demands robust methods to account for noise or other artifacts often present in experimental data, to avoid misleading interpretations.

Structural bioinformatics aims at predicting the three-dimensional structure of biomolecules, especially proteins and RNA, which is of particular interest, since the three-dimensional structure determines their function and also the interactions between different molecules. Hence, the structural information can give insights into more complex biological systems. The analysis and simulation of such systems like networks of metabolites, protein-protein interaction networks, gene-regulatory networks, and signal transduction pathways is addressed by the research area of *network and systems biology*.

1.2 About this thesis

In this thesis, we present algorithmic approaches to three relevant problems related to bioinformatics. All three approaches are based on a linear programming (LP) or integer linear programming (ILP) formulation of the problem. This means, we optimize a linear objective function over some set of variables, subject to a set of linear constraints that must be fulfilled by the solution. In case of integer linear programming, all variables are restricted to take integer values, which renders the problem computationally hard.

While there exist general purpose solver software solutions that can be applied to general LP or ILP formulations, the formulations considered in this thesis demand advanced techniques that exploit their specific properties. For each of the three approaches, we apply a different technique to deal with the specific properties of the associated formulation and to obtain algorithms with good practical performance.

After a brief introduction to basic concepts in linear programming and integer linear programming in Chapter 2, we proceed with a comprehensive discussion of the three algorithmic approaches in the following chapters.

In Chapter 3, we present our algorithmic approach to the *de novo* peptide sequencing problem for tandem mass spectrometry data, which aims at the inference of the amino acid sequence of an unknown peptide solely from its tandem mass spectrum. As sketched in Figure 1.1, this important problem in the area of computational proteomics can be formulated as a graph-theoretical problem, asking for a so-called *longest antisymmetric path*. This problem corresponds to the longest path problem in an directed acyclic graph, complicated by a set of additional constraints to prevent the simultaneous selection of certain vertex pairs. As this problem is known to be NP-hard in the general case, without restrictions on the structure of forbidden vertex pairs, most existing approaches consider only restricted models that allow to solve the problem in polynomial time. However, the structure of the integer linear programming formulation for the longest antisymmetric path problem is particularly well suited for application of the *Lagrangian relaxation* technique, which is based on the idea of relaxing complicating constraints and adding their violation as a penalty to the objective

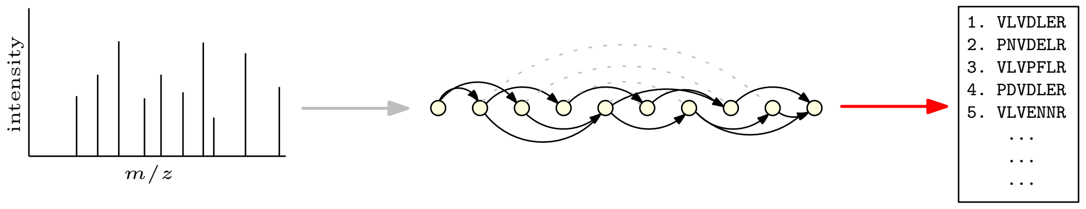


Figure 1.1: *Schematic workflow of de novo peptide sequencing. Given a tandem mass spectrum of the unknown peptide, the first step is the construction of the associated spectrum graph. Every path in the spectrum graph corresponds to a unique peptide sequence. However, a simple path search is not sufficient to solve the problem, as a feasible solution must not contain particular pairs of vertices that imply ambiguous interpretations of mass peaks. The result is a list of candidate peptide sequences ranked by some score to estimate the likelihood of each candidate to generate the given tandem mass spectrum.*

function. We embed our Lagrangian relaxation formulation into a branch and bound algorithm and a graph algorithm for efficient enumeration of suboptimal solutions. The resulting algorithm outperforms a straightforward ILP formulation solved by one of the worlds leading commercial solvers by several orders of magnitude, while, at the same time, it does not depend on a certain structure of forbidden vertex pairs like existing polynomial time dynamic programming algorithms.

In Chapter 4, we discuss our approach to the problem of isoform inference and abundance estimation from whole transcriptome shotgun sequencing (RNA-Seq) data, as outlined in Figure 1.2. The general idea is to use read mapping coverage on exons and splice junctions to identify the set of expressed isoforms, e.g., alternative splice variants, and their associated abundance. Our approach to select a subset of expressed isoforms from a possibly very large set of candidate isoforms and to estimate their abundance is based on regularized linear least squares problem. While in general a regularized least squares problem can be solved efficiently using quadratic programming solvers or other special purpose algorithms, the potentially large number of variables, i.e., candidate isoforms, renders these standard approaches impractical for complex genes. We tackle this problem by means of a linear programming technique, called *delayed column generation*, to avoid an exhaustive enumeration of all variables. As such, we begin with an initial solution computed for a subset of candidate isoforms and successively modify the model by adding new variables that lead to an improved solution. The central prob-

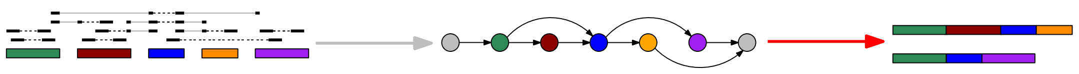


Figure 1.2: *Schematic workflow of isoform inference from RNA-Seq data. From a set of paired-end reads mapped onto exons and splice junctions (i.e., exon-exon junctions), we first construct the splicing graph, which encodes a set of candidate isoforms. The number and location of mapped reads is then used to infer a (preferably small) set of expressed isoforms together with an estimate of their respective abundance that yields an good explanation (prediction) of observed read mappings.*

lem in delayed column generation is the so-called *pricing problem*, which consists in the identification of such a variable or a proof that no such variable exists. We formulate the pricing problem as a graph problem based on a hypergraph, which we solve by means of an ILP formulation. Our algorithm is therefore capable of considering a large space of candidate isoforms while, at the same time, incorporating paired-end read information into the inference process.

In Chapter 5, we describe a *branch and cut* algorithm for the *pairwise duplication-loss alignment problem*, a recently defined problem in the research area of computational evolutionary biology. In the duplication-loss model of evolution, genomes are supposed to evolve from the evolutionary ancestor only by the two content modifying evolutionary operations: loss and duplication. The *two species small phylogeny problem* depicted in Figure 1.3 asks for a common ancestor genome of two given genome sequences with minimal total cost of implied evolutionary operations. As the two considered operations preserve gene order, this problem can be stated as an alignment problem. Given two gene sequences as input, the duplication-loss alignment problem asks for a *labeled alignment* of the two sequences where every unaligned gene is labeled as the product of a duplication event or as a loss in the other genome such that the summed cost of the implied evolutionary operations is minimized. Our approach builds upon an existing ILP formulation, for which we present several classes of constraints that yield a stronger LP-relaxation. Since the number of constraints is too large for an exhaustive enumeration and integration into the ILP model, we use the cutting plane technique, which consists of iteratively solving the LP-relaxation and adding new valid inequalities that are violated by the current optimal solution. Further, in a proof of concept study, we extend the model to three species to solve a restricted version of the *median-of-three* problem, which is a central component

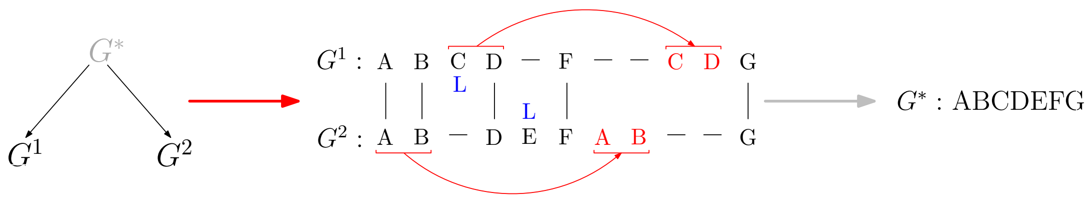


Figure 1.3: *Schematic description of the duplication-loss alignment problem. Given two gene sequences G^1 and G^2, we infer a common ancestor G^* that implies the least number of duplication events and loss events by computing an optimal labeled alignment of G^1 and G^2, where every unaligned gene is labeled either as a loss (denoted by blue "L") or as the product of a duplication event (denoted by red arcs).*

of the Steinerization heuristic for the *small phylogeny problem* on phylogenetic trees with more than two extant species.

1.3 Mathematical optimization in bioinformatics

With our application of linear programming and integer linear programming to problems related to bioinformatics, we extend a considerable library of successful approaches that have been published within the last 20 years. A comprehensive review summarizing successful applications published before 2006 is given by Lancia [2008]. We will now briefly present a selection of these approaches, extended by some more recent publications, without any claim of completeness.

Mathematical optimization approaches have been proposed for a variety of problems in different areas of bioinformatics. Some applications in sequence analysis include a branch and cut approach to the *multiple sequence alignment problem* by Kececioglu et al. [2000] and Althaus et al. [2005], an ILP formulation for the *inverse alignment problem*, which aims at determining a cost function for a given, biologically validated multiple sequence alignment, proposed by Kececioglu and Kim [2006], and an ILP formulation to determine the consensus of a given set of sequences by Meneses et al. [2004]. A more recent application proposed by Ritz et al. [2010] uses an ILP formulation to identify *structural variations* in genomic DNA from strobe sequencing data.

In structural bioinformatics, the *protein threading problem* gave rise to several publications describing approaches to this problem by means of an ILP formulation by Xu et al. [2003], a branch and cut formulation by Xu et al. [2004] and a Lagrangian relaxation approach by Balev [2004]. Another, more recent approach by Collet et al. [2011] studies a local variant of the protein threading problem. Some approaches to the *protein side-chain positioning problem* comprise ILP formulations by Eriksson et al. [2001] and Kingsford et al. [2005] and a Lagrangian relaxation algorithm proposed by Canzar et al. [2011]. Althaus et al. [2002], solve a similar problem in their approach to the *protein docking problem*, using a branch and cut approach. For the problem of *protein structure alignment* based on the *contact map overlap* measure, Lancia et al. proposed a branch and cut algorithm introduced in [Lancia et al., 2001] and a Lagrangian relaxation algorithm presented in [Caprara and Lancia, 2002], which has later been extended by Wohlers et al. [2010]. An alternative ILP model and Lagrangian relaxation algorithm have been proposed by Andonov et al. [2008; 2011]. In their approach to the related problem of *structural RNA alignment*, Bauer et al. [2005; 2007] proposed a Lagrangian relaxation algorithm similar to the one by Caprara and Lancia [2002]. Motivated by this work, Kato et al. [2010] developed an ILP formulation for accurate RNA-RNA interaction prediction.

In *computational proteomics*, two examples for approaches based on ILP formulations are given by Zerck et al. [2013], who propose a model for optimal *precursor ion selection* in Liquid chromatography tandem mass spectrometry and by Bertsch et al. [2010], proposing a model for optimal *de novo* design of *multiple reaction monitoring* (MRM) experiments.

Examples for applications to systems biology are given by the work of Backes et al. [2012], presenting an ILP formulation and a branch and cut algorithm to identify deregulated subgraphs in gene regulatory networks, Dittrich et al. [2008] proposing a branch and cut algorithm to identify functional modules in protein-protein interaction networks, and a very recent publication by Lu et al. [2014], who propose an ILP formulation for the design of synthetic metabolic networks by *minimum reaction insertion*.

This small selection from a plethora of publications describing integer linear programming approaches to bioinformatics reveals the importance and the potential of these methods to enhance our understanding of biological processes.

Chapter 2 Mathematical Preliminaries

This chapter serves as a short introduction to linear and integer linear programming. It is based on the textbooks by Bertsimas and Tsitsiklis [1997] and Wolsey [1998], but covers only the most important aspects required to follow the approaches presented throughout this thesis. For a deeper study of this topic, the interested reader should refer to one of many available textbooks on linear and combinatorial optimization, e.g., reference material [Bertsimas and Tsitsiklis, 1997; Wolsey, 1998; Nemhauser and Wolsey, 1988; Chvátal, 1983].

2.1 Linear programming

The concepts, definitions, and notation we present in this section are based on the textbook [Bertsimas and Tsitsiklis, 1997, chap. 1-3]. Note that some definitions may differ slightly between textbooks.

Linear programming defines the problem of minimizing or maximizing a linear objective function, subject to a set of linear inequalities and equalities. Since many important problems in production planning, transporting, scheduling and other economic fields can be stated as a *linear program*[1] (LP), this mathematical discipline developed rapidly since 1947 when G. B. Dantzig proposed an algorithm to solve linear programs, called the simplex method.

The following is an example of a linear program:

$$\begin{array}{rlcl}
\max & 2x_1 + 3x_2 + 5x_3 & & \\
\text{subject to} & 5x_1 + 2x_2 + 3x_3 & \leq & 35 \\
 & 3x_1 \qquad\;\; + 4x_3 & \geq & 3 \\
 & 1x_1 + 5x_2 - 3x_3 & = & 6 \\
 & x_1 & \geq & 0 \\
 & \qquad\qquad\quad x_3 & \leq & 0\,,
\end{array} \tag{2.1}$$

with *decision variables* x_1, x_2, x_3 and linear objective function $2x_1 + 3x_2 + 5x_3$ to

[1] We use the abbreviation LP for both terms "linear programming" and "linear program".

be maximized. The linear equalities and inequalities that have to be fulfilled are called the *constraints* of the LP. Often the simple constraints that restrict the sign of a single decision variable are written separately from the remaining constraints (see below). Variables without explicit restriction on their sign imposed by a simple constraint, like variable x_2 in (2.1), are called *free variables*. In case of a minimization problem, the objective function is usually referred to as *cost function*, whereas for a maximization problem it is often called *profit function*. A realization of the decision variables that satisfies all constraints is called a *feasible solution*. If, in addition, the feasible solution minimizes (resp. maximizes) the objective function, it is called an *optimal solution* with associated *optimal cost* (resp. *optimal profit*). The set of all feasible solutions defines the *feasible region* of the LP. In case of an empty feasible region, i.e., no feasible solutions exist, the LP is said to be *infeasible*. Otherwise, if for a given LP the feasible region is non-empty but no optimal solutions exists, i.e., the objective function value can be arbitrarily improved, the optimal cost (resp. profit) as well as the LP itself are said to be *unbounded*.

Standard form

For the analysis of LPs and the discussion of related algorithms it is convenient to define a standard representation of LPs. The following representation of an LP is called the *standard form*:

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}\,. \end{aligned} \tag{2.2}$$

The n-dimensional vector $\mathbf{c} = (c_1, \ldots, c_n)^\top$ of objective function coefficients is called the *cost vector* and $\mathbf{A}$ is the $m \times n$ *constraint matrix*. The i-th row a_i for $i = 1, \ldots, m$ of matrix $\mathbf{A}$ is an n-dimensional vector $(a_{i,1}, \ldots, a_{i,n})$ containing the coefficients of the i-th constraint given by

$$a_{i,1}x_1 + a_{i,2}x_2 + \cdots + a_{i,n}x_n = b_i\,.$$

Every LP formulation can be transformed into an equivalent LP formulation in standard form such that every feasible solution to the standard form problem can be directly converted into a feasible solution to the other problem with the same cost (resp. profit after multiplication with -1). We will use the example LP (2.1) to demonstrate the transformation rules to generate an equivalent standard form LP. As a first step, we transform (2.1) into a minimization problem by multiplying the profit vector by -1. Therefore, the new objective function reads

$$\min \quad -2x_1 - 3x_2 - 5x_3\,.$$

Next, we transform the two inequality constraints into equality constraints by introducing a *slack variable* x_1^s and a *surplus* variable x_2^s, both non-negative, into the first and second constraint, which then become

$$5x_1 + 2x_2 + 3x_3 + x_1^s = 35 \quad \text{and} \quad 3x_1 + 4x_3 - x_2^s = 3\,.$$

In the final two steps, we replace the free variable x_2 and the non-positive restricted variable x_3 by equivalent non-negative variables. We substitute x_3 by some variable $\hat{x}_3$, which is defined as $\hat{x}_3 := -x_3$ and therefore restricted to non-negative values. Finally, we substitute the free variable x_2 by the difference $x_2^+ - x_2^-$ of two non-negative variables x_2^+ and x_2^-. After applying these transformations, we obtain the following LP:

$$\begin{array}{rlcl}
\min & -2x_1 - 3x_2^+ + 3x_2^- + 5\hat{x}_3 & & \\
\text{subject to} & 5x_1 + 2x_2^+ - 2x_2^- - 3\hat{x}_3 + x_1^s & = & 35 \\
& 3x_1 \qquad\qquad\quad - 4\hat{x}_3 \qquad - x_2^s & = & 3 \\
& 1x_1 + 5x_2^+ - 5x_2^- + 3\hat{x}_3 & = & 6 \\
& x_1, x_2^+, x_2^-, \hat{x}_3, x_1^s, x_2^s & \geq & 0\,,
\end{array}$$

which is equivalent to (2.1) and obviously in standard form (2.2) with

$$\mathbf{c} = \begin{bmatrix} -2 \\ -3 \\ 3 \\ 5 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 5 & 2 & -2 & -3 & 1 & 0 \\ 3 & 0 & 0 & -4 & 0 & -1 \\ 1 & 5 & -5 & 3 & 0 & 0 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2^+ \\ x_2^- \\ \hat{x}_3 \\ x_1^s \\ x_2^s \end{bmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 35 \\ 3 \\ 6 \end{bmatrix}.$$

A geometric view on linear programming

To understand the general idea of solving LPs, it is helpful to consider linear programming from a geometric perspective. Before we discuss the geometric aspects of linear programming, we begin with some basic geometric definitions.

A set S that can be described by $S = \{x \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \geq \mathbf{b}\}$ with $m \times n$ matrix $\mathbf{A}$ and $\mathbf{b} \in \mathbb{R}^m$ defines a *polyhedron*. If the absolute value of every component x_i of every element $\mathbf{x} \in S$ is bounded, then S is a *bounded* polyhedron, sometimes denoted as *polytope*. For every polyhedron S, its dimension $\dim(S)$ is defined as the maximum number of affinely independent points in S minus one.

For polyhedra defined by a single linear constraint two possible cases are distinguished. If the single constraint is an equality, i.e.,

$S = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^\top \mathbf{x} = b \wedge \mathbf{a} \in \mathbb{R}^n\}$, then S is called a *hyperplane*. Otherwise, if the single constraint is an inequality, i.e., $S = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^\top \mathbf{x} \geq b \wedge \mathbf{a} \in \mathbb{R}^n\}$, then S is called a *half-space*. According to these definitions, every polyhedron can be described as the intersection of a finite number of half-spaces, which is sometimes called the *outer description* of a polyhedron. Another important property of polyhedra is that they define *convex sets*, meaning that whenever some point $\mathbf{p} \in \mathbb{R}^n$ belongs to a line segment between any two points in S, then $\mathbf{p}$ must also be an element of S.

From these definitions it follows that the feasible region of every LP defines a polyhedron P. An important observation is that whenever an LP has a unique or multiple optimal solutions, at least one optimal solution will correspond to an *extreme point* of P. The extreme points represent the corners of P, also denoted as *vertices*, which are the points that do not correspond to the convex combination of any two other points in P.

In addition to the outer description, every bounded polyhedron has an alternative description based on extreme points. For a finite set of points $X = \mathbf{x}_1, \ldots, \mathbf{x}_n$, the *convex hull* of X (CH(X)) is defined as the set of convex combinations of all points in X:

$$\mathrm{CH}(X) = \{\mathbf{y} \mid \mathbf{y} = \sum_{i=0}^{n} \lambda_i \mathbf{x}_i \ \wedge \ \sum_{i=0}^{n} \lambda_i = 1 \ \wedge \ \boldsymbol{\lambda} \in \mathbb{R}^n_+\}\,.$$

Using this definition, every bounded polyhedron P can be described as the convex hull of all extreme points of P, sometimes called the *inner description* of P. Note that for unbounded polyhedra there exists a similar representation based on extreme points and so-called *extreme rays* (details omitted). For a given polyhedron P, we say that an inequality constraint $\mathbf{a}^\top \mathbf{x} \leq b$ is *valid* for P if the half-space $S = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^\top \mathbf{x} \leq b\}$ contains P, i.e., $P \subseteq S$.

In Figure 2.1 we present a two-dimensional example ($n = 2$) to illustrate the geometric representation of a general LP formulation. As we can observe in this example, every constraint defines an $(n-1)$-dimensional hyperplane that separates the space $\mathbb{R}^n$ into two half-spaces such that exactly one of them contains the feasible region. Obviously, in $\mathbb{R}^2$ every hyperplane is of dimension one and presents a line. Only four of the five hyperplanes in the example are *supporting hyperplanes*, meaning they have a non-empty intersection with the feasible region and therefore define its boundaries, called *faces*. A face $F = P \cap \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^\top \mathbf{x} \geq b \wedge \mathbf{a} \in \mathbb{R}^n\}$ is a *proper face* if $F \subset P$, and, depending on their dimension, proper faces have special names like “vertex” and “edge” for faces of dimension zero and one respectively. Every vertex of a polyhedron corresponds to the intersection of at least n supporting hyperplanes. Another important class of faces are the $(n-1)$-dimensional *facets* of the polyhedron, which

$$
\begin{array}{lrrl}
\text{max} & 14x_1 + & 5x_2 & \\
\text{subject to} & x_1 + & 2x_2 & \geq 4 \\
 & 2x_1 + & 3x_2 & \leq 15 \\
 & x_1 - & x_2 & \leq 2 \\
 & -2x_1 + & x_2 & \leq 1 \\
 & & x_2 & \leq 5 \\
 & x_1, & x_2 & \geq 0
\end{array}
$$

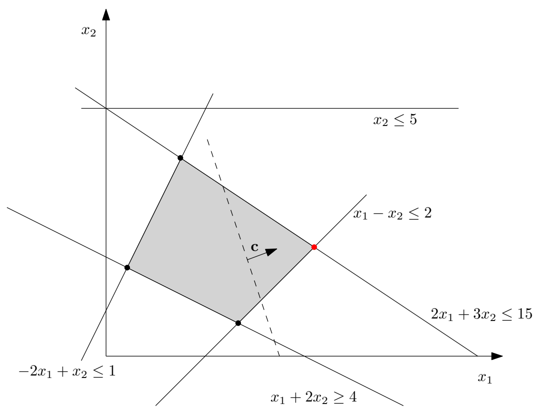


Figure 2.1: *Graphical geometric representation of an LP in* $\mathbb{R}^2$. *Every line is annotated with the corresponding inequality. Vertices of the feasible region (gray area) are denoted by dots and the red dot marks the optimal solution* $\mathbf{x} = (4.2, 2.2)^\top$ *with optimal profit* 69.8. *The optimal solution is obtained by moving the dashed line, which is perpendicular to the vector* $\mathbf{c} = (14, 5)^\top$, *as far as possible into direction* $\mathbf{c}$ *without leaving the feasible region. Note that inequality* $x_2 \leq 5$ *does not define a supporting hyperplane.*

are of particular interest, as they are the only hyperplanes required to completely define the polyhedron. In the two-dimensional example in Figure 2.1, four of the five constraints define supporting hyperplanes, each containing one of the four facets of the polyhedron. Further, in the two-dimensional example the optimal solution can be obtained by drawing the line that is perpendicular to the profit vector and moving it as far as possible towards the direction of the profit vector such that it still intersects the feasible region. This intersection then contains all optimal feasible solutions of the LP.

An algebraic view on linear programming

In addition to the geometric definition of the corners of a polyhedron as extreme points or vertices, there exists an equivalent algebraic definition. Given a polyhedron P defined by a set of equality and inequality constraints, every vector $\mathbf{x}^* \in \mathbb{R}^n$ that fulfills n linearly independent constraints, including all equality constraints, with equality is called a *basic solution*. The corresponding constraints fulfilled with equality are denoted as *active* constraints. If, in addition, the vector $\mathbf{x}^*$ does not violate any of the remaining constraints, it is called a *basic feasible solution*.

For a given LP in standard form with $m \times n$ constraint matrix $\mathbf{A}$, every basic feasible solution $\mathbf{x}^*$ corresponds to a certain subset of column indices $\mathcal{B} = \mathcal{B}_1, \ldots, \mathcal{B}_m$ such that the *basic columns* $\mathbf{A}_{\mathcal{B}_1}, \ldots, \mathbf{A}_{\mathcal{B}_m}$ are linearly independent and all components of $\mathbf{x}^*$, except $x^*_{\mathcal{B}_1}, \ldots, x^*_{\mathcal{B}_m}$, have value zero. The set $\mathcal{B}$ is called a *basis* with corresponding vector of *basic variables* $\mathbf{x}_{\mathcal{B}} = (x_{\mathcal{B}_1}, \ldots, x_{\mathcal{B}_m})$ and $m \times m$ *basis matrix* $\mathbf{B} = [\mathbf{A}_{\mathcal{B}_1}, \ldots, \mathbf{A}_{\mathcal{B}_m}]$.

A pair of distinct basic solutions that share $(n-1)$ basic columns are said to be *adjacent basic solutions* with *adjacent bases*. With these definitions we are now prepared for a brief outline of the predominantly applied simplex method, which employs the equivalence between extreme points of the polyhedron and basic feasible solutions of the LP.

2.1.1 The simplex method

From a geometric perspective, the simplex method finds the optimal solution of a given LP in standard form by *traversing* the extreme points of the feasible region until it identifies an optimal solution. From an algebraic perspective, in every iteration of this search, the simplex method moves from one basic feasible solution to an adjacent basic feasible solution such that the cost is improved. One important property of linear programs is that every local optimal solution is also a global optimal solution, since a convex function is optimized over a convex set of feasible solutions.

Therefore, in every iteration the simplex method determines whether the cost of the current basic feasible solution is at most the cost of all adjacent basic feasible solutions. In this case, the algorithm stops since a local, and therefore also global, optimal solution has been found. Otherwise, it proceeds to one of the adjacent basic feasible solutions with improved cost and starts the next iteration. Finally, the simplex method will either find an optimal basic feasible solution, or it will determine that the problem is unbounded and no optimal solution exists.

We will now briefly review the most important steps of a single iteration of the simplex method. Starting at a basic feasible solution $\mathbf{x}^*$ with basis $\mathcal{B}$, the first step is to determine whether $\mathbf{x}^*$ is an optimal solution or whether the cost can be improved by moving to an adjacent basic feasible solution. Moving to an adjacent basic solution means to construct a new basic feasible solution by replacing one basic column $\mathbf{A}_{j'}$ with a nonbasic column $\mathbf{A}_j$. Consequently the value of the *leaving variable* $x_{j'}$ will be set to zero, while the value of the *entering variable* x_j can possibly be increased to a value greater than zero.

The first step is to determine the entering variable by computing for every nonbasic variable x_j the *reduced cost* $\bar{c}_j$, defined as

$$\bar{c}_j := c_j - \mathbf{c}_{\mathcal{B}}^\top \mathbf{B}^{-1} \mathbf{A}_j \ , \tag{2.3}$$

with $\mathbf{c}_{\mathcal{B}} = (c_{\mathcal{B}_1}, \ldots, c_{\mathcal{B}_m})$. The reduced cost intuitively represents the rate at which the cost changes when the value of variable x_j is increased while maintaining feasibility. If the reduced costs of all nonbasic variables are non-negative, the current basic feasible solution $\mathbf{x}^*$ is an optimal solution and the algorithm stops. Otherwise, one of the nonbasic variables x_j with negative reduced cost $\bar{c}_j$ is selected as entering variable.

Once the entering variable has been determined, the next step is to identify the variable that will leave the basis. Increasing the value of variable x_j corresponds to moving along the j-th *basic direction* $\mathbf{d} \in \mathbb{R}^n$ with components given by

$$d_j = 1\,, \quad \mathbf{d}_{\mathcal{B}} = -\mathbf{B}^{-1}\mathbf{A}_j\,, \quad \text{and} \quad d_i = 0 \;\forall\; i \notin \mathcal{B} \cup \{j\}\,.$$

If all entries of $\mathbf{d}$ are non-negative, it is possible to move infinitely far towards direction $\mathbf{d}$ without leaving the feasible region, and therefore the algorithm terminates as the problem is unbounded. Otherwise, for every $d_{\mathcal{B}_i} < 0$, the value $\Theta_{\mathcal{B}_i} = -x^*_{\mathcal{B}_i}/d_{\mathcal{B}_i}$ is computed, which indicates how far to move into direction $\mathbf{d}$ until the basic variable $x_{\mathcal{B}_i}$ becomes zero. Let $\Theta_{\mathcal{B}_k}$ be the minimum of all computed $\Theta_{\mathcal{B}_i}$, then $x_{\mathcal{B}_k}$ is selected as leaving variable and is set to zero. The new basic feasible solution $\hat{\mathbf{x}}^*$ is then constructed as follows:

$$\hat{x}^*_j = \Theta_{\mathcal{B}_k}, \quad \hat{x}^*_{\mathcal{B}_k} = 0 \quad \text{and} \quad \hat{x}^*_{\mathcal{B}_i} = x^*_{\mathcal{B}_i} + \Theta_{\mathcal{B}_k} d_{\mathcal{B}_i}, \;\forall\; i \neq k\,.$$

Finally, updating the basis by setting $\mathcal{B}_k$ to j completes the iteration.

Degeneracy

Problems for the simplex method can arise in case of *degenerate* basic feasible solutions, which are basic solutions with more than n active constraints. In the context of the simplex algorithm, a degenerate basis corresponds to basic variables with value zero, which can cause infinite cycling through different bases, all defining the same basic solution. This cycling can be avoided by clever rules for the selection of entering and leaving variables, e.g., Bland's anti cycling rule [Bland, 1977], such that the simplex method is guaranteed to either find an optimal solution or determine unboundedness after a finite number of iterations.

Initial basic feasible solution

Before the simplex method can be applied to solve an LP, it requires an initial feasible basis and the corresponding basic feasible solution. Therefore, the process of solving an LP is usually divided into two *phases*. In phase I, an initial solution is computed by solving an *auxiliary* LP that corresponds to the original LP with additional *artificial* variables. For an LP in standard form with the additional property that all entries of the right hand side vector $\mathbf{b}$ are non-negative, the auxiliary LP with artificial variables $y_1, \ldots, y_m$ is given by

$$\begin{aligned} \min \quad & y_1 + y_2 + \ldots + y_m \\ \text{subject to} \quad & \mathbf{Ax} + \mathbf{y} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \\ & \mathbf{y} \geq \mathbf{0} \,. \end{aligned}$$

This problem has a trivial initial basic feasible solution given by $\mathbf{x} = \mathbf{0}$, $\mathbf{y} = \mathbf{b}$, and the $m \times m$ identity matrix as initial basis matrix. Using this initial basis, the simplex method can be applied to solve the auxiliary LP. If the optimal cost of the auxiliary LP is not equal to zero, the original LP must be infeasible. Otherwise, the returned optimal solution can be transformed into a initial basic feasible solution of the original LP such that the simplex method can be applied (phase II).

Duality

An important concept in linear programming, also employed in a variant of the simplex method, is linear programming *duality*. For any given *primal* LP in standard form with $m \times n$ matrix $\mathbf{A}$, its corresponding *dual* LP is defined as follows:

Primal: Dual:

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned} \qquad \Longleftrightarrow \qquad \begin{aligned} \max \quad & \mathbf{b}^\top \mathbf{u} \\ \text{subject to} \quad & \mathbf{A}^\top \mathbf{u} \leq \mathbf{c} \\ & \mathbf{u} \in \mathbb{R}^m \,. \end{aligned}$$

For every equality constraint in the primal LP, the dual LP problem contains a free variable, and every variable in the primal LP has a corresponding constraint in the dual LP. The important relation between the primal LP and its dual is stated by two theorems known as the *weak duality theorem* and the *strong duality theorem*. Given an arbitrary feasible solution $\mathbf{x}^*$ to the primal problem and an arbitrary feasible solution $\mathbf{u}^*$ to the dual problem, the weak duality theorem states that $\mathbf{c}^\top \mathbf{x}^* \geq \mathbf{b}^\top \mathbf{u}^*$. The points $\mathbf{x}^*$ and $\mathbf{u}^*$ are also said to be *primal feasible* and *dual feasible* respectively. Additionally, the strong duality theorem states that if an LP has an optimal solution $\hat{\mathbf{x}}^*$, then its dual problem must also have an optimal solution $\hat{\mathbf{u}}^*$ and it holds that $\mathbf{c}^\top \hat{\mathbf{x}}^* = \mathbf{b}^\top \hat{\mathbf{u}}^*$. Based on the *complementary slackness* property (not shown) between primal and dual problem, an optimal solution to the primal problem can be derived from an optimal solution to the dual problem and vice versa. The complementary slackness property also underlies the so-called *dual simplex* method that solves an LP by maintaining dual feasibility and working towards a primal feasible and hence optimal solution. Working with the dual simplex can be particularly favorable in scenarios where the problem is repeatedly solved and modified by changing the right hand side $\mathbf{b}$ or by adding additional constraints. The latter usually occurs when solving integer linear programs as discussed in Section 2.2. These modifications can render the previous optimal basic feasible solution of the primal LP infeasible, and therefore, the modified problem must be solved starting from a new initial basic feasible solution. In contrast, the previous optimal feasible basis remains feasible for the dual of the modified problem, but it is possibly no longer optimal. Therefore, the dual simplex can start from the previous optimal basis, which often requires only a few simplex iterations to find the optimal solution of the modified problem.

Complexity of the simplex method

Although the simplex method may require an exponential number of iterations to find an optimal solution in the worst case, in practical observations it "typically" requires only $\mathcal{O}(m)$ iterations [Bertsimas and Tsitsiklis, 1997]. Other algorithms, called interior points methods, that start at an internal point of the polyhedron and move in the interior of the feasible regions towards the optimal solution, have a theoretical polynomial worst case time complexity, and in some cases they outperform the simplex method in practical applications. Nevertheless, especially in integer linear programming applications the (dual) simplex method performs very well due to its practical efficiency in re-optimizing an LP that has been modified by additional constraints.

Delayed column generation

For some optimization problems, the number of variables of the associated LP formulation in standard form is too large to generate and hold the complete constraint matrix $\mathbf{A}$ in memory. *Delayed column generation* is a linear programming technique to handle such large scale problems, based on the observation that during the complete course of the Simplex algorithm most of these columns usually never enter the basis. Hence, a feasible approach to solve some of these large scale problems is to circumvent the generation of such columns. Therefore, starting from an initial basic feasible solution computed for a smaller subset of all variables, in every iteration of the simplex algorithm with basis matrix $\mathbf{B}$ and associated *simplex multipliers* $\boldsymbol{\rho}^\top = \mathbf{c}_B^\top \mathbf{B}^{-1}$, a secondary algorithm is applied to either identify a variable x_i with negative reduced cost $\bar{c}_i = c_i - \boldsymbol{\rho}^\top \mathbf{A}_i$ or prove that no such variable exists. The latter case then proves the optimality of the current basic solution. The subproblem of identifying variables with negative reduced cost, called *pricing problem*, is often stated as an optimization problem searching for a variable with minimum reduced cost. In this way, some LPs involving an exponential number of variables can be solved efficiently if the associated pricing problem has a certain structure that allows for efficient identification of variables with minimum reduced cost.

2.2 Integer linear programming

The following descriptions of integer linear programming methods are based on the textbook [Bertsimas and Tsitsiklis, 1997, chap. 11].

Many optimization problems that can be formulated as linear programs have the additional requirement that some or all of the decision variables are restricted to be integer numbers, as they represent undividable entities. A linear program where all the decision variables are restricted to integer values is called an *integer linear program*[2] (ILP). If only a fraction of decision variables are restricted to integers, it is called *mixed integer program* (MIP). The special case of an ILP where all the decision variables are restricted to be either zero or one is sometimes referred to as *zero-one integer program* (ZOIP). However, in the remainder of this thesis, we will not distinguish between ILP and ZOIP and refer to both as ILP.

Unlike LPs, which can be solved in polynomial time, no efficient algorithm is known for solving a general ILP, and the problem is known to be NP-hard. Nevertheless, there exist algorithmic approaches that allow to solve many real life ILP instances in a reasonable amount of time despite their exponential worst case complexity. We will now introduce two of these algorithmic approaches, known as *branch and cut* and *Lagrangian relaxation*, both of which we apply to the ILP formulations presented in Chapter 3 and Chapter 5.

2.2.1 Branch and cut

The Branch and cut method for solving ILPs is based on the combination of the two methods branch and bound and cutting planes.

[2] We use the abbreviation ILP for both terms “integer linear programming” and “integer linear program”.

Cutting planes

For a given ILP defined by

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}_+^n \,, \end{aligned}$$

the corresponding *LP-relaxation* is obtained by removing the integrality restriction:

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \,. \end{aligned}$$

The general idea of cutting plane algorithms for integer linear programming is to repeatedly solve and modify the LP-relaxation. Whenever the optimal solution $\mathbf{x}^*$ of the LP-relaxation is integer, then $\mathbf{x}^*$ defines also an optimal solution to the original ILP and the algorithm terminates. Otherwise, if $\mathbf{x}^*$ contains fractional elements, new inequalities are added to the LP-relaxation that are violated by $\mathbf{x}^*$ but fulfilled by all feasible solutions of the ILP, and the modified LP-relaxation is re-optimized (see example in Figure 2.2). These new inequalities are called *cutting planes*, or simply *cuts*, as they "cut off" the fractional solution $\mathbf{x}^*$.

One example for cutting plane algorithms is the well known cutting plane algorithm proposed by Gomory [1958]. While this algorithm has been proven to solve general ILPs within a finite number of iterations, its practical performance is often insufficient.

The practical performance of every cutting plane algorithm depends on the strength of the cutting planes, that is, how much of the feasible region of the LP-relaxation not containing any integer feasible solutions can be cut off by the new inequality. To compare the strength of two valid cuts $\boldsymbol{\pi}_0^\top \mathbf{x} \leq \mu_0$ and $\boldsymbol{\pi}_1^\top \mathbf{x} \leq \mu_1$, we say that $\boldsymbol{\pi}_0^\top \mathbf{x} \leq \mu_0$ *dominates* $\boldsymbol{\pi}_1^\top \mathbf{x} \leq \mu_1$ if

$$\{\mathbf{x} \in \mathbb{R}_+ \mid \boldsymbol{\pi}_0^\top \mathbf{x} \leq \mu_0\} \subseteq \{\mathbf{x} \in \mathbb{R}_+ \mid \boldsymbol{\pi}_1^\top \mathbf{x} \leq \mu_1\} \,.$$

If, in addition, the two cuts do not define the same hyperplane, i.e., $(\boldsymbol{\pi}_0, \mu_0) \neq \lambda(\boldsymbol{\pi}_1, \mu_1)$ for all $\lambda \in \mathbb{R}$, we say that $\boldsymbol{\pi}_0^\top \mathbf{x} \leq \mu_0$ *strictly dominates* $\boldsymbol{\pi}_1^\top \mathbf{x} \leq \mu_1$.

The central problem in the design of cutting plane algorithms is the *separation problem* defined as follows: Given a polyhedron $P \subset \mathbb{R}^n$ and a point $\mathbf{x}^* \in \mathbb{R}^n$, decide whether $\mathbf{x}^* \in P$ and if not, find an inequality $\boldsymbol{\pi}^\top \mathbf{x} \leq \mu$ that separates P from $\mathbf{x}^*$, i.e., $\boldsymbol{\pi}^\top \mathbf{y} \leq \mu \ \forall \mathbf{y} \in P$ and $\boldsymbol{\pi}^\top \mathbf{x}^* > \mu$. While there exist general purpose cutting planes, like Gomory's cuts, usually stronger, problem specific cutting planes can be identified by exploiting the polyhedral structure of the particular problem.

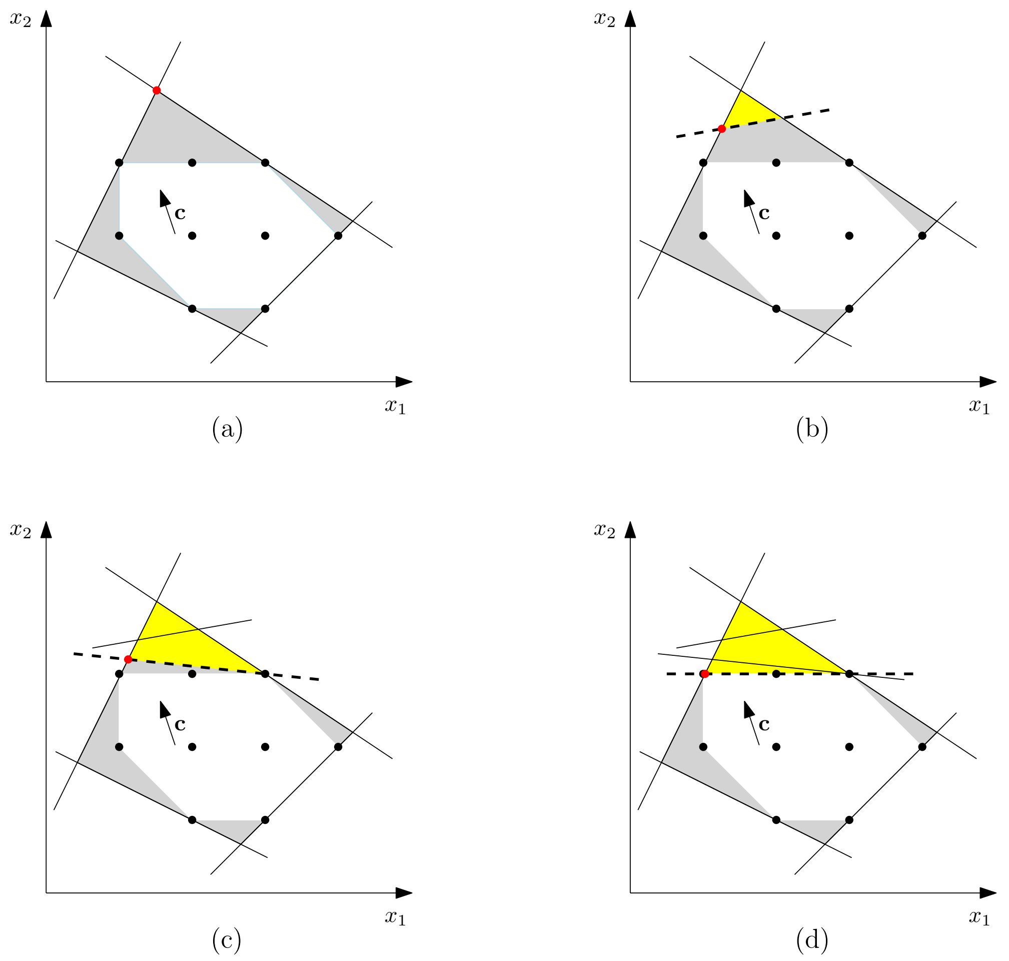


Figure 2.2: *Cutting planes example: In (a)-(d) the white region represents the convex hull P_{IP} of all feasible solutions of some ILP defined by four inequalities, and the grey region denotes the feasible region of the associated LP-relaxation. Red dots mark the optimal solution for the current LP-relaxation, and dashed lines represent cutting planes subsequently added to the problem. In (d) the optimal solution of the LP-relaxation with three cutting planes equals the optimal solution of the original ILP. The last cutting plane added contains a facet of P_{IP}.*

A particularly important class of cutting planes are inequalities that define the facets of the polyhedron $P_{IP} = \text{CH}(\{\mathbf{x} \in \mathbb{Z}_+^{\times} \mid \mathbf{Ax} \leq \mathbf{b}\})$. Like every polyhedron, P_{IP} can be completely described by only *facet defining inequalities*, and therefore these inequalities represent the strongest cuts to be separated in a cutting plane algorithm. An important theorem based on the work of Grötschel et al. [1981] states that the optimization problem for a given polyhedron is polynomial time solvable if and only if the separation problem is polynomial time solvable. This implies that the separation problem for NP-hard problems is also NP-hard, which implies that the separation of certain classes of facet defining inequalities is also NP-hard or not all classes of facet defining inequalities are known. As a consequence, the practical performance of pure cutting plane algorithms is often insufficient and the cutting plane approach is embedded into a branch and bound framework.

Branch and bound

Branch and bound is a divide and conquer approach to solve an optimization problem over some set of feasible solutions S by partitioning S into subsets $S_1, S_2, \ldots, S_n$ with $S_1 \cup \ldots \cup S_n = S$ and solving the optimization problem for each of the subsets separately. This process of partitioning into subproblems, called *branching*, is usually performed recursively, resulting in a branch tree as depicted in Figure 2.3. In the remainder, we will consider the case of a minimization problem. While the original problem and even the subproblems can be computationally hard to solve, in many cases a lower bound for the optimal value can be computed efficiently. We will denote this lower bound computed for some subproblem S_i as its associated *local lower bound* $LLB(S_i)$. Whenever one of the subproblems can be solved optimally, this solution constitutes a feasible solution to the original problem, and therefore its associated cost $\text{cost}(S_i)$ defines an upper bound to the optimal cost of the original problem. Whenever $\text{cost}(S_i)$ is lower than the lowest upper bound computed so far, called *global upper bound* GUB, we update GUB to $\text{cost}(S_i)$, and the associated solution of S_i is marked as *incumbent* solution.

The central idea of the branch and bound algorithm is that whenever the lower bound for some subproblem S_i is greater than the current value of GUB, i.e., $LLB(S_i) \geq GUB$, S_i and all subproblems $S_j \subseteq S_i$ can be discarded, since their optimal costs will also be greater than GUB. According to the representation in a branch tree this process is often denoted as *pruning*.

When solving some ILP IP by branch and bound, an obvious choice for computing a lower bound is to solve the LP-relaxation of IP. If the optimal solution $\mathbf{x}^*$ of this relaxation is integer, it is also a feasible solution for IP. Otherwise, if the obtained lower bound does not allow for pruning, some variable x_k with fractional optimal solution value x_k^* is selected for branching, and two subproblems are generated, the first by adding constraint $x_k \leq \lfloor x_k^* \rfloor$ and the second by adding constraint $x_k \geq \lceil x_k^* \rceil$. Obviously, the union of all feasible solutions for the two subproblems equals the set of all feasible solutions for *IP*, but the fractional solution $\mathbf{x}^*$ is not a feasible solution to the LP-relaxation of either subproblem. Therefore, the optimal solutions of the LP-relaxation for both subproblems will be different from $\mathbf{x}^*$ and, since the feasible regions of the subproblems are disjoint, they will also be distinct.

Combining branch and bound with cutting planes

In the branch and bound approach for ILPs, as described above, the lower bound for every subproblem is obtained by solving the LP-relaxation. A crucial factor that determines the performance of the algorithm is the quality of the computed bounds, since tighter bounds allow for more rigorous pruning and can therefore lead to a significant reduction of the number of subproblems to be considered. To obtain tighter bounds, the branch and cut algorithm embeds cutting planes into the branch and bound algorithm to improve the bounds for every subproblem by iteratively adding cutting planes and re-solving the LP-relaxation. A schematic workflow of the branch and cut algorithm is depicted in Figure 2.3. The practical performance of a branch and cut algorithm depends on several parameters like, e.g.,

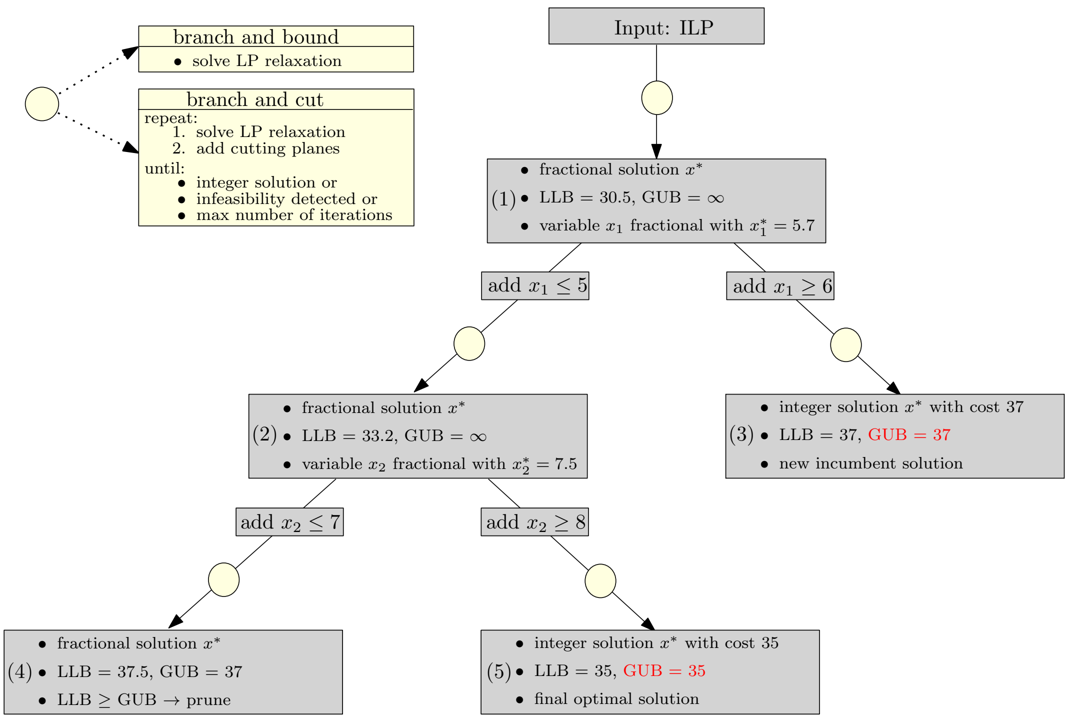
- the choice of the branching variable if multiple fractional variables exist (branching rule),
- the order of subproblem processing,
- the choice of cutting planes to be separated, and
- the maximum number of cutting planes generated per subproblem,

which are often adjusted for the particular problem.

2.2.2 Lagrangian relaxation

An alternative approach to obtain tight bounds for the optimal cost of ILPs is Lagrangian relaxation. This method is motivated by the observation that some difficult ILPs could be solved efficiently if only some of the constraints are ignored. Given the ILP *IP* defined by

$$\begin{aligned} Z_{IP} = \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{D}\mathbf{x} \geq \mathbf{d} \\ & \mathbf{x} \in \mathbb{Z}_+^n , \end{aligned} \tag{2.4}$$



branch and bound
• solve LP relaxation
branch and cut
repeat:
1. solve LP relaxation
2. add cutting planes
until:
• integer solution or
• infeasibility detected or
• max number of iterations
Input: ILP
(1) • fractional solution x^* • LLB = 30.5, GUB = ∞ • variable x_1 fractional with $x_1^* = 5.7$
add $x_1 \leq 5$
add $x_1 \geq 6$
(2) • fractional solution x^* • LLB = 33.2, GUB = ∞ • variable x_2 fractional with $x_2^* = 7.5$
(3) • integer solution x^* with cost 37 • LLB = 37, GUB = 37 • new incumbent solution
add $x_2 \leq 7$
add $x_2 \geq 8$
(4) • fractional solution x^* • LLB = 37.5, GUB = 37 • LLB $\geq$ GUB $\rightarrow$ prune
(5) • integer solution x^* with cost 35 • LLB = 35, GUB = 35 • final optimal solution

Figure 2.3: *An example workflow of a branch and bound (resp. branch and cut) algorithm for ILPs. Numbers represent the order of subproblem computation. In case of a pure branch and bound algorithm, the bound computation step represented by yellow circles corresponds to solving the LP-relaxation of the subproblem once. In a branch and cut algorithm, the bounds are iteratively improved by cutting planes until some termination criterion is met. Subtrees under subproblem (3) and (5) are pruned due to integer feasibility, while subtree under subproblem (4) is pruned by bounding.*

where the constraints are divided into two sets, $\mathbf{Ax} \geq \mathbf{b}$ and $\mathbf{Dx} \geq \mathbf{d}$, of cardinality m_A and m_D. Now, assume this problem is efficiently solvable after *relaxing* (i.e., removing) the set of constraints $\mathbf{Dx} \geq \mathbf{d}$. Obviously, the optimal cost for this relaxed problem serves as a lower bound to the optimal cost of *IP*. In many cases this bound will not be very tight, since the relaxation allows for violating probably very important constraints. The idea of Lagrangian relaxation is to generate tighter bounds by penalizing the violation of relaxed constraints by a certain factor, called *Lagrange multiplier*. This leads to the *Lagrangian relaxation*

LR of the problem *IP*, defined as

$$\begin{aligned} Z(\boldsymbol{\lambda}) = \min \quad & \mathbf{c}^\top \mathbf{x} + \boldsymbol{\lambda}(\mathbf{d} - \mathbf{D}\mathbf{x}) \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^n_+ , \end{aligned} \tag{2.5}$$

with $\boldsymbol{\lambda} \in \mathbb{R}^{m_D}_+$ denoting the vector of Lagrange multipliers.

According to this definition, for every non-negative choice of $\boldsymbol{\lambda}$, the optimal cost $Z(\boldsymbol{\lambda})$ of problem *LR* constitutes a lower bound for the optimal cost Z_{IP} of problem *IP*. To obtain the tightest possible bound, $Z(\boldsymbol{\lambda})$ is maximized over all $\boldsymbol{\lambda} \in \mathbb{R}^{m_D}_+$, known as the *Lagrangian dual* problem:

$$Z_D = \max_{\boldsymbol{\lambda} \geq \mathbf{0}} Z(\boldsymbol{\lambda}) . \tag{2.6}$$

Solving the Lagrangian dual problem

An interesting property of the function $Z(\boldsymbol{\lambda})$ is that this function is piecewise linear and concave. A popular approach to maximize a concave and differentiable function $f(u)$ is the steepest ascent method, which starts at some initial point u^0 and iteratively computes the sequence $u^1, u^2, \ldots u^r$ according to the formula

$$u^{k+1} = u^k + \mu_k \nabla f(u^k) ,$$

where $\nabla f(u^k)$ defines the gradient of function f at point u^k and μ_k is a non-negative step size parameter. Since function $Z(\boldsymbol{\lambda})$ is piecewise linear and therefore not differentiable, the gradient $\nabla Z(\boldsymbol{\lambda})$ is not always defined for $\boldsymbol{\lambda} \in \mathbb{R}^{m_D}_+$. Therefore, the frequently used *subgradient algorithm* to solve the Lagrangian dual employs a generalization of the gradient of some function. For any given concave function $f : \mathbb{R}^n \to \mathbb{R}$, a vector $\mathbf{s}$ is called a *subgradient* of f at point $\mathbf{u}^* \in \mathbb{R}^n$ if it satisfies

$$f(\mathbf{u}) \leq f(\mathbf{u}^*) + \mathbf{s}^\top (\mathbf{u} - \mathbf{u}^*) ,$$

for all $\mathbf{u} \in \mathbb{R}^n$. It holds that some point $\mathbf{u}^*$ maximizes f if and only if $\mathbf{0}$ defines a subgradient of f at point $\mathbf{u}^*$. It can be shown that for a given solution $\mathbf{x}^*$ of problem *LR* at point $\boldsymbol{\lambda}^*$, the vector $(\mathbf{d} - \mathbf{D}\mathbf{x}^*)$ defines a subgradient of $Z(\boldsymbol{\lambda})$ at point $\boldsymbol{\lambda}^*$. Hence, the subgradient algorithm for the Lagrangian dual problem iteratively computes the sequence of nonnegative values $\boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2, \ldots, \boldsymbol{\lambda}^r$ according to the formula

$$\boldsymbol{\lambda}^{k+1} = \max\{\boldsymbol{\lambda}^k + \mu_k(\mathbf{d} - \mathbf{D}\mathbf{x}^k), \, \mathbf{0}\} ,$$

where $\mathbf{x}^k$ denotes the optimal solution of problem *LR* at point $\boldsymbol{\lambda}^k$.

The practical performance of the subgradient algorithm strongly depends on the selection of the step size parameter μ_k in each iteration k. A theoretical result states that for every sequence of step size parameters fulfilling

$$\sum_{k=1}^{\infty} \mu_k = \infty, \quad \text{and} \quad \lim_{k \to \infty} \mu_k = 0\,,$$

the sequence $Z(\boldsymbol{\lambda}^k)$ is guaranteed to converge to Z_D, unless Z_D is infinite. Nevertheless, not every choice of the step size with this property will yield a good performance, as for example the choice $\mu_k = 1/k$ leads to a very slow convergence in practice. A widely used and sophisticated choice of the step size that shows fast convergence in practice is instead given by

$$\mu_{k+1} = \frac{\gamma_k (Z^* - Z(\boldsymbol{\lambda}^k))}{\|\mathbf{d} - \mathbf{D}\mathbf{x}^k\|^2}\,, \tag{2.7}$$

where Z^* denotes some estimate for the optimal value Z_D. One approach is to update Z^* to the objective function value of *IP* at point $\mathbf{x}^k$ whenever the solution $\mathbf{x}^k$ is also feasible for *IP* and yields a lower value [Fisher, 1985]. Factor γ_t is a scalar between 0 and 2, which decreases by a given factor whenever $Z(\boldsymbol{\lambda})$ did not decrease within a specified number of iterations [Fisher, 1985].

Despite guaranteed convergence, the optimality of some value $Z(\boldsymbol{\lambda}^k)$ can only be detected if this value matches the lowest upper bound known for Z_{IP}. Since this criterion is often not met in practice, usually the subgradient method is terminated after a predefined number of iterations.

The strength of the Lagrangian dual bound

In contrast to linear programming, there exists no strong duality for ILPs and therefore the optimal value of the Lagrangian dual problem will not equal the optimal value of the original ILP in general. Therefore, to solve the ILP optimally, Lagrangian relaxation is usually embedded into a branch and bound algorithm as introduced in Section 2.2.1.

An interesting theoretical result states that the bound provided by the optimal value of the Lagrangian dual problem of an ILP *IP* is at least as tight as the bound obtained by solving the LP-relaxation of *IP*. Therefore, for certain problems, Lagrangian relaxation combined with branch and bound can achieve a very good performance and is possibly more effective than pure branch and bound or branch and cut algorithms. In practice, the performance of a Lagrangian relaxation formulation is usually a trade-off between the tightness of the relaxation and the computational complexity of the relaxed problem.

Part II

THREE APPLICATIONS

Chapter 3

de novo Peptide Sequencing With Lagrangian Relaxation

In this chapter, we present our algorithmic approach to the *de novo* peptide sequencing problem together with experimental results based on our journal publication [Andreotti et al., 2012] and the preparatory work in [Andreotti, 2008].

3.1 Background

Proteomics describes the large scale study of proteins including their structures, interactions, and their functions in metabolic pathways. While the name suggests a high degree of similarity to genomics, the study of the genome, proteomics is in general more complex. In contrast to the constant genome, the set of expressed proteins in a certain cell, tissue, or organism, called *proteome* [Wilkins et al., 1996], is highly dynamic [Tyers and Mann, 2003]. However, in order to understand the biological processes of an organism it is indispensable to perform analyses on the level of proteins, as these processes are governed by their abundances, three-dimensional structures, interactions, complexes, and post translational modifications [Tyers and Mann, 2003].

Throughout this chapter, we only consider the *primary structure* of proteins and peptides. Thus, we define a peptide as a linear chain of amino acids, referred to as *residues*, that are linked by peptide bonds between adjacent amino acids. Short peptides containing only a few residues are called *oligopeptides*, whereas longer peptide chains are referred to as *polypeptides*. The term protein then describes macromolecules constructed from one or more polypeptides [Lehninger et al., 2008]. For eucaryotes, 21 proteinogenic amino acids are known, 20 *canonical amino acids* that are directly encoded by the universal genetic code [Alberts, 2007] and the non-canonical amino acid Selenocysteine [Böck et al., 1991]. In addition, the 22nd proteinogenic amino acid, Pyrrolysine, is used by methanogenic *archaea* [Srinivasan et al., 2002]. Peptides are *directed*, as they contain an *N-terminus* with a free amino group and a *C-terminus* with a free carboxyl group. The amino acid sequence of a peptide is usually represented as a string of the one-letter amino acid codes (see Table 3.1) in N- to C-terminal direction.

Table 3.1: *Amino acid masses*
One letter code and monoisotopic masses of the 20 canonical amino acids in eukaryotes and Selenocysteine.

amino acid	code	mass (Da)		amino acid	code	mass (Da)
Glycine	G	57.02147		Aspartic acid	D	115.02695
Alanine	A	71.03712		Glutamine	Q	128.05858
Serine	S	87.03203		Lysine	K	128.09497
Proline	P	97.05277		Glutamic acid	E	129.04260
Valine	V	99.06842		Methionine	M	131.04049
Threonine	T	101.04768		Histidine	H	137.05891
Cysteine	C	103.00919		Phenylalanine	F	147.06842
Isoleucine	I	113.08407		Arginine	R	156.10112
Leucine	L	113.08407		Tyrosine	Y	163.06333
Asparagine	N	114.04293		Tryptophan	W	186.07932
				Selenocysteine	U	150.95363

For many analysis methods, an important step is the identification and quantification of individual or all proteins in a given sample. In recent years, mass spectrometry has become the standard technique for high throughput identification and quantification of proteins in complex samples.

A mass spectrometer separates ionized analytes in a sample according to their mass-to-charge ratio (m/z) to produce a mass spectrum - a two-dimensional plot that displays the absolute or relative abundance of detected ions by mass-to-charge ratio. In the remainder of this section, masses are always given in unit *Dalton* (Da), and mass-to-charge ratios are given in unit *Thomson* (Th). For the analysis of proteins, usually a bottom up workflow known as *shotgun proteomics* is applied, where proteins are first enzymatically digested into smaller peptides by a protease (usually trypsin) [Aebersold and Mann, 2003]. The resulting peptide mixture is subsequently separated by liquid chromatography (LC-MS) or high performance liquid chromatography (HPLC-MS). Separated peptides are ionized using either electrospray ionization (ESI) [Fenn et al., 1989] or matrix assisted laser desorption ionization (MALDI) [Hillenkamp et al., 1991] and subsequently analyzed by the mass spectrometer. Successfully identified peptides can be used afterwards for protein identification, usually accompanied by further analyses like quantification of individual proteins in the sample. The mass of an individual peptide or a set of peptides is usually not sufficient for a unique identification, therefore, the peptide identification usually employs a method called *tandem mass spectrometry* or MS/MS [Aebersold and Mann, 2003]. Tandem mass spectrometry couples multiple steps of mass selection and fragmentation, as depicted in Figure 3.1. In the first step (MS1) peptides with a certain m/z, called *precursor*

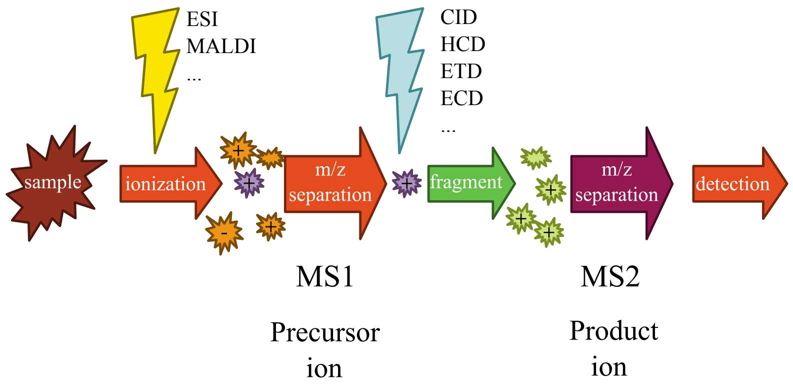


Figure 3.1: *Schematic workflow of tandem mass spectrometry (Based on K. Murray, 2006).*

ions or *parent ions*, are isolated and subsequently fragmented by methods like *collision induced dissociation* (CID) [Wells and McLuckey, 2005], *higher-energy collisional dissociation* (HCD) [Olsen et al., 2007], *electron transfer dissociation* (ETD) [Syka et al., 2004], or *electron capture dissociation* (ECD) [Zubarev et al., 1998]. Finally, the *tandem mass spectrum* of the generated *fragment ions*, also called *product ions*, is produced in the second mass spectrometry step (MS2).

The idea of tandem mass spectrometry is to identify the individual peptides based on their product ion masses. It takes advantage of the observation that, depending on the applied method, peptide fragmentation follows specific patterns. Most of the time, the precursor ions are fragmented along the *backbone* close to the peptide bond between two amino acids, producing a complementary pair of N-terminal and C-terminal fragments. Depending on the exact split position, the most prominent N-terminal product ions are referred to as a-, b-, and c-ions with their complementary x-, y-, and z-ions [Roepstorff and Fohlman, 1984] (see Figure 3.2). Among them, the most common fragments produced by CID are b- and y-ions, resulting from breakage directly at the peptide bond [Steen and Mann, 2004]. In addition to these major ion types, *neutral loss* variants, such as loss of water (e.g., b-H_2O) or ammonia (e.g., y-NH_3), can be produced [Steen and Mann, 2004]. Every ion type and neutral loss variant has a specific mass offset from the summed mass of the corresponding N-terminal or C-terminal residues, referred to as *prefix residue mass* for N-terminal fragments and *suffix residue mass* for C-

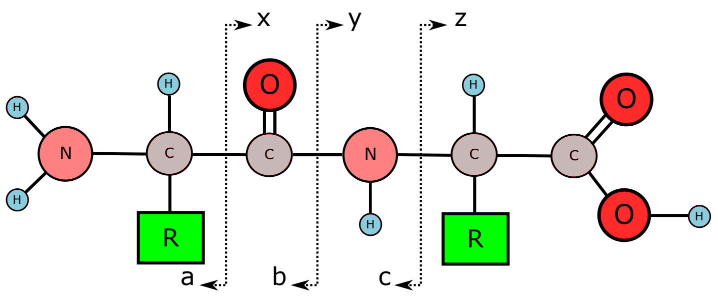


Figure 3.2: *The three main fragmentation sites along the peptide backbone in tandem mass spectrometry and the corresponding ion types. Green boxes represent amino acid specific side chains linked to the α-C-atom of the corresponding residue. The repeating sequence of* N − H, *α-C-atom, and* C = O *of each residue forms the peptide backbone. Adjacent residues are linked by peptide bonds.*

terminal fragments. The mass offsets for the most important ion types are listed in Table 3.2. Further, depending on the type of mass spectrometer and the charge of the precursor ion, also multiply charged fragment ions are frequently observed in tandem mass spectra. These charge variants are usually denoted by superscript notation, e.g., y^2 and b^2 for doubly charged y-ion and b-ion respectively.

According to the position in the peptide of length n, the product ions are numbered from one to $n-1$, such that ion b_i defines the b-ion consisting of the first i amino acids, and y_j defines the y-ion consisting of the last j amino acids.

For the identification of peptides from tandem mass spectra, there exist two different approaches, database assisted identification and the *de novo* peptide sequencing. When a database of known proteins for the analyzed sample is at hand, the method of choice is usually the database assisted identification where the experimental tandem mass spectra are matched against candidate peptides in the database. Given an experimental spectrum and a candidate peptide, database search algorithms like INSPECT [Tanner et al., 2005], SEQUEST [Eng et al., 1994], Mascot [Perkins et al., 1999], and OMSSA [Geer et al., 2004] compute a matching score that reflects the likelihood that the spectrum has been generated by the candidate peptide. Finally, they return a ranked list of highest scoring candidate peptides. Since computing such a score for every peptide in the database is very time consuming, a crucial step in database search algorithms is to filter the database from unlikely candidates. One filtering approach, implemented by INSPECT, is to generate short peptide sequence tags, i.e., short amino acid sequences that are supported by the peaks in the experimental spectrum, and to

Table 3.2: *Ion types*

List of most important N- and C-terminal ion types created in MS/MS peptide fragmentation and their offsets (rounded to integer values) from the prefix residue mass (m_N) for N-terminal fragment ions or suffix residue mass (m_C) for C-terminal fragment ions.

N-terminal		C-terminal	
ion type	offset (Da)	ion type	offset (Da)
b	$m_N + 1$	y	$m_C + 19$
b-H_2O	$m_N - 17$	y-H_2O	$m_C + 1$
b-NH_3	$m_N - 16$	y-NH_3	$m_C + 2$
b-H_2O-H_2O	$m_N - 35$	y-H_2O-H_2O	$m_C - 17$
b-H_2O-NH_3	$m_N - 34$	y-H_2O-NH_3	$m_C - 16$
b^2	$(m_N + 2)/2$	y^2	$(m_C + 20)/2$
a	$m_N - 27$	x	$m_C + 45$
a-H_2O	$m_N - 45$	z	$m_C + 3$
a-NH_3	$m_N - 44$		
c	$m_N + 18$		

perform sequence based filtering of the database, keeping only those candidates that are compatible with at least one tag. Another filtering approach, as implemented by SEQUEST, is to consider only candidate peptides whose mass is within a certain tolerance to the measured precursor ion mass of the experimental spectrum, called *parent mass* or *precursor mass*.

Obviously, the performance of every database search algorithm depends on the completeness and correctness of the underlying database. Therefore, these methods may fail to identify peptides when no complete protein database is available for the organism being studied. Even for well studied organisms, these methods can suffer from previously unknown alternative splice variants or mutations that may prevent successful identification.

de novo peptide sequencing

In contrast to database assisted identification methods, *de novo* peptide sequencing algorithms like PEAKS [Ma et al., 2003], PepNovo [Frank and Pevzner, 2005], NovoHMM [Fischer et al., 2005], Lutefisk [Taylor and Johnson, 1997], Sherenga [Dančík et al., 1999], PNovo+ [Chi et al., 2013], EigenMS [Bern and Goldberg, 2006], and PILOT [DiMaggio and Floudas, 2007] aim to infer the peptide sequence solely from the spectrum, without any additional information stored in protein databases. The key to *de novo* sequencing are the so-called *ion ladders*, which correspond to a sequence of consecutive ions of the same type, e.g., $b_i, b_{i+1}, \ldots, b_k$. An important observation is that whenever all peaks corresponding a complete ion ladder can be identified, then these peaks provide all information required to reconstruct the peptide sequence. Assuming that all peaks corresponding to the complete ladder of singly charged b-ions $b_1, b_2, \ldots, b_{n-1}$ are known, then the m/z difference between the peaks of two consecutive b-ions, b_i and b_{i+1}, is exactly the residue mass of the amino acid located at position $i + 1$ in the peptide. The first and last amino acid can be inferred from the m/z of the b_1-ion and the m/z difference between the b_{n-1}-ion and the measured parent mass. Therefore, the central problem in *de novo* peptide sequencing is the identification of a hopefully complete ion ladder of a certain ion type. This problem is complicated by factors like *noise peaks* in the spectrum that do not correspond to fragments of the analyzed peptide or incomplete fragmentation, resulting in incomplete ladders. One approach to reduce the problem of incomplete fragmentation is the combination of tandem mass spectra of the same peptide generated by different, so-called complementary, fragmentation methods, as implemented by Datta and Bern [2009], Bertsch et al. [2009], and Chi et al. [2013]. However, these complicating factors still render *de novo* peptide sequencing a challenging task in mass spectrometry based proteomics, demanding efficient algorithmic solutions.

New algorithm

Our novel algorithmic approach to *de novo* peptide sequencing is based on an efficient Lagrangian relaxation algorithm to solve the underlying *longest antisymmetric path* problem, without any restrictions on the set of considered ion types that are imposed by existing methods. At the same time, even for this restricted case, our algorithm is at least competitive to existing algorithms based on dynamic programming [Lu and Chen, 2003] or *hidden Markov models* [Fischer et al., 2005], and it is orders of magnitude faster compared to solving the underlying ILP formulation by means of state-of-the-art solvers. Further, the flexibility of our formulation offers several opportunities for future adaptions to integrate additional information into the sequencing process and to account for mass spectrometer type-specific properties of the generated spectra. This is further supported by our generic probabilistic scoring model, based on a Bayesian network, that captures mass spectrometer type-specific dependencies between the signals generated for particular ion types.

3.2 A Lagrangian relaxation algorithm

In the following, we provide a detailed description of our Lagrangian relaxation approach to the *de novo* peptide sequencing problem. We first introduce the graph-theoretical formulation before we derive an equivalent ILP formulation that underlies our algorithm.

3.2.1 Graph-theoretical formulation

A common approach to the *de novo* peptide sequencing problem is to formulate it as a graph-theoretical problem by transforming the spectrum into the so-called *spectrum graph*, first introduced by Bartels [1990]. After this transformation, the identification of candidate peptide sequences that are supported by the peaks in the spectrum amounts to the identification of paths through the spectrum graph, with some additional restrictions.

Before we define the spectrum graph and the associated graph-theoretical formulation of the *de novo* peptide sequencing problem, we begin with some basic definitions. Given a peptide $P = \langle p_1, p_2, \ldots, p_n \rangle$, we define the *residual mass* $m_r(P)$ as the sum of the monoisotopic residual masses (see Table 3.1) of all amino acid residues in P (i.e., $m_r(P) := \sum_{i=1}^{n} m_r(p_i)$). Analogously, for every prefix (resp. suffix) of P containing k residues, the *prefix residue mass* (resp. *suffix residue mass*) is defined as $m_r(\, \langle p_1, p_2, \ldots, p_k \rangle \,)$ (resp. $m_r(\, \langle p_{n-k+1}, p_{n-k+2}, \ldots, p_n \rangle \,)$). By the *parent mass* $m_p(P)$ we denote the total mass of P, which is the residual mass $m_r(P)$ plus $\sim$18 Da for an additional water molecule.

Spectrum graph

The spectrum graph $\mathcal{G} = (V, E_D, E_U)$ consists of a set of vertices V, a set of directed edges E_D and a set of undirected edges E_U. Note that this definition corresponds to the so-called *extended spectrum graph*, proposed by Liu et al. [2006], while the original definition of the spectrum graph does not include the set of undirected edges. Every vertex v_i represents a candidate prefix residue mass $m(v_i)$ of the unknown peptide sequence, and every directed edge represents a single amino acid or a combination of amino acids. Undirected edges connect pairs of vertices that correspond to contradicting interpretations of the same mass peak. In the remainder, we will refer to these pairs of contradicting vertices as *incompatible* vertices.

For a given tandem mass spectrum S with measured parent mass M_p, the associated spectrum graph is constructed as follows: Given is a set $\mathcal{T}$ of k ion types consisting of k_N N-terminal ion types $\tau_1^N, \ldots, \tau_{k_N}^N$ with mass offsets

$\delta_1^N, \ldots, \delta_{k_N}^N$ from the prefix residue mass and k_C C-terminal ion types $\tau_1^C, \ldots, \tau_{k_C}^C$ with mass offsets $\delta_1^C, \ldots, \delta_{k_C}^C$ from the suffix residue mass. In the following and without loss of generality, we assume all considered ion types in $\mathcal{T}$ to have unit charge. For every peak π in the spectrum with m/z value m_π, we generate a set of k vertices $\mathcal{I}_\pi$, where each vertex represents a prefix residue mass under the assumption that π was produced by an ion of one of the k considered types. These vertices have masses $m_\pi - \delta_1^N, \ldots, m_\pi - \delta_{k_N}^N$ for N-terminal types and $M_p - 18 - (m_\pi - \delta_1^C), \ldots, M_p - 18 - (m_\pi - \delta_{k_C}^C)$ for C-terminal ion types respectively. Once the vertices have been generated for all peaks in S, pairs of vertices having a mass difference below a certain mass tolerance are merged into a single vertex.

Since we assume that peak π was produced by a single ion that corresponds to exactly one type in $\mathcal{T}$ or some unknown ion, at most one of the generated vertices in $\mathcal{I}_\pi$ can present a true prefix residue mass of the unknown peptide. Therefore, all vertices in $\mathcal{I}_\pi$ are contradicting each other, and each pair is connected by an undirected edge.

Whenever the mass difference of two vertices v_i and v_j $(m(v_j) - m(v_i))$ is within a certain tolerance to the mass of some amino acid α, we connect v_i and v_j by a directed edge (v_i, v_j), labeled with α. Equivalent to vertices, we say that every directed edge $e = (v_i, v_j)$ has a certain edge mass $m(e)$ and define it as the mass difference of the head and tail vertex, i.e., $m(e) := m(v_j) - m(v_i)$. Finally, we add two so-called *goalpost* vertices s and t with masses $0\,\mathrm{Da}$ and $M_p - 18\,\mathrm{Da}$ respectively.

According to the construction of the spectrum graph, there exists a vertex with the correct prefix residue mass for every prefix of the unknown peptide P, if for every pair of complementary prefix and suffix fragments, at least one ion of the considered k types produced a peak in the spectrum. In this case, we can reconstruct the correct sequence of P by finding the directed s-t-path of vertices corresponding to the true prefix residue masses of P and concatenating the edge labels along this path (see Figure 3.3). In the remainder, a path in the spectrum graph always refers to a directed path using directed edges in E_D exclusively.

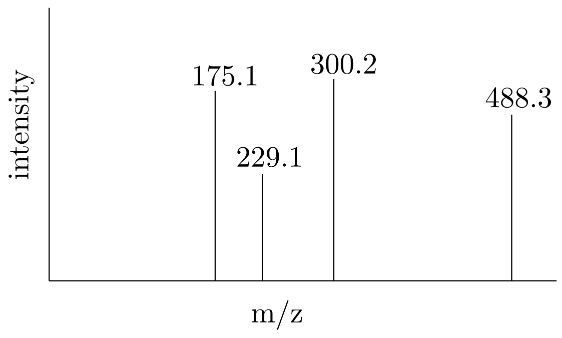
Each vertex v in the spectrum graph is assigned a score $\mathrm{score}(v)$ that represents the likelihood of this vertex to correspond to a true prefix residue mass of the unknown peptide. In addition, also directed edges can have individual scores that reflect the likelihood that some edge corresponds to a true subsequence of the unknown peptide. These edge scores can be used to penalize deviation of the edge mass from the exact mass of the amino acid label or introduce prior knowledge about the likelihood of observing certain amino acids in the unknown peptide.

Finally, we transform the vertex and edge scores into edge weights for directed edges. Since the artificial vertices s and t have no associated scores, we assign them a dummy score of zero. Now, for every vertex $v \in V$, we add its score to the weight of all outgoing edges. Hence, the edge weights $w : E_D \to \mathbb{R}$ are given by

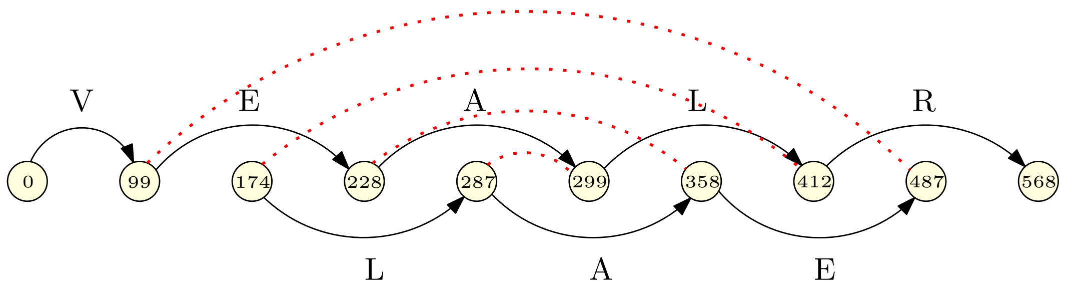
$$w(e) = score(v_i) + score(e) \qquad \forall e = (v_i, v_j) : e \in E_D \,.$$

Therefore, the *de novo* peptide sequencing problem can be reduced to computing longest s-t-paths in the spectrum graph.

However, searching a longest s-t-path in the spectrum graph can lead to infeasible solutions if pairs of incompatible vertices are included in the path, since,



(a)



(b)

Figure 3.3: *(a) Simplified tandem mass spectrum of the peptide* `VEALR` *with peaks for ions b_2, b_3, y_1, and y_4. Corresponding m/z values in* Th *are given on top of each peak. (b) The corresponding spectrum graph with undirected edges drawn as dashed lines and two generated vertices for each peak with associated rounded mass. The two vertices generated for each peak correspond to the interpretations as b-ion and y-ion. The (only) path starting at vertex s with mass 0 and ending at vertex t with mass 568 encodes the correct peptide sequence.*

as we pointed out above, in general at most one of them will correspond to a true prefix residue mass of the unknown peptide. Exceptions are peptides with fragment ions of different ion types in the set $\mathcal{T}$ having the same m/z value, as in this case more than one interpretation of the corresponding peak is correct. The associated s-t-path in the spectrum graph is therefore infeasible, and the correct peptide cannot be generated. As this situation occurs rarely in practice it is excluded by most algorithms that employ the spectrum graph.

The problem of incompatible vertices is aggravated when the scoring function generates several high scoring, incompatible vertices for the same peak, as a simple longest path is then likely to contain pairs of incompatible vertices.

Antisymmetric paths

In the context of *de novo* peptide sequencing, a path without any pair of incompatible vertices is usually called an *antisymmetric* path, as introduced by Dančík et al. [1999]. However, this term is mostly used for the case when only one pair of N-terminal and C-terminal ion types is considered, usually only b- and y-ions for CID spectra, since in this case incompatible pairs have a non-interleaving and hierarchical structure, as shown in Figure 3.3(b). Dančík et al. refer to graphs with this structure as *proper* graphs. For unrestricted structures of incompatible vertex pairs, this problem is also known as the *paths avoiding forbidden pairs* problem and has been shown to be NP-complete [Gabow et al., 1976; Kolman and Pangrác, 2009]. However, for proper graphs the problem of computing a longest antisymmetric path can be solved with a polynomial time dynamic programming algorithm proposed by Chen et al. [2001]. Later, Lu and Chen [2003] extended this approach to compute suboptimal solutions by constructing a so-called *matrix spectrum graph* and applying depth-first search and a backtracking algorithm. While PepNovo and SHERENGA both use dynamic programming algorithms similar to the one proposed by Chen et al., NovoHMM implements a *factorial* hidden Markov model (HMM) to overcome the problem of contradicting interpretations of peaks, and the approach by Liu et al. is based on *tree decomposition*. Bafna and Edwards [2003] proposed a variant of the dynamic programming approach that also allows to consider more than only a single pair of N- and C-terminal ion types. This algorithm is limited to so-called *simple ion types*, excluding doubly and higher charged ions, which could also support the sequencing process. In Section 3.2.5, we will show that the matrix spectrum graph approach by Lu and Chen also works for more than a single pair of N- and C-terminal ion types if these types fulfill some properties, similar to simple ion types.

In contrast to all approaches discussed above, the ILP formulation presented in the next section does not restrict the sets of considered ion types, as it does not depend on any particular structure of incompatible vertex pairs. Therefore, unlike any previous *de novo* sequencing algorithm, this formulation is also capable of solving the longest antisymmetric path problem even when incompatible vertices correspond to ion type interpretations with different charge states.

3.2.2 Integer linear programming formulation

We model the longest antisymmetric path problem for *de novo* peptide sequencing by the following ILP formulation, which resembles the formulation by DiMaggio and Floudas [2007], implemented in their *de novo* peptide sequencing tool PILOT. For every directed edge $(v_i, v_j) \in E_D$, we introduce a binary variable $x_{i,j}$, which has value one if edge (v_i, v_j) is part of the path (active) and zero otherwise (inactive). As the solution of the ILP is supposed to correspond to a longest antisymmetric path, the objective function (3.1) is to maximize the summed score of all active directed edges, i.e.,

$$\max \sum_{(v_i,v_j)\in E_D} w_{i,j} x_{i,j} \,, \tag{3.1}$$

where $w_{i,j} := w(e)$ for $e = (v_i, v_j)$. For the two goalposts s and t, we ensure that exactly one outgoing edge of s and one incoming edge of t are active by adding the two constraints

$$\sum_{(v_s,v_i)\in E_D} x_{s,i} = 1 \,, \tag{3.2}$$

and

$$\sum_{(v_i,v_t)\in E_D} x_{i,t} = 1 \,. \tag{3.3}$$

Every other vertex v is either contained in the path, having exactly one active incoming and one active outgoing edge, or it is not contained in the path and hence, none of its incoming or outgoing edges are active. This property is captured by the following flow conservation constraint, which ensures an equal number of active incoming and outgoing edges:

$$\sum_{(v_i,v_k)\in E_D} x_{i,k} - \sum_{(v_k,v_j)\in E_D} x_{k,j} = 0 \qquad \forall v_k \in V \setminus \{v_s, v_t\} \,. \tag{3.4}$$

Finally, we must ensure that no pair of incompatible vertices is contained in the solution path, which we achieve by adding the following constraint:

$$\sum_{v_i \in e} \sum_{(v_i,v_k)\in E_D} x_{i,k} \le 1 \qquad \forall e \in E_U \,. \tag{3.5}$$

The objective function (3.1) and constraints (3.2)-(3.5) correspond to the ILP formulation we proposed in [Andreotti et al., 2012]. In the remainder, we will use a slight modification of this model by generalizing constraint (3.5) from single pairs of incompatible vertices to maximal sets of pairwise incompatible vertices. As described for the construction of the spectrum graph, for every peak π in the input spectrum, we generate a set of pairwise incompatible vertices $\mathcal{I}_\pi$. By defining $\mathcal{I}_S$ to be the set of all sets $\mathcal{I}_\pi$ for all peaks π in spectrum S, i.e., $\mathcal{I}_S = \{\mathcal{I}_\pi \mid \pi \in S\}$, we can replace constraint (3.5) by

$$\sum_{v_i \in \mathcal{I}_\pi} \sum_{(v_i,v_k) \in E_D} x_{i,k} \leq 1 \qquad \forall\, \mathcal{I}_\pi \in \mathcal{I}_S\,, \tag{3.6}$$

which yields a stronger formulation and reduces the number of constraints if more than two vertices are generated for the same peak. This leads to the complete ILP formulation for the longest antisymmetric path problem expressed in (3.7).

$$\begin{aligned}
\max \sum_{(v_i,v_k)\in E_D} w_{i,k} x_{i,k} & \\
\text{subject to} \quad \sum_{(v_s,v_k)\in E_D} x_{s,k} &= 1 \\
\sum_{(v_k,v_t)\in E_D} x_{k,t} &= 1 \\
\sum_{(v_i,v_k)\in E_D} x_{i,k} - \sum_{(v_k,v_j)\in E_D} x_{k,j} &= 0 \qquad \forall v_k \in V \setminus \{v_s, v_t\} \\
\sum_{v_i \in \mathcal{I}_\pi} \sum_{(v_i,v_k)\in E_D} x_{i,k} &\leq 1 \qquad \forall\, \mathcal{I}_\pi \in \mathcal{I}_S \\
x_{i,k} \in \{0,1\} & \quad \forall (v_i, v_k) \in E_D
\end{aligned} \tag{3.7}$$

This ILP formulation differs from the formulation proposed by DiMaggio and Floudas [2007] in two aspects. First, our formulation employs only variables for directed edges, while DiMaggio and Floudas use additional variables for the vertices of the spectrum graph. This is only a minor technical detail that does not change the general structure or efficiency of the formulation. Second, DiMaggio and Floudas add an additional constraint that bounds the deviation of the exact mass of the predicted peptide from the measured parent mass by a certain value (usually 2 Da). This deviation is calculated as the difference between $m(e)$ and the exact mass of the amino acid label of e, summed over all active edges $e \in E_D$. Hence, this constraint filters out candidate peptide sequences with high mass deviation from the measured parent mass. We do not include this constraint into our model, as we believe it is more promising to defer this filtering to a later stage of the complete sequencing algorithm. Like other *de novo* sequencing algorithms, we do not only add edges to the spectrum graph that correspond to single amino acids but also edges corresponding to pairs and triples of amino acids, to account for incomplete fragmentation. These edges often do not have a unique label, but instead they represent multiple possible combinations of amino acids with slight mass differences within the allowed mass tolerance. As a consequence, no exact mass is known a priori for these edges that could be used in such a constraint. Hence, we prefer to perform the filtering at a later stage of the algorithm after every predicted path has been transformed into a candidate set of peptide sequences containing all possible combinations and permutations of ambiguous edge labels.

3.2.3 Applying Lagrangian relaxation

In our approach to the longest antisymmetric path problem, we do not solve the ILP formulation (3.7) directly by a generic ILP solver, but instead we apply Lagrangian relaxation. Obviously, the longest antisymmetric path problem corresponds to a longest path problem in a directed acyclic graph subject to the additional constraint (3.6) that prevents the simultaneous selection of incompatible vertices. A longest path in a directed acyclic graph with n vertices and m edges can be computed in time $\mathcal{O}(n+m)$ by the simple DAG-LONGEST-PATH algorithm (adaption of DAG-SHORTEST-PATHS presented in [Cormen et al., 2001]), outlined in Algorithm 1 (see page 64).

As a consequence, an obvious choice for Lagrangian relaxation is to dualize constraints (3.6) and add their violation as penalty term to the objective function, as explained in Section 2.2.2. This leads to the following Lagrangian relaxation formulation of ILP (3.7) with Lagrange multipliers $\boldsymbol{\lambda} \in \mathbb{R}_+^{|\mathcal{I}_S|}$.

$$\begin{aligned}
Z_{\text{path}}(\boldsymbol{\lambda}) = \max \sum_{(v_i,v_k)\in E_D} w_{i,k}x_{i,k} + \sum_{\mathcal{I}_\pi \in \mathcal{I}_S} \lambda_\pi \Big(1 - \sum_{\substack{(v_i,v_k)\in E_D:\\ v_i \in \mathcal{I}_\pi}} x_{i,k}\Big) & \\
\text{subject to} \quad \sum_{(v_s,v_k)\in E_D} x_{s,k} = 1 & \\
\sum_{(v_k,v_t)\in E_D} x_{k,t} = 1 & \qquad (3.8)\\
\sum_{(v_i,v_k)\in E_D} x_{i,k} - \sum_{(v_k,v_j)\in E_D} x_{k,j} = 0 \qquad \forall v_k \in V \setminus \{v_s, v_t\} & \\
x_{i,k} \in \{0,1\} \quad \forall (v_i,v_k) \in E_D &
\end{aligned}$$

When we rewrite the objective function as

$$Z_{\text{path}}(\boldsymbol{\lambda}) = \max \sum_{(v_i,v_k)\in E_D} x_{i,k}(w_{i,k} - \sum_{\substack{\mathcal{I}_\pi \in \mathcal{I}_S: \\ v_i \in \mathcal{I}_\pi}} \lambda_\pi) + \sum_{\mathcal{I}_\pi \in \mathcal{I}_S} \lambda_\pi \,, \tag{3.9}$$

it becomes obvious that, for every fixed Lagrange multiplier $\boldsymbol{\lambda}$, we can solve the relaxed problem by computing the longest path in the spectrum graph with adjusted edge weights w_λ, defined as

$$w_\lambda(i,j) := w(i,j) - \sum_{\substack{\mathcal{I}_\pi \in \mathcal{I}_S: \\ v_i \in \mathcal{I}_\pi}} \lambda_\pi \qquad \forall (v_i, v_j) \in E \,. \tag{3.10}$$

Thus, it follows that we can solve the relaxed problem (3.8) in linear time and space, as stated in the following theorem.

Theorem 3.1. *The Lagrangian relaxed problem* (3.8) *can be solved in linear time and space.*

Proof. Solving the Lagrangian relaxed problem for a fixed Lagrange multiplier $\boldsymbol{\lambda}$ consists of the following steps:

1. Compute modified edge weights w_λ.
2. Apply DAG-LONGEST-PATH algorithm to compute the longest s-t-path in the spectrum graph with modified edge weights w_λ.
3. Add value $\sum_{\mathcal{I}_\pi \in \mathcal{I}_S} \lambda_\pi$ to the length of the longest s-t-path.

Each of the steps requires only $\mathcal{O}(|E_D| + |V|)$ time and space. □

3.2.4 Solving the longest antisymmetric path problem by Lagrangian relaxation and branch and bound

To solve the longest antisymmetric path problem optimally, we embed our Lagrangian relaxation formulation into a branch and bound algorithm, as introduced in Section 2.2.1. In the remainder, we will refer to this algorithm as LAG-ANTISYMMETRIC-PATH. For every subproblem in the branch and bound tree, beginning with the original problem, we solve the Lagrangian dual of our Lagrangian relaxation using the subgradient optimization method outlined in Section 2.2.2. In every iteration of the subgradient optimization, we compute a longest s-t path in the spectrum graph using the adapted edge weights w_λ. If this path is primal feasible, i.e., it does not contain any pair of incompatible vertices, we can compute its primal score as the length of the path for the original edge weights w. This value serves as a lower bound for the length of the longest antisymmetric path, while the value $Z_{\text{path}}(\boldsymbol{\lambda})$ provides an upper bound. Thus, a primal feasible path with a length equal to the lowest value of $Z_{\text{path}}(\boldsymbol{\lambda})$ found thus far is a primal optimal solution and therefore a longest antisymmetric path. When no primal optimal solution is found after a predefined number of subgradient iterations, we branch by creating two subproblems as follows: Let p^* be the lowest scoring primal infeasible solution found during subgradient optimization and let C_{p^*} be a set of pairwise incompatible vertices in p^*. We randomly select one vertex $v_b \in C_{p^*}$ and solve the Lagrangian relaxations of the two subproblems sub_1 and sub_2, which we obtain by:

- sub_1: adding constraint
$$\sum_{(v_b,v_k)\in E_D} x_{b,k} = 0\,,$$

- sub_2: adding constraints
$$\sum_{(v_b,v_k)\in E_D} x_{b,k} = 1 \quad \text{and} \quad x_{i,j} = 0 \quad \forall (v_i, v_j) \in E_D : \{v_b, v_i\} \in E_U\,.$$

Hence, every solution to sub_1 will not include vertex v_b, whereas every solution to sub_2 includes vertex v_b but no vertex that is incompatible to v_b. To implement these additional constraints into our algorithm for solving the Lagrangian relaxation, we modify the spectrum graph by:

- sub_1: removing all edges $(v_b, v_k) \in E_D$,
- sub_2: removing all edges $(v_i, v_k) \in E_D$ with $v_i < v_b < v_k$ in topological order and removing all edges $(v_i, v_j) \in E_D : \{v_b, v_i\} \in E_U$.

3.2.5 Polynomial algorithm for longest antisymmetric paths with restricted sets of ion types

In this section, we introduce the data structure underlying the polynomial time algorithm for enumerating optimal and suboptimal antisymmetric paths on proper spectrum graphs, proposed by Lu and Chen [2003]. Further, we will show that this algorithm also solves the longest antisymmetric path problem optimally for non-proper graphs if the set of considered ion types fulfills certain properties. In the remainder, we will call every set of ion types with these properties a set of *basic ion types*. Note that our definition of basic ion types is a slight modification of the *simple ion types* introduced by Bafna and Edwards [2003].

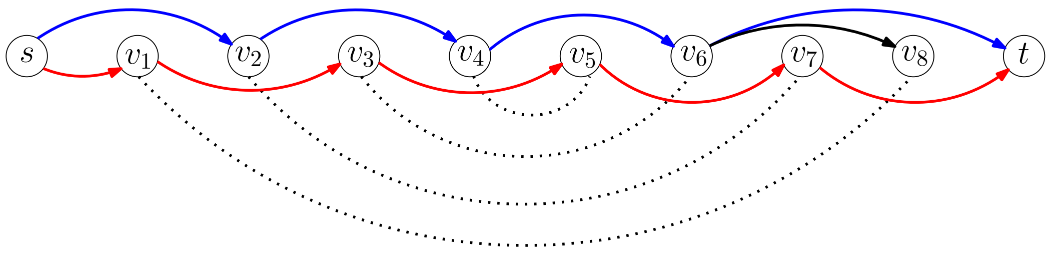
Definition 3.1. Basic Ion types
Let m_α denote the minimal mass of all considered amino acids and M_p the measured parent mass of a given mass spectrum. We call a set of ion types $\mathcal{T}$ basic *if:*

1. *For all peaks $\pi \in S$ and all $v_i, v_j \in \mathcal{I}_\pi$ with $m(v_i), m(v_j) \leq (M_p - 18)/2$:*
$$| m(v_i) - m(v_j) | < m_\alpha \, .$$
2. *For all peaks $\pi \in S$ and all $v_i, v_j \in \mathcal{I}_\pi$ with $m(v_i), m(v_j) > (M_p - 18)/2$:*
$$| m(v_i) - m(v_j) | < m_\alpha \, .$$
3. *For all peaks $\pi \in S$ and all $v_i, v_j \in \mathcal{I}_\pi$ with $m(v_i) \leq (M_p - 18)/2$ and $m(v_j) > (M_p - 18)/2$:*
$$| m(v_i) + m(v_j) - (M_p - 18) | < m_\alpha \, .$$

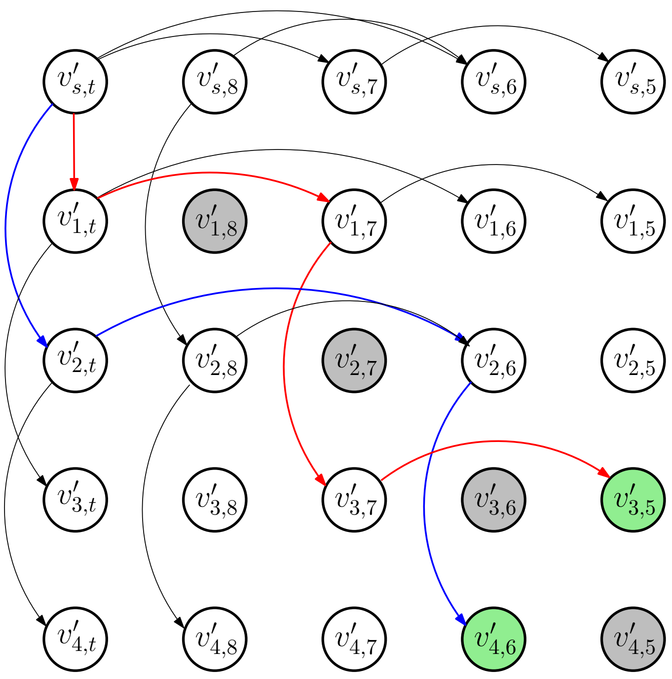
Intuitively, these restrictions ensure that for every pair of incompatible vertices either their mass difference is less than the minimal amino acid mass or the difference between the sum of their masses and the peptide residue mass is less than the minimal amino acid mass. Next, we will show how these restrictions allow for a polynomial time algorithm to compute longest antisymmetric paths by means of the matrix spectrum graph. In Section 3.5.1, we will compare the practical performance of our Lagrangian relaxation approach to the polynomial algorithm on sets of basic ion types.

Matrix spectrum graph

The algorithm proposed by Lu and Chen is based on the transformation of the spectrum graph $\mathcal{G} = (V, E_D, E_U)$ into the associated *matrix spectrum graph* $\mathcal{G}' = (V', E')$ as follows: For a given spectrum graph, split the set of vertices $V = \{s, v_1, \ldots, v_n, t\}$ into two disjoint subsets $V^x = \{s, v_1, \ldots, v_m\}$ and $V^y = \{v_{m+1}, \ldots, v_n, t\}$. All vertices in subset V^x have mass at most $(M_p - 18)/2$, while the mass of all vertices in V^y is strictly greater than $(M_p - 18)/2$. For every pair of vertices $v_i \in V^x$ and $v_j \in V^y$ that are not incompatible (i.e., $\{v_i, v_j\} \notin E_U$), the matrix spectrum graph contains a vertex $v'_{i,j}$. Further, the matrix spectrum graph contains a *horizontal* edge $(v'_{i,j}, v'_{i,k})$ if $(v_k, v_j) \in E_D$ and $m(v_k) + m(v_i) \leq (M_p - 18)$ and a *vertical* edge $(v'_{i,j}, v'_{k,j})$ if $(v_i, v_k) \in E_D$ and $m(v_k) + m(v_j) > (M_p - 18)$. Finally, the matrix spectrum graph defines a single source vertex $v'_{s,t}$ (corresponding to $v_{1,n}$) and a set of *terminal* vertices V'_{term},



(a)



(b)

Figure 3.4: *Spectrum graph (a) and associated matrix spectrum graph (b). Grey shaded vertices in matrix spectrum graph are only drawn for reasons of readability. As these vertices represent pairs of incompatible vertices in the spectrum graph, they are not part of the matrix spectrum graph. The two green vertices in the matrix spectrum graph are terminal vertices. Corresponding paths in spectrum graph and matrix spectrum graph are highlighted in blue and red.*

consisting of all vertices $v'_{i,j}$ with $(v_i, v_j) \in E_D$. An example of a spectrum graph with associated matrix spectrum graph is depicted in Figure 3.4.

Intuitively, using the matrix spectrum graph $\mathcal{G}'$, we compute s-t paths in $\mathcal{G}$ not in s-t direction, but instead we start at the two goalposts and construct the path from these endpoints towards the center of the spectrum graph. At the center, the two subpaths in the left and right half of $\mathcal{G}$, referred to as left subpath and right subpath, are joined by a terminal vertex that connects the last vertex in the left subpath with the first vertex in the right subpath. More formally, every path $\langle v'_{s,t}, v'_{p_1,q_1}, \ldots, v'_{p_k,q_l} \rangle$ in $\mathcal{G}'$ corresponds to a left subpath $p^x = \langle s, v_{p_1}, \ldots, v_{p_k} \rangle$ and a right subpath $p^y = \langle v_{q_l}, \ldots, v_{q_1}, t \rangle$ in $\mathcal{G}$. Since $\mathcal{G}'$ does not contain a vertex $v'_{i,j}$ for any pair of incompatible vertices in $\mathcal{G}$ and the set of ion types is basic, we can show that the union of both paths p^x and p^y does not contain any pair of incompatible vertices[1]. Thus, every path in $\mathcal{G}'$ beginning at $v'_{s,t}$ and ending at a terminal vertex corresponds to an antisymmetric s-t path in $\mathcal{G}$.

Theorem 3.2. *For a set of basic ion types, every path* $p = \langle v'_{s,t}, v'_{p_1,q_1}, \ldots, v'_{p_k,q_l} \rangle$ *in* $\mathcal{G}'$ *with* $v'_{p_k,q_l} \in V'_{term}$ *corresponds to an antisymmetric* s-t *path in* $\mathcal{G}$.

Proof. First, we show that p corresponds to some s-t path in $\mathcal{G}$. According to the construction of $\mathcal{G}'$, the following holds: For every pair of successive vertices $v_{p_i}, v_{p_{i+1}}$ in the sequence $\langle v_s, v_{p_1}, \ldots, v_{p_k} \rangle$, either $v_{p_i} = v_{p_{i+1}}$ or $(v_{p_i}, v_{p_{i+1}}) \in E_D$. Hence, after removing successive identical vertices, the resulting sequence corresponds to a path in $\mathcal{G}$ from vertex s to vertex v_{p_k}. By the same argument, the sequence $v_{q_l}, \ldots, v_{q_1}, t$ corresponds to a path in $\mathcal{G}$ from vertex v_{q_l} to vertex t. Since $v'_{p_k,q_l} \in V'_{term}$, there exists an edge $(v_{p_k}, v_{q_l}) \in E_D$, which joins these two paths into a single s-t path in $\mathcal{G}$.

Second, we show that the corresponding s-t path in $\mathcal{G}$ is antisymmetric. Every edge in E' represents an edge in E_D connecting either two vertices in V^x or two vertices in V^y. Since every edge corresponds to an amino acid with mass at least m_α and every pair of incompatible vertices both in V^x (resp. V^y) has a mass difference strictly less than m_α, no pair of incompatible vertices both in V^x (resp. V^y) can be selected for the path using edges in E'. Finally, we have to show that also no pair of incompatible vertices $v_a \in V^x$ and $v_b \in V^y$ can be selected. Since V' does not contain any vertex $v'_{a,b}$ for any pair of incompatible vertices $v_a, v_b \in V$, a path including such a pair must contain two vertices $v'_{a,k}$ and $v'_{l,b}$ with $a \neq l$ and $b \neq k$. Assume $k > b$ and $a < l$, that is, $v'_{a,k}$ appears before $v'_{l,b}$ in the path, and consider two cases. Note that the symmetric case with $k < b$ and $a > l$ can be shown in the same way.

Case 1: Assume the first edge in the subpath from $v'_{a,k}$ to $v'_{l,b}$ is a vertical edge connecting $v'_{a,k}$ to some vertex $v'_{r,k}$. Since $k > b$, the subpath must also contain at least one horizontal edge ending at some vertex $v'_{c,d}$ with $m(v_c) \geq m(v_r) \geq m(v_a) + m_\alpha$. From the construction rule of horizontal edges we

[1] While Lu and Chen [2003] briefly perform a case study with an additional ion type (b-H_2O), a formal statement about the extension to multiple ion types is not given.

know that

$$m(v_d) \leq M_p - 18 - m(v_c)\,,$$

which implies

$$m(v_d) \leq M_p - 18 - m(v_a) - m_\alpha\,.$$

As we consider only basic ion types, the following inequality holds:

$$m(v_a) + m(v_b) > M_p - 18 - m_\alpha\,,$$

which ultimately leads to

$$m(v_d) < m(v_b)\,.$$

Since the masses of vertices in V^y are monotonically decreasing along the path, there cannot exist a path from $v'_{c,d}$ to $v'_{l,b}$ in $\mathcal{G}'$.

Case 2: Now assume the first edge in the subpath from $v'_{a,k}$ to $v'_{l,b}$ is a horizontal edge connecting $v'_{a,k}$ to some vertex $v'_{a,r}$. If $m(v_r) < m(v_b)$, we can apply the same argument as above to show that no path from $v'_{a,r}$ to $v'_{a,b}$ exists. Hence, we assume $m(v_r) \geq m(v_b)$. Again, from the construction rule of horizontal edges it follows that:

$$m(v_a) + m(v_r) \leq M_p - 18\,.$$

Using again the basic ion type property, this can be transformed to

$$m(v_r) - m(v_b) < m_\alpha\,,$$

which implies that no path from $v'_{a,r}$ to $v'_{a,b}$ can exist, since every edge corresponds to a mass of at least m_α. □

Since $|V'| = \mathcal{O}(|V|^2)$ and $|E'| = \mathcal{O}(|V||E_D|)$, for any set of basic ion types, the longest antisymmetric path problem can be solved in time $\mathcal{O}(|V|^2 + |V||E_D|)$ using algorithm DAG-LONGEST-PATH on the matrix spectrum graph.

3.3 Enumeration of suboptimal antisymmetric paths

Most *de novo* peptide sequencing algorithms compute not only a single candidate peptide but a set of high scoring candidates. Similar to other algorithms such as PILOT, we follow a two step approach where we first generate a certain number of candidate peptide sequences by enumerating longest antisymmetric paths in the spectrum graph. All candidates generated in this first step are then again compared to the query spectrum in a second, re-scoring step.

This demands an algorithm to enumerate the k longest antisymmetric paths in the spectrum graph for a given number k. A straightforward strategy to enumerate suboptimal solutions for the ILP formulation, as implemented in PILOT, is to cut off every previous path $\langle v_{p_1}, \ldots, v_{p_n} \rangle$ by an additional constraint, e.g., $\sum_{i=1}^{n-1} x(v_{p_i}, v_{p_{i+1}}) \leq n-2$, and re-solve the problem.

In our Lagrangian relaxation approach we follow a different algorithmic approach based on an algorithm proposed by Yen [1971] for the enumeration of the k shortest simple paths (no repeated vertex) in a directed graph. Yen's algorithm is a deviation algorithm based on the fact that the i-th shortest path $p^i = \langle p_1^i, \ldots, p_{n_i}^i \rangle$ shares with every shorter path $p^l \in \{p^1, \ldots, p^{i-1}\}$ its first $\mathrm{lcp}(p^i, p^l)$ vertices, until the paths deviate (see Figure 3.5). We call the farthest of these common vertices of p^i, beginning at the source, the *deviation vertex* $d(p^i)$ of path p^i:

$$d(p^i) := p_m^i \quad \text{with} \quad m = \max_{l \in [1..i-1]} \mathrm{lcp}(p^i, p^l)\,.$$

Let $\mathcal{D}(p^i)$ denote the set of previously determined paths that coincide with p^i until vertex $d(p^i)$, i.e., $\mathcal{D}(p^i) = \{p^l \mid p^i_{\mathrm{lcp}(p^i,p^l)} = d(p^i) \wedge 1 \leq l < i\}$. Further, let the deviation vertex be the d-th vertex of path p^i, i.e., $p_d^i = d(p^i)$.

The idea of Yen's algorithm to determine the $(i+1)$-st shortest s-t path p^{i+1} is to compute the shortest path to t that deviates from p^i at vertex p_j^i, for every vertex p_j^i, $d \leq j \leq n_i$. To avoid repeated computation of the same path for vertex p_d^i, the new path must also deviate from all paths in $\mathcal{D}(p^i)$. A path deviating from p^i at vertex p_j^i is generated by computing a shortest p_j^i-t path that does not include edge (p_j^i, p_{j+1}^i). This so-called *spur path* p^{spur} from p_j^i to t is then concatenated with the so-called *root path* $p^{\mathrm{root}} = \langle p_1^i, \ldots, p_j^i \rangle$, and the resulting *$s$-$t$ deviation path* $p^{\mathrm{root}} \cdot p^{\mathrm{spur}}$ is added to a candidate set X. Note that with a slight abuse of notation, we let the concatenation of the root path and the spur path contain only the prefix $\langle p_1^i, \ldots, p_{j-1}^i \rangle$ of p^{root}, as otherwise the deviation vertex p_j^i would appear twice in the deviation path. Once all deviation paths for *parent path* p^i have been computed, the shortest path in the candidate set X corresponds to the $(i+1)$-st shortest s-t path p^{i+1} and is subsequently removed from X. Since the original algorithm is supposed to compute only simple paths, it performs an additional trick. During every shortest path computation this algorithm applies Dijkstra's algorithm, which is guaranteed to compute a simple path. However, additional care has to be taken to ensure that $p^{\mathrm{root}} \cdot p^{\mathrm{spur}}$ still defines a simple path. Yen's algorithm achieves this by removing all vertices $p_1^i, \ldots, p_{j-1}^i$ from the graph before computing the spur path. Hence, since both subpaths are simple and do not share any vertices, also their concatenation defines a simple s-t path.

The longest antisymmetric path problem differs from the problem solved by Yen's original algorithm in a few points, so we need to adapt the algorithm as

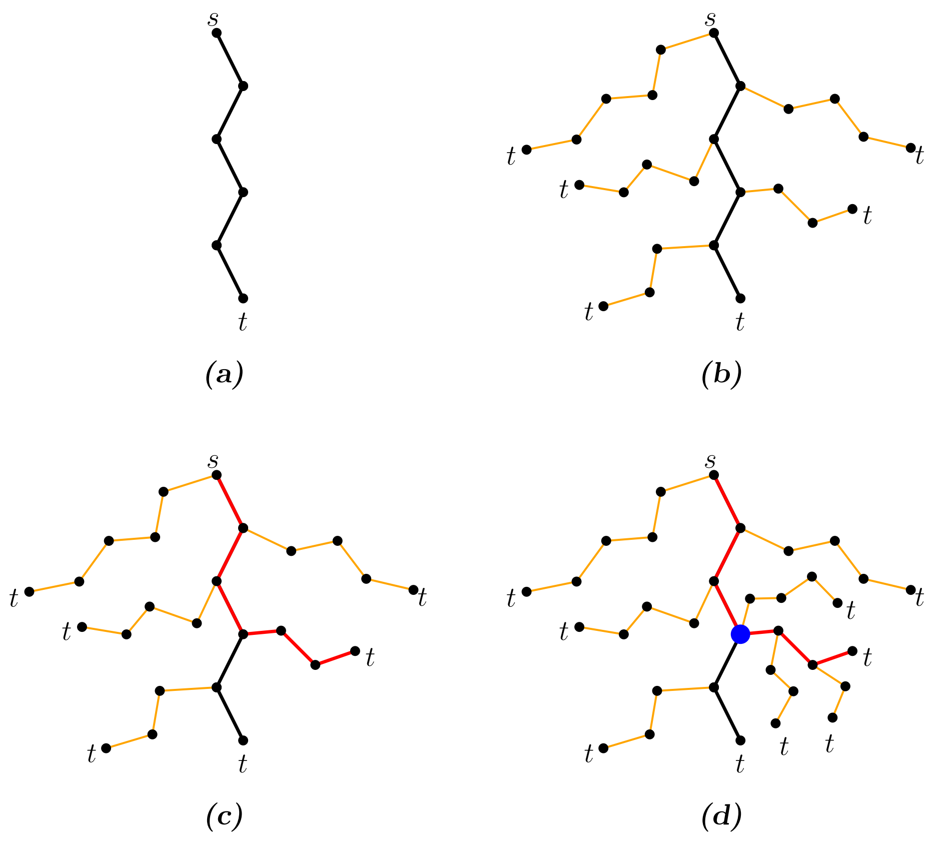


Figure 3.5: *Example workflow of a suboptimal path enumeration algorithm based on deviation paths. Beginning with the optimal s-t path p^1 (a), for each vertex $v \in p^1$, the spur path to t is computed. All spur paths, shown in orange (b), are concatenated with their associated root path and added to the set of candidates. The longest candidate path is marked red (c). Beginning at the deviation vertex marked in blue (d), the new spur paths to t are computed, and the resulting deviation paths are added to the candidate set.*

follows: While the original algorithm is designed for general directed graphs that may contain cycles, the spectrum graph is a directed acyclic graph. Thus, we can safely replace every application of Dijkstra's algorithm by algorithm Dag-Longest-Path to enumerate longest s-t paths. Further, while the original algorithm must ensure that the concatenated paths are still simple, for the antisymmetric path problem we must ensure that concatenated paths do not contain any pair of incompatible vertices. By means of algorithm Lag-Antisymmetric-Path, we can compute p^i_j-t spur paths without pairs of incompatible vertices. To ensure that the concatenation of the root path $\langle p^i_1, \ldots, p^i_j \rangle$ and the spur path also satisfies this condition, we adopt Yen's idea and remove all vertices that are incompatible to any vertex in $\langle p_1^i, \ldots, p_j^i \rangle$ from the spectrum graph before computing the spur path.

The properties of our algorithmic approach for the enumeration of suboptimal antisymmetric paths, referred to as k-Longest-Antisymmetric-Paths (see Algorithm 2, page 65), are captured in the following theorem:

Theorem 3.3. *The combination of algorithm* Lag-Antisymmetric-Path *and the modified version of Yen's algorithm, as described above, solves the problem of enumerating the k longest antisymmetric paths in a spectrum graph $\mathcal{G} = (V, E_D, E_U)$ in time $O(k\ell\beta(|E_D| + |V|))$, where ℓ is the maximum number of vertices in any generated candidate path and β is the maximum number of subgradient optimization iterations required to solve a single longest antisymmetric path instance over all instances during the algorithm.*

Proof. In iteration $i+1$ of algorithm k-Longest-Antisymmetric-Paths, every computed path deviating from p^i at vertex p_j^i must satisfy two conditions in order to form an antisymmetric path in $\mathcal{G}$:

1. The spur path from p_j^i to t contains no pair of incompatible vertices.
2. No vertex in the spur path from p_j^i to t is incompatible to any vertex of the root path from s to p_j^i.

The first condition is satisfied by application of algorithm Lag-Antisymmetric-Path, because if we set the source vertex $s = p_j^i$, every optimal solution corresponds to a longest antisymmetric p_j^i-t path. To meet the second condition, it is sufficient to remove all vertices from the spectrum graph that are incompatible to some vertex of root path $\langle p_1^i, \ldots, p_j^i \rangle$ before computing the longest antisymmetric p_j^i-t path. For each of the k iterations, at most ℓ antisymmetric deviation paths must be computed, each requiring $\mathcal{O}(\beta)$ calls to algorithm Dag-Longest-Path, which has complexity $\mathcal{O}(|E_D| + |V|)$. The optimality of the computed k antisymmetric paths follows from the correctness of Yen's algorithm. $\square$

Note that the value of β is possibly exponential if large parts of the branch and bound tree have to be enumerated. However, in Section 3.5.1 we will show that for realistic spectrum graph instances our formulation requires only very few iterations on average and achieves a good practical performance.

3.3.1 Enumerating longest unconstrained paths

When paths are not required to be simple, there exist faster algorithms to enumerate k shortest s-t paths in a directed graph with n vertices and m edges with non-negative weights than the one proposed by Yen, which runs in $\mathcal{O}(kn(m+n\log n))$.

The theoretically most efficient algorithm known to date is by Eppstein [1998] with a worst case time complexity of $\mathcal{O}(m + n \log n + k)$.

If the considered set of ion types is basic, the problem of enumerating the k longest antisymmetric paths reduces to the enumeration of the k longest paths in the acyclic matrix spectrum graph. Hence, after negating all edge weights, we can solve this problem using algorithms that solve the unconstrained shortest path problem like the one by Eppstein. For our benchmark study we implemented a simpler algorithm that has a higher worst case complexity but shows sufficient practical performance, as demonstrated in the benchmarks in Section 3.5.1. This algorithm, which is a modification of Yen's algorithm proposed by de Queirós Vieira Martins et al. [1999], improves Yen's worst case time complexity on acyclic graphs from $\mathcal{O}(kn(m+n))$ to $\mathcal{O}(kn^2)$. The idea of this algorithm is as follows: A straight forward adaption of Yen's algorithm to longest paths in directed acyclic graphs computes in iteration $i+1$ for every vertex $v = p^i_d, \ldots, p^i_{n_i}$ with $p^i_d = d(p^i)$, the longest v-t spur path by calling algorithm Dag-Longest-Path. These path computations can be avoided by using a longest path tree T that contains for every vertex v the longest path to vertex t and the associated weight of this path, denoted by $\hat{w}(v)$. This tree can be computed by reversing all edges and calling algorithm Dag-Longest-Path, using vertex t as the source. Note that T must be computed only once before the enumeration of longest paths. Now, using the information stored T, the length of the longest p^i_j-t spur path can be determined by inspecting all vertices $v' \neq p^i_{j+1}$ with $(p^i_j, v') \in E$ and selecting the vertex v^* that maximizes the sum $\hat{w}(v^*) + w(p^i_j, v^*)$. Again, for $p^i_j = d(p^i)$ only vertices $v' \neq p^l_{j+1} \, \forall p^l \in \mathcal{D}(p^i)$ are considered to avoid repeated computation of previous paths. The longest p^i_j-t spur path is then finally obtained by backtracking in T.

The generation of T requires time $\mathcal{O}(n + m)$, and in each of the k iterations in Yen's algorithm at most n deviation paths must be computed, where the identification of vertex v^* and the backtracking are both performed in time $\mathcal{O}(n)$. Hence, the complete algorithm runs in time $\mathcal{O}(kn^2)$, which could even be improved to $\mathcal{O}(km+n)$ by backtracking only the spur path for the longest path in X at the end of each of iteration. This could be further improved to $\mathcal{O}(m \log n + kn)$ by computing reduced edge costs and sorting edges (details omitted, see [de Queirós Vieira Martins et al., 1999]). However, our running time benchmarks suggest that for relatively small values of k, which are typically used for *de novo* peptide sequencing, the total running time is dominated by the matrix spectrum graph construction and the computation of T, even without these further improvements.

3.3.2 Backward heuristic

To avoid unnecessary calls to algorithm LAG-ANTISYMMETRIC-PATH, we can incorporate the longest path tree approach for unconstrained paths into algorithm k-LONGEST-ANTISYMMETRIC-PATHS as a heuristic. In the remainder, we will refer to the following heuristic as *backward heuristic*. In iteration i of k-LONGEST-ANTISYMMETRIC-PATHS, before computing a longest antisymmetric p_j^i-t spur path, we use the longest path tree to generate a candidate p_j^i-t spur path. If the concatenation of the root path and the candidate spur path does not contain any pair of incompatible vertices, we have found the longest antisymmetric deviation s-t path deviating at vertex p_j^i. Hence, (only) in this case we can skip the call to algorithm LAG-ANTISYMMETRIC-PATH. Note that this heuristic does not affect the optimality of our algorithm, it may only sometimes reduce the running time required to generate the k longest antisymmetric paths.

In Section 3.5.1, we will analyze the effect of the backward heuristic on the practical performance of the k-LONGEST-ANTISYMMETRIC-PATHS algorithm.

3.4 Scoring model

3.4.1 Scoring vertices in the spectrum graph

For our *de novo* peptide sequencing tool ANTILOPE, we implemented a probabilistic scoring model to compute vertex scores in the spectrum graph based on Bayesian networks, similar to PepNovo. A Bayesian network is defined as a directed acyclic graph, where vertices correspond to random variables and directed edges model conditional dependencies between them. Therefore, pairs of random variables with respective vertices not connected by directed edges are considered as conditionally independent.

In our scoring model, we create a random variable for every considered *scored ion type* $\tau' \in \mathcal{T}'$ to represent the observed intensity of the associated peak (possibly missing) in the mass spectrum. Note that the set of scored ion types is not necessarily equal to the set of ion types $\mathcal{T}$ used for the construction of the spectrum graph. Instead, usually more ion types are considered for scoring such that $\mathcal{T} \subseteq \mathcal{T}'$. The Bayesian network approach is used to capture dependencies between observed peak intensities of different ion types that arise since they are products of fragmentation events that cannot be considered as independent. For example, we expect a relationship between the intensities of complementary b- and y-ion peaks, as both ions result from the same fragmentation event. For the same reason, we expect a relationship between the intensities of a b- or y-ion and its associated neutral loss variants, e.g., b-H_2O and y-H_2O.

We normalize observed peak intensities using the approach proposed by Dančík et al. [1999] and apply binning according to peak intensity ranks. Thus, for a given spectrum and a given number of intensity levels N_B, we calculate the normalized intensity norm(π) of some peak with intensity rank r as

$$\text{norm}(\pi) = \max(0, N_B - \left\lfloor \frac{r}{B} \right\rfloor),$$

with bin size B calculated as $B = \lceil M_p/100 \rceil$ for parent mass M_p. As depicted in Figure 3.6, the Bayesian network contains an additional *class* vertex representing the two classes: true prefix residue mass (*True*) and false prefix residue mass (*False*). This class vertex has an outgoing edge to every other vertex in the network.

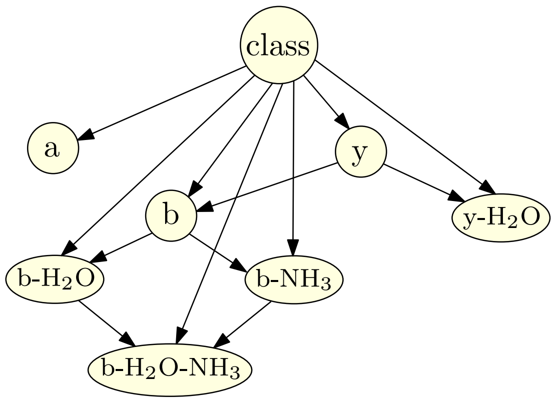


Figure 3.6: *Sample Bayesian network for spectrum graph vertex scoring with a set of seven scored ion types.*

In practice, different mass spectrometer types and experimental setups can produce tandem mass spectra with considerable differences in the peak intensity distribution for certain ion types. Hence, the ability to train the scoring model individually for the specific experimental equipment, based on a set of reliably identified spectra, is to the best advantage for successful peptide sequencing. For this reason, we implemented a flexible interface that allows for parameter training on a customized Bayesian network structure or automatic inference of a suitable network structure, based on a training dataset. We achieve this flexibility of the training process by resorting to the machine learning suite Weka [Hall et al., 2009], which offers a convenient interface to either define the network structure manually or learn a structure based on different algorithms (e.g., K2-HillClimber) and metrics (e.g., Bayesian metric for local scoring [Bouckaert, 2004]).

Given a set of identified tandem mass spectra, in our training procedure we compute all prefix residue masses according to the known peptide sequence. Then, for each prefix residue mass, we construct a vector of observations containing for each scored ion type the normalized intensity of the peak that is nearest to the expected m/z position, known from the mass offsets defined in Table 3.2. If no peak is found within a certain window (default: 0.5 Th), the corresponding observed intensity is set to zero. In addition to these positive training vectors, we generate an equal number of negative training samples containing the observed intensities for false prefix residue masses. In contrast to Datta and Bern [2009], we create negative training vectors only for those false prefix residue masses that would have a vertex being created during the spectrum graph construction, since otherwise, the negative training set could contain a significant amount of vectors without any observed peaks. These vectors do not represent realistic negative samples, as, in our *de novo* sequencing algorithm, we only compute scores for prefix residue mass candidates that are represented by a vertex, and hence at least one peak must have been observed. As intensity distributions can vary with the relative position of the fragmentation site within the intact peptide, we follow the approach implemented in PepNovo and group the candidate prefix residue masses into a certain number (default: 3) of equally sized m/z regions and train an individual Bayesian network for each region.

Once the network structure for each region has been determined (inferred or custom), and the conditional probability tables have been computed, we use the network to generate spectrum graph vertex scores as follows: For the mass of a given vertex v, we construct the vector of observed normalized intensities I_v in the same way as for the training samples. Using the trained Bayesian network BN for the corresponding m/z region of vertex v, we compute the log likelihood ratio (LLR(v)) as follows:

$$\mathrm{LLR}(v) = \log \frac{\Pr(I_v \mid BN, \mathit{class} = \mathit{True})}{\Pr(I_v \mid BN, \mathit{class} = \mathit{False})}\,, \tag{3.11}$$

where $\Pr(I_v \mid BN, \mathit{class} = c)$ for $c \in \{\mathit{True},\mathit{False}\})$ denotes the probability of the observed intensity vector I^v under model BN when the *class* variable is set to c. This value is defined as

$$\Pr(I_v \mid BN, \mathit{class} = c) := \prod_{\tau' \in \mathcal{T}'} \Pr(I_v[\tau'] \mid I_v[{\tau'}^{in}_1], \ldots, I_v[{\tau'}^{in}_r], \mathit{class} = c)\,,$$

where $\{{\tau'}^{in}_1, \ldots, {\tau'}^{in}_r\}$ is the set of all scored ion types that are parents of τ' in BN and $I_v[\tau]$ denotes the observed normalized intensity for ion type τ.

We combine the score obtained from the Bayesian network with an intensity rank score $S_R(v)$ as implemented in the INSPECT algorithm [Tanner et al., 2005]. Assume vertex v was constructed for some peak π with rank r (not binned) to interpret π as an ion of type $\tau \in \mathcal{T}$. Then, we compute $S_R(v)$ as the log ratio of the odds that a peak with rank r is produced by an ion of type τ and the odds of a randomly chosen peak being produced by an ion of type τ. Again, we split the mass spectrum into equally sized m/z regions and learn these scores individually for each region.

The final score for each vertex v of the spectrum graph is then given by

$$\operatorname{score}(v) = \operatorname{LLR}(v) + S_R(v)\,. \tag{3.12}$$

Before generating the vertex set, we apply a window filter, keeping only a given number (default 3) of highest intensity peaks for every window of a given mass width (default 56 Da). To speed up the algorithm, we reduce the size of the spectrum graph by removing low scoring vertices with negative scores before we compute antisymmetric paths, since these vertices are unlikely to represent true prefix residue masses. For the filtered spectrum graph, we generate directed edges corresponding to combinations of up to three amino acids, where edges that correspond to combinations of amino acids are charged a penalty such that single amino acid edges are preferred in longest path computations.

3.4.2 Re-scoring of candidate peptides

For every computed antisymmetric *s*-*t* path, in a second step we resolve edges labeled with multiple combinations of amino acids and generate a candidate sequence for every combination of possible edge labels along the path. We re-score these candidates by a weighted version of shared peaks count to reward abundant and penalize missing peaks.

Given a candidate peptide sequence P', for every prefix residue mass, we search for peaks in the query spectrum at the expected positions for all considered scored ion types. If we find a peak π for ion type τ, we add some ion type specific score γ_τ. In addition, we determine whether the peak is a *primary isotopic peak*, a *secondary isotopic peak*, or a *lone peak*. According to the definition by Tanner et al. [2005], a peak is called primary isotopic if the spectrum contains a so-called *child peak* at an m/z offset of 1 Th for singly charged ions or 0.5 Th for double charged ions. Conversely, for a secondary isotopic peak, the mass spectrum contains a so-called *parent peak* at m/z offset -1 Th for singly charged and -0.5 Th for doubly charged ions. If a peak is neither a primary nor a secondary isotopic peak, it is called a lone peak. A secondary peak has a lower probability to correspond to one of the scored ion types [Tanner et al., 2005], as it is more likely to correspond to a heavier isotope of some other ion. Therefore, we charge a penalty determined by parameter η_s for a peak classified as secondary isotopic. On the other hand, a peak classified as primary isotopic is rewarded an additional bonus, defined by parameter η_p, to account for the presence of additional isotopic peaks. Finally, if a peak is missing, we charge an ion type specific penalty $\sigma_\tau < 0$.

Hence, the score PSM(π, τ) of a (possibly missing) peak π, interpreted as ion type τ, is defined as follows:

$$\text{PSM}(\pi, \tau) := \begin{cases} \gamma_\tau & \text{if } \pi \text{ lone,} \\ \gamma_\tau(1 - \eta_s) & \text{if } \pi \text{ secondary isotopic,} \\ \gamma_\tau(1 + \eta_p) & \text{if } \pi \text{ primary isotopic,} \\ \sigma_\tau & \text{if } \pi \text{ missing.} \end{cases} \tag{3.13}$$

The final score of candidate peptide sequence P' is then defined as the sum over all peak scores of the scored ion types for every prefix residue mass of P'. In the final step of the algorithm, we rank the candidates according to this score and return a predefined number of highest scoring peptide candidates.

3.5 Results

In this section, we evaluate the practical performance of our Lagrangian relaxation algorithm compared to directly solving the ILP formulation and the matrix spectrum graph based polynomial time algorithm. Further, we compare the peptide sequencing performance of our tool ANTILOPE to the performance of established *de novo* peptide sequencing software. We implemented the described algorithms in C++, using data structures of the OpenMS library for handling spectral data [Sturm et al., 2008] and the graph data structures and algorithms from the SeqAn library [Döring et al., 2008]. All computations were performed on a machine equipped with 2 Intel Xeon CPU X5550 @2.67GHz Quad Core and 72 GB memory, but all algorithms were limited to a single thread.

3.5.1 Running time

To assess the performance of our Lagrangian relaxation algorithm k-LONGEST-ANTISYMMETRIC-PATHS, we analyzed the running time required to generate the longest 20, 50, and 100 antisymmetric paths for a set of 1656 real tandem mass spectra from the ISB dataset [Keller et al., 2002], further described in Section 3.5.2. We compared this running time to the time required for solving the ILP formulation directly, using the well recognized commercial solver software CPLEX (version 12.4) [IBM, 2011], and the running time of the matrix spectrum graph approach, as described in Section 3.3.1. As the latter is limited to basic ion types, we created spectrum graphs for two sets of basic ion types and different vertex scoring functions. For the ILP formulation solved with CPLEX, we generated suboptimal solutions by cutting off previous solutions, as described in Section 3.3. Further, we studied the effect of applying the backward heuristic to the k-LONGEST-ANTISYMMETRIC-PATHS algorithm, as introduced in Section 3.3.2.

Benchmark 1

For the first benchmark, we generated spectrum graphs using the set of basic ion types containing b-, y-, a-, b-NH_3-, and y-H_2O-ions and a simple scoring function that uses normalized peak intensities as score for all generated vertices. As this scoring function does not discriminate between vertices produced by the same peak, it becomes more likely that simple longest *s*-*t* paths would contain pairs of incompatible vertices. Further, we did not filter low scoring vertices. Hence, this set represents relatively difficult spectrum graph instances with an average number of 303 vertices. In Figure 3.7 we present the average running time per spectrum for each of the four compared algorithms. Solving the ILP formulation with CPLEX was more than one order of magnitude slower than all other algorithms for the generation of 100 longest paths, where it required on average 24 seconds per spectrum. Among the remaining three methods, the polynomial algorithm using the matrix spectrum graph (MAT) achieved the best performance. The running time advantage compared to the Lagrangian relaxation approach (LAG) increased with the number of suboptimal solutions, up to a factor of $\sim$5 for the 100 longest paths. Further, we observed only a slight performance gain of up to 10% for the longest 100 paths when using the backward heuristic for the Lagrangian relaxation approach (LAG-B). This suggests that in many cases the candidate path obtained by the backward heuristic contains incompatible vertices, and thus algorithm LAG-ANTISYMMETRIC-PATH must be applied. Even for the slower variant without the heuristic, the 20 longest paths were computed on average in less than 0.3 seconds per spectrum, which is comparable to the running time of other *de novo* sequencing algorithms like PepNovo and NovoHMM.

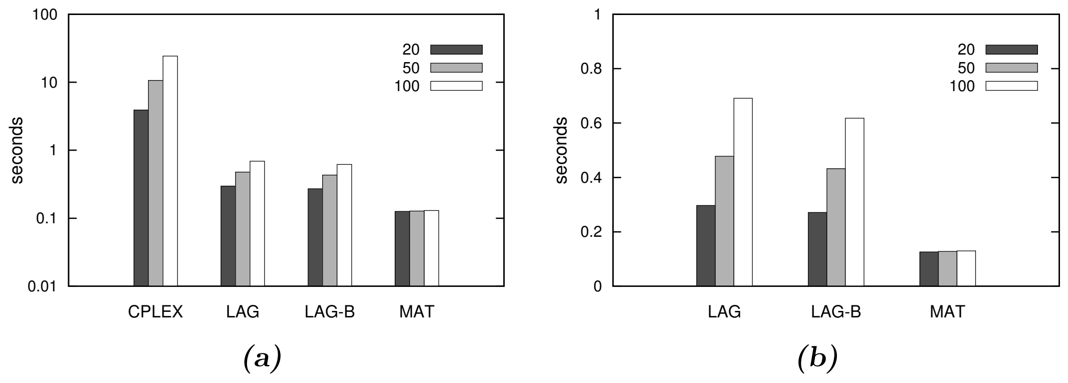


Figure 3.7: *Average running times per spectrum for generating the 20, 50 and 100 longest antisymmetric paths in Benchmark 1. Note the logarithmic scale of plot (a). Plot (b) shows the running times of (a) in non-logarithmic scale, excluding the running times for CPLEX.*

However, in the second benchmark we will show that for more realistic spectrum graph instances the average running time per spectrum decreases significantly.

Benchmark 2

In the second benchmark, we applied the default settings of ANTILOPE for spectrum graph generation, using the basic set containing b- and y-ions with vertex scores computed according to the scoring function outlined in Section 3.4.1 (see Section 3.5.2 for training details). Hence, this set represents more realistic spectrum graph instances with an average number of 80 vertices after filtering low scoring vertices. As depicted in Figure 3.8, solving the ILP formulation with CPLEX was this time even more than two orders of magnitude slower than the other three approaches for the 100 longest paths. For this set of spectrum graph instances, the gain in performance due to the backward heuristic was more prominent, yielding a factor of three for the 100 longest paths. The Lagrangian relaxation together with the backward heuristic was even slightly faster than the matrix spectrum graph algorithm for the 20 and 50 longest paths. This advantage decreases with increasing number of paths as the efficient path enumeration for the matrix spectrum graph amortizes the construction overhead. For the computation of the 100 longest paths, we measured an average time per spectrum of ~0.024 seconds without the backwards heuristic and less than 0.01 seconds for the matrix spectrum graph and the Lagrangian relaxation with backward heuristic. Hence, the time required for candidate path generation using algorithm k-LONGEST-ANTISYMMETRIC-PATHS is negligible for the complete sequencing algorithm, especially when using the backward heuristic, as the running time is

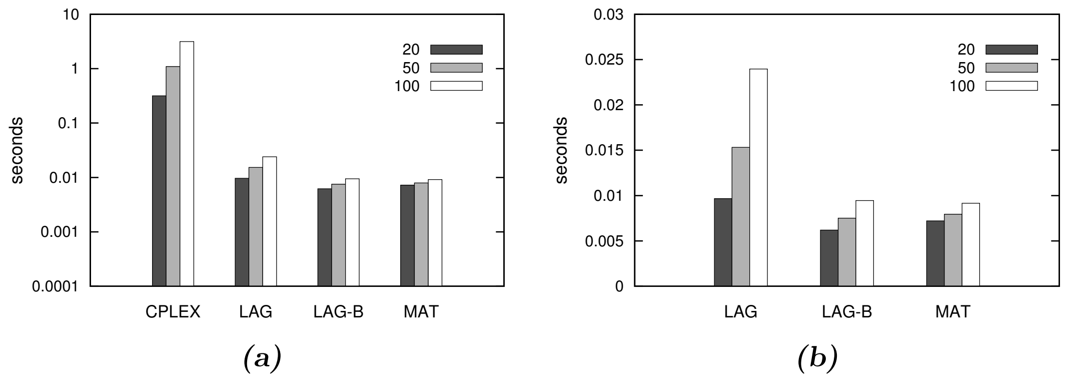


Figure 3.8: *Average running times per spectrum for generating the 20, 50 and 100 longest antisymmetric paths in Benchmark 2. Note the logarithmic scale of plot (a). Plot (b) shows the running times of (a) in non-logarithmic scale, excluding the running times for CPLEX.*

mostly dominated by the re-scoring step.

3.5.2 Sequencing performance

To assess the quality of *de novo* peptide predictions of our tool ANTILOPE, we conducted a comparative benchmark study with the four non-commercial tools LutefiskXP[2], NovoHMM, PILOT[3], and PepNovo[4] and the commercial software PEAKS[5]. We evaluated peptide predictions in terms of recall and accuracy, which we define as follows:

$$\text{recall} := \frac{\#\text{correctly predicted amino acids}}{\#\text{amino acids in unknown peptides}}\,,$$

$$\text{accuracy} := \frac{\#\text{correctly predicted amino acids}}{\#\text{predicted amino acids}}\,.$$

For each tool, except NovoHMM, which generates only a single prediction per spectrum, we computed the performance considering the top scoring 1, 3, 5, and 10 candidates, where we always selected the prediction with the highest recall. In case of multiple predictions with the same recall, we selected the one with maximal accuracy.

For this benchmark, we selected a set of reliably annotated tandem mass spectra from tryptic peptides, which had already been used for training and evaluating PepNovo and NovoHMM. These spectra originate from the ISB dataset of Keller et al. [2002], produced by a Thermo Finnigan ESI-ion trap mass spectrometer, and from the open proteomics database. From this set, we selected 1214 spectra generated by doubly charged precursor ions as training set for the Bayesian network scoring model. For each of the three m/z regions, only ion types observed for at least 20% of all true prefix residue masses in the training samples were included into the final Bayesian network model (see Figure 3.9).

We generated vertices for the basic set of b-ions and y-ions, and we used the following parameters to score peptide-spectrum matches in the candidate re-scoring step: For singly charged b- and y-ions, we used the scores $\gamma_b = \gamma_y = 1$, and for their doubly charged variants we used $\gamma_{b^2} = \gamma_{y^2} = 0.5$. In addition, we scored a-ions with $\gamma_a = 0.3$ and all neutral losses with 0.2. For the parameters to reward primary isotopic peaks and penalize secondary isotopic peaks, we set $\eta_p = \eta_s = 0.2$, and for every missing peak of type τ we added value $\sigma_\tau = -\gamma_\tau/2$.

[2] LutefiskXP version 1.0.5

[3] As PILOT was not available, the identifications for the test data were generated by the authors of PILOT.

[4] PepNovo+ version 3.1

[5] PEAKS 7 (30 days trial version)

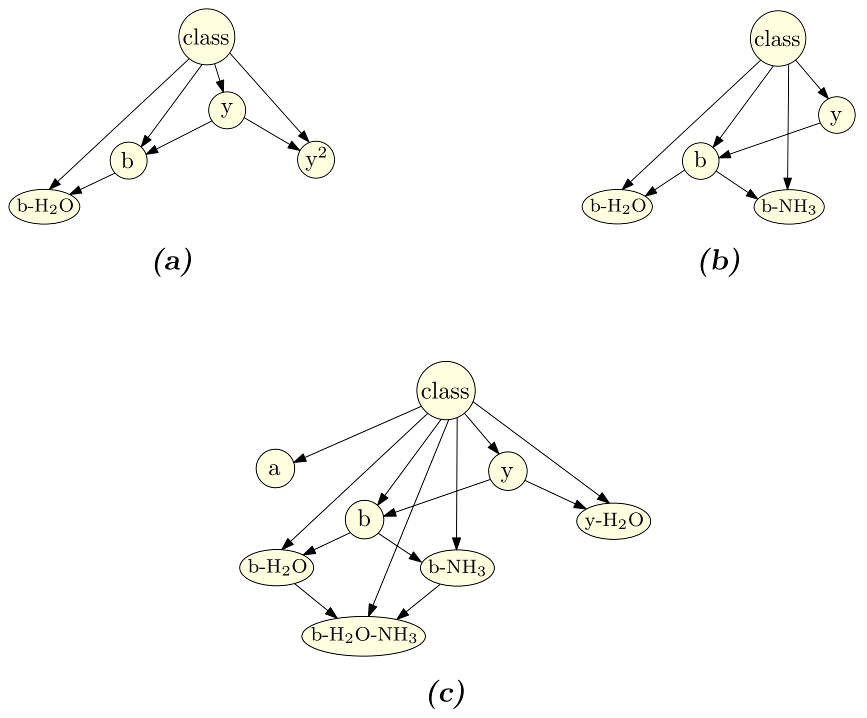
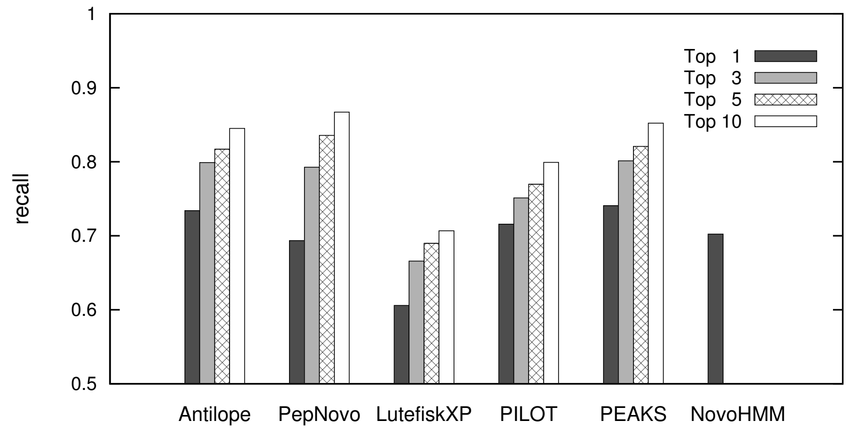


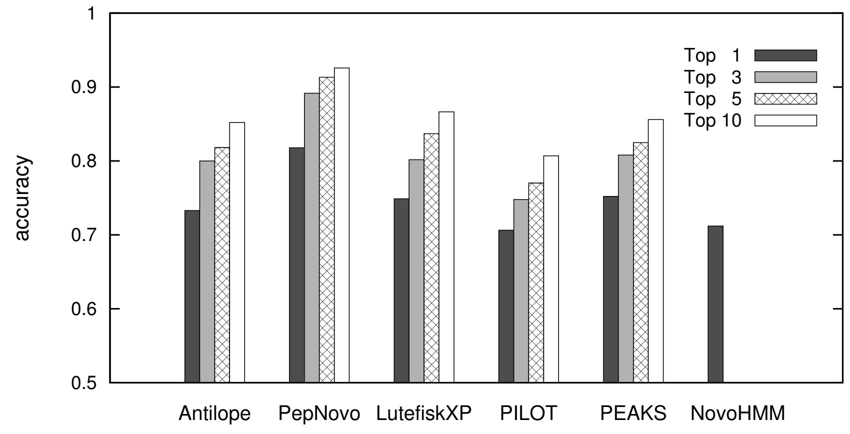
Figure 3.9: *The Bayesian network structures used in benchmark study for (a) low m/z region, (b) high m/z region, and (c) center m/z region.*

We tested on a set of 200 tandem mass spectra, not included in the training set, that were produced by doubly charged precursor ions from peptides with a molecular mass below 1600 Da and an average length of ∼10 amino acids. For an unknown peptide $P = \langle p_1, p_2, \ldots, p_k \rangle$ and a candidate peptide sequence $P' = \langle p'_1, p'_2, \ldots, p'_{k'} \rangle$, we considered amino acid p_i as correctly predicted if p_i and p'_j were equal for some $1 \leq j \leq k'$ and $|m_r(\langle p_1, \ldots, p_{i-1} \rangle) - m_r(\langle p'_1, \ldots, p'_{j-1} \rangle)| \leq 2.5\,\text{Da}$. For the calculation of recall and accuracy, we did not discriminate between the isobaric amino acids Isoleucine and Leucine and the near isobaric amino acids Lysine and Glutamine.

The results for all tools are summarized in Figure 3.10. Considering only the first top scoring prediction, we observed that the recall values of ANTILOPE (∼73.4%) and PEAKS (∼74.1%) were almost equal and slightly higher than for PILOT (∼71.5%), NovoHMM (∼70.2%), and PepNovo (∼69.4%). ANTILOPE and NovoHMM both compute only peptide sequences without gaps, which results in only small differences between recall and accuracy values. LutefiskXP, PILOT and PepNovo predict also gapped candidates, containing unexplained mass differences, e.g., YGLAVA[198.1]K with an internal mass gap of 198.1 Da or [220.1]LAVAVVK with an N-terminal mass gap of 220.2 Da. For PepNovo and LutefiskXP, this led to an improvement in terms of accuracy, which exceeds the



(a)



(b)

Figure 3.10: *Comparison of peptide sequencing performance in terms of (a) recall and (b) accuracy between* ANTILOPE*, NovoHMM, PepNovo, PILOT, PEAKS, and LutefiskXP. We compared the accuracy and recall of the best prediction among the top 1, 3, 5 and 10 ranked candidates returned by each tool. As the best prediction we selected the one with the highest recall among all candidates. Note that NovoHMM generates only one candidate per spectrum.*

recall for both tools.

Considering also suboptimal solutions, we observed that ANTILOPE was always very close to PepNovo and PEAKS in terms of recall (almost equal for the top 3, ∼2.5 percentage points advantage of PepNovo for the top 10) and at least 4 percentage points better than PILOT. In terms of accuracy, PepNovo achieved the best results (∼92.6% for top 10) due to gapped peptide predictions. Also LutefiskXP showed a slightly better performance than ANTILOPE and PILOT in terms of accuracy, but this was attained at a much lower recall, which was between 12 and 14 percentage points lower than for ANTILOPE in all four cases.

Looking at the average running time per spectrum, ANTILOPE (∼0.2 seconds) was slightly faster than PepNovo (∼0.4 seconds) and eight times faster than LutefiskXP (∼1.6 seconds). For PEAKS and NovoHMM we observed average running times of ∼0.1 seconds and ∼0.5 seconds respectively, both run on a comparable hardware configuration with Microsoft Windows operating system.

3.6 Conclusion

In this chapter, we introduced an efficient and flexible algorithmic approach to the *de novo* peptide sequencing problem using Lagrangian relaxation combined with a k longest path enumeration algorithm. Even though our formulation does not restrict the structure of forbidden pairs in the spectrum graph, we were able to show that the performance of our algorithm is comparable to the polynomial time algorithm based on the matrix spectrum graph for restricted sets of ion types. Compared to solving the equivalent ILP formulation with the commercial CPLEX ILP solver software, our algorithm was more than two orders of magnitude faster and required on average less than a hundredth of a second to generate the 100 longest antisymmetric paths for realistic spectrum graph instances.

For the sequencing benchmark, we restricted the vertex generation to the set of basic ion types b, and y, as we experienced a decline in sequencing performance for the benchmark spectra when generating also vertices for doubly charged ion types. Nevertheless, we emphasize that the capability of our approach to handle such sets of non-basic ion types, in combination with an enhanced scoring function, can lead to an improved sequencing performance for certain kinds of spectra, as it allows to incorporate more knowledge into the spectrum graph. In addition, the flexibility of the ILP formulation paves the way for individual adaptions to special properties of certain kinds of tandem mass spectra. One example is the possibility to introduce internal fragments into the model, which are frequently observed in HCD spectra [Michalski et al., 2012]. This can be achieved by introducing constraints into the model to ensure that at most two vertices of a given triple may be selected simultaneously, to avoid a conflicting interpretation of a peak as terminal fragment ion and internal fragment ion. While our algorithm is primarily designed to compute antisymmetric paths, the formulation can be further generalized to allow for the simultaneous selection of incompatible vertices, charged with an individual penalty. This could be generalized even further to incorporate pairwise vertex scores (resp. edge scores) for pairs of non-adjacent and non-incompatible vertices (resp. edges), to capture global information about fragmentation patterns or amino acid composition. For the ILP formulation and our associated Lagrangian relaxation algorithm, we can easily implement such new constraints into the model without inducing elementary changes to the general algorithm.

Further, we implemented a generic scoring function to generate discriminative vertex scores and we were able to show that the performance of our *de novo* peptide sequencing tool ANTILOPE is on a par with established tools. A central step in the algorithm, which affects the prediction quality, is the re-scoring of candidate peptides. Thus, it is worthwhile to devote further effort to develop a more sophisticated method like, e.g., the rank boosting approach [Frank, 2009], implemented in PepNovo. Ideally, this method should also allow for automated individual tuning based on annotated training datasets. Another point to be addressed is the introduction of post translational modifications into the model. The straightforward approach is to augment the set of *normal* amino acids with modified amino acids for a given set of modifications to be considered during sequencing. While this approach does not imply any changes to the algorithm, it results in denser spectrum graphs as the number of edges is expected to increase with the number of considered modifications.

Algorithm 1 DAG-LONGEST-PATH

INPUT: Acyclic Graph $G = (V, E)$ with edge weights w. Vertices s and t
OUTPUT: Longest s-t path in G

```
1:  for each vertex $v \in V$ do                                  // Initialization
2:      $dist[v] \leftarrow -\infty$
3:      $pred[v] \leftarrow$ NIL
4:  end for
5:  $dist[s] \leftarrow 0$
6:  for each vertex $u$ in topologically sorted order do
7:      if $dist[u] \neq -\infty$ then
8:          for each edge $(u, v) \in E$ do
9:              if $dist[v] < dist[u] + w(u, v)$ then
10:                 $dist[v] \leftarrow dist[u] + w(u, v)$
11:                 $pred[v] \leftarrow u$
12:             end if
13:         end for
14:     end if
15: end for
16: if $dist[t] \neq -\infty$ then                                // Traceback path
17:     add $t$ to $path$
18:     $v \leftarrow t$
19:     while $v \neq s$ do
20:         $v \leftarrow pred[v]$
21:         add $v$ to front of $path$
22:     end while
23:     return $path$
24: else
25:     return "no path"
26: end if
```

Algorithm 2 k-LONGEST-ANTISYMMETRIC-PATHS

INPUT: Acyclic Graph $G = (V, E)$ with edge weights w. Vertices s and t. k
OUTPUT: k longest antisymmetric s-t paths $\langle p^1, \dots, p^k \rangle$ in G.
REMARKS: Lines 2 and 19 call algorithm LAG-ANTISYMMETRIC-PATH

```
 1: $X \leftarrow$ empty candidate set
 2: $p^1 \leftarrow$ longest antisymmetric $s$-$t$ path in $G$
 3: $d(p^1) \leftarrow s$                                   // set deviation vertex of $p^1$ to $s$

 4: for $i \leftarrow 2$ to $k$ do
 5:     $d \leftarrow$ deviation index of $p^{i-1}$                           // $d(p^{i-1}) = p^{i-1}_d$

 6:     if $i > 2$ then                    // force deviation from all paths in $\mathcal{D}(p^{i-1})$
 7:         $p^{par} \leftarrow p^{i-1}$
 8:         while $p^{par} \neq p^1$ & $d(p^{par}) = d(p^{i-1})$ do
 9:             $p^{par} \leftarrow parent(p^{par})$            // $parent(p^{par})$ returns parent of $p^{par}$
10:             remove edge $(p^{par}_d, p^{par}_{d+1})$ from $E$
11:         end while
12:     end if

13:     for $j \leftarrow 1$ to $d - 1$ do
14:         remove all edges $(v_u, v_w)$ from $E$ for all $v_u$ incompatible to $p^{i-1}_j$
15:     end for

16:     for $m \leftarrow d$ to $n_{i-1} - 1$ do                        // compute spur paths
17:         remove edge $(p^{i-1}_m, p^{i-1}_{m+1})$ from $E$                // deviation from $p^{i-1}$
18:         remove all edges $(v_u, v_w)$ from $E$ for all $v_u$ incompatible to $p^{i-1}_m$
19:         $p^{\text{spur}} \leftarrow$ longest antisymmetric $p^{i-1}_m$-$t$ path in $G$
20:         if $p^{\text{spur}}$ is non empty then
21:             $p^{\text{cand}} \leftarrow \langle p^{i-1}_1, \dots, p^{i-1}_{m-1} \rangle \cdot p^{\text{spur}}$
22:             $d(p^{\text{cand}}) \leftarrow p^{i-1}_m$
23:             $parent(p^{\text{cand}}) \leftarrow p^{i-1}$
24:             insert $p^{\text{cand}}$ into $X$
25:         end if
26:     end for

27:     if X empty then                    // only $i - 1$ antisymmetric $s$-$t$ paths in $G$
28:         return $p^1, \dots, p^{i-1}$
29:     end if
30:     $p^i \leftarrow$ longest path in $X$
31:     remove $p^i$ from $X$
32:     restore all edges
33: end for

34: return $p^1, \dots, p^k$
```

CHAPTER 4

Isoform Inference And Abundance Estimation With Delayed Column Generation

In this chapter, we present our approach to the isoform inference and abundance estimation problem based on RNA-Seq data. The presented algorithm has been developed in a joint project with Dr. Stefan Canzar and Dr. David Weese.

4.1 Background

For many years the "one gene-one enzyme hypothesis" [Beadle, 1945], later reformulated to the "one gene-one polypeptide hypothesis" suggested that every gene has a unique RNA product. This simplistic assumption was confuted by the discovery of diverse post transcriptional modification mechanisms that can generate many different mRNA products for a single gene, referred to as *isoforms*. Three important post-transcriptional mechanisms that enhance the diversity of the *transcriptome*, i.e., the set of all expressed RNA molecules (mRNA, tRNA, rRNA and other non-coding RNA) in a single cell or a population of cells [Velculescu et al., 1997], are *alternative splicing*, *alternative promoter usage* and *alternative polyadenylation* (see Figure 4.1). Alternative splicing, first discovered by Berget et al. [1977] in *adenovirus*, produces different isoforms by including or excluding particular exons into the final mRNA. Alternative promoter usage, analyzed in [Davuluri et al., 2008], and alternative polyadenylation, reviewed in [Di Giammartino et al., 2011], generate isoforms with differing $5'$- and $3'$-ends respectively.

First large scale studies based on *expressed sequence tag* (EST) [Adams et al., 1991] analysis, reported in [Mironov, 1999; Brett et al., 2000; Croft et al., 2000; Kan et al., 2001], identified alternative splice variants for a fraction of 40% to 60% of all human genes [Lee and Roy, 2004]. More recent studies estimate that alternative splice variants are produced for $\sim$95% of all human multi-exon genes

[Pan et al., 2008], while alternative polyadenylation was detected in ∼70% of all human genes [Derti et al., 2012]. Alternative splicing has been shown to play decisive roles in cellular processes like cell growth, differentiation, and death [Chen and Manley, 2009], and misregulation of alternative splicing is known to be related to several human diseases [Garcia-Blanco et al., 2004; Douglas and Wood, 2011].

These findings demonstrate the need for and the potential of large scale studies to gain more insights into the particular roles and functions of different gene products and into the mechanisms regulating alternative splicing.

One approach to large scale analysis of RNA expression and detection of alternative splicing is by using *DNA-microarrays*. There exist three groups of microarrays for transcriptome analysis: *exon arrays*, *exon junction arrays*, and high density *genome tiling arrays*. All three kinds of microarrays can be used to detect alternative splicing events for known genes, as reported for exon arrays in [Clark et al., 2007; Laajala et al., 2009; Moller-Levet et al., 2009; Chen et al., 2011] and for exon junction arrays in [Johnson et al., 2003; Pan et al., 2004; Fagnani et al., 2007; Shen et al., 2010]. In addition, high density genome tiling arrays can be used to identify new transcripts from previously un-annotated genes [Kapranov et al., 2002; Kampa et al., 2004; Cawley et al., 2004; Stolc et al., 2005], as well as alternative splicing events [Ner-Gaon and Fluhr, 2006; Eichner et al., 2011]. However, compared to sequencing-based approaches, discussed below, hybridization methods like microarrays have several limitations like, e.g., their dependence on a known genome sequence, a high background signal induced by cross-hybridization [Okoniewski and Miller, 2006; Royce et al., 2007], and a limited dynamic range of detection due to signal saturation [Wang et al., 2009]. In addition, expression ratios measured for different probes are not directly comparable due to different signal-to-noise ratios [Forrest and Carninci, 2009].

As an alternative to microarrays, sequencing based methods can be used to detect and quantify individual RNAs in a sample without knowledge of the genome sequence. ESTs, obtained by Sanger sequencing, have been the major source for the majority of annotated regions in mammalian genomes [Forrest and Carninci, 2009]. However, for transcriptome scale analyses, Sanger EST sequencing has several drawbacks as it is relatively low throughput, in general not quantitative, and expensive [Wang et al., 2009]. While other, *tag based* methods like *cap analysis of gene expression* (CAGE) [Kodzius et al., 2006], *serial analysis of gene expression* (SAGE) [Velculescu et al., 1995], and *massively parallel signature sequencing* (MPSS) [Brenner et al., 2000] allow for large scale analysis of gene expressions, they often cannot distinguish between isoforms, as they measure only few positions along the transcript [Forrest and Carninci, 2009].

The RNA-Seq method (the term RNA-Seq was coined by Mortazavi et al.

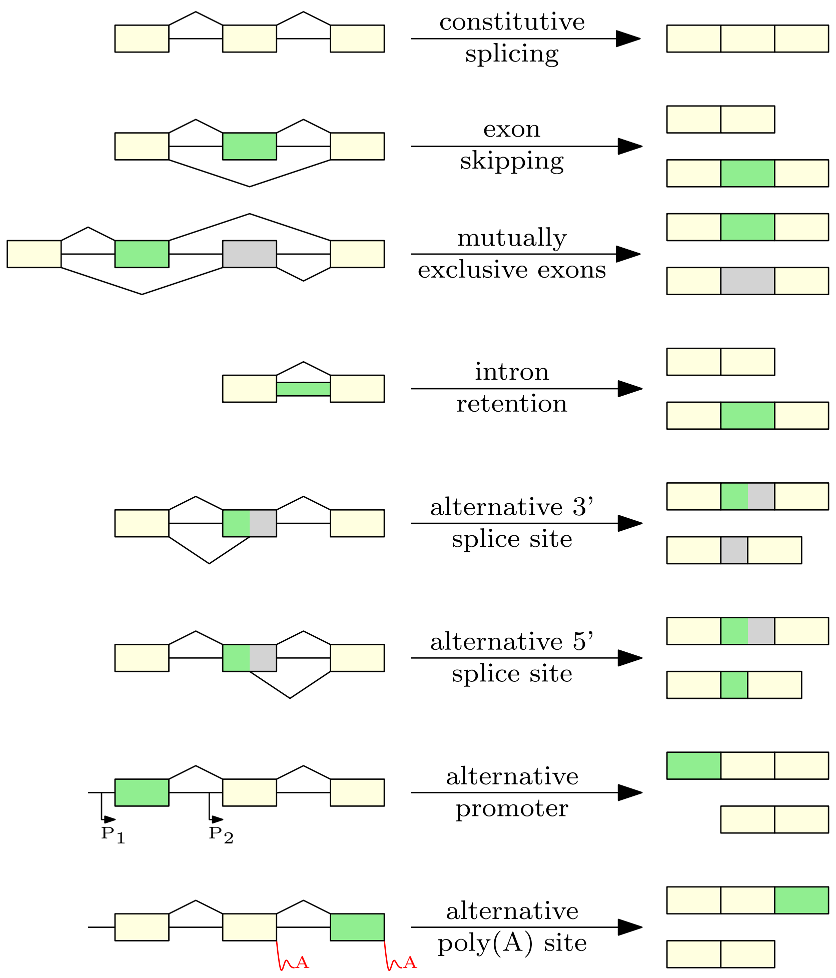


Figure 4.1: *Types of alternative splicing events, alternative promoter usage and alternative polyadenylation. Splice patterns on the left side and corresponding alternative splicing products on the right in. Every box represents an exon in $5' \rightarrow 3'$ direction, lines denote introns and arcs denote splice junctions. Constitutive exons in yellow, alternative exons in green and grey.*

[2008]) overcomes these shortcomings by using second generation high-throughput sequencing technologies like Illumina/Solexa, Roche 454, and ABI SOLiD to conduct whole transcriptome scale sequencing. Unlike tag-based methods, RNA-Seq provides sequence information along complete transcripts, which enables the identification and quantification of particular isoforms for the same gene and allows for detection of sequence variations, e.g., single nucleotide polymorphisms (SNPs) [Chepelev et al., 2009].

Moreover, RNA-Seq offers a higher sensitivity and superior quantification capabilities across a much higher dynamic range, which improves the detection of differentially expressed genes and transcripts [Zhao et al., 2014]. Due to these advantages, combined with a steadily improved quality of sequencing machines and decreasing costs, RNA-Seq is currently replacing microarrays as the method of choice for transcriptome scale analysis of gene expression [Saliba et al., 2014].

The RNA-Seq method

Depending on the exact experimental setup and sequencing architecture, the exact workflow of an RNA-Seq experiment can differ in several aspects. In general, the first step of the *cDNA library preparation* is the isolation of RNA-molecules using methods like *poly(A)-enrichment* of mRNA or *rRNA-depletion*, where the latter allows to remove the highly abundant rRNA molecules while keeping small and other non-coding RNA molecules in the sample. The library preparation involves the fragmentation of long RNA molecules into smaller pieces using either *RNA-fragmentation* by *RNA-hydrolysis* or *nebulization*, or *cDNA-fragmentation* by treatment with DNase or sonication [Wang et al., 2009]. After further, architecture- and protocol-specific preparation steps, usually involving amplification by *polymerase chain reaction* (PCR), a short sequence, called *read*, is obtained from each fragment in the library. The number and length of reads generated in a single experiment vary between different sequencing architectures, e.g., the Illumina HiSeq 2500 system[1] produces up to 300 million reads of length up to 150 bp on a single flow cell (rapid run mode).

Depending on the experimental protocol, the produced reads are either *single-end reads*, where fragments are sequenced in one direction only, or *paired-end reads*, with fragments being sequenced from both ends to generate a pair of reads.

Bioinformatics for RNA-Seq data analysis

Once the RNA-Seq reads have been produced, bioinformatics comes into play to analyze and interpret the large amount of sequencing data. The subsequent bioinformatics analysis pipeline depends on the exact experimental goals and the availability of a reference genome for the studied organism. Throughout this chapter, we focus on the analysis of RNA-Seq data for organisms with known reference genome. However, in the absence of a reference genome, *de novo* transcriptome construction methods like Velvet [Zerbino and Birney, 2008], OASES [Schulz et al., 2012] and Trinity [Haas et al., 2013] can be applied.

In the presence of a reference genome, the first step in the analysis pipeline is to map the RNA-Seq reads back to the reference genome. While there exist a multitude of efficient short read mapping tools to map genomic DNA reads back

[1] http://www.illumina.com

to the genome, e.g., [Langmead et al., 2009; Li and Durbin, 2009; Weese et al., 2012; Siragusa et al., 2013], mapping of RNA-Seq reads is further complicated, since reads spanning a splice junction (i.e., exon-exon junction) cannot be mapped as a contiguous block. One approach to tackle this problem is by mapping the reads to a "reference transcriptome" consisting of all known transcripts and by calculating the corresponding genomic positions for each mapped read afterwards. As this approach relies on the availability of a comprehensive genome annotation, it does not allow for the identification of new, previously un-annotated genes or previously unknown alternative splice variants that involve new splice junctions. To overcome this limitation, special purpose *spliced alignment* tools have been developed that allow for *de novo* detection of splice junctions [Jean et al., 2010; Wang et al., 2010; Huang et al., 2011], with TopHat [Trapnell et al., 2009] being the most popular and most intensively used tool.

After the RNA-Seq reads have been mapped to the genome, they can be used for diverse analyses like, e,g., expression analysis of known genes or individual known isoforms, detection of novel genes and isoforms, identification of differentially expressed genes or isoforms between different samples [Robinson et al., 2010; Hardcastle and Kelly, 2010; Trapnell et al., 2013], or identification of fusion genes [Kim and Salzberg, 2011; Jia et al., 2013; McPherson et al., 2011].

Isoform inference and abundance estimation

Recently, several algorithmic approaches have been published that use mapped RNA-Seq reads to estimate the individual abundances for a set of known isoforms (abundance estimation), or to infer expressed isoforms (isoform inference), together with their respective abundances. Examples of approaches towards abundance estimation of known isoforms are the two expectation maximization methods IsoEM [Nicolae et al., 2011] and RSEM [Li and Dewey, 2011], the Bayesian inference method MISO [Katz et al., 2010], and the weighted non-negative least squares method IsoformEx [Kim et al., 2011].

In the remainder of this chapter, we will focus on the problem of simultaneous isoform inference and abundance estimation. The most important published tools to solve this problem are Cufflinks [Trapnell et al., 2010], Scripture [Guttman et al., 2010], IsoInfer [Feng et al., 2010], IsoLasso [Li et al., 2011b], SLIDE [Li et al., 2011a], iReckon [Mezlini et al., 2013], CLASS [Song and Florea, 2013], Traph [Tomescu et al., 2013] and MiTie [Behr et al., 2013]. These tools differ not only in their underlying algorithmic approaches but also in the kind of information they include into the inference process. While Cufflinks, IsoLasso, CLASS, MiTie, and Traph are primarily designed to work without a reference annotation, IsoInfer, SLIDE and iReckon introduce previous knowledge about gene structure, i.e., exon boundaries, transcription start sites (TSSs), polyadenylation sites (PASs), or known splice junctions from annotated transcripts. Also from an algorithmic perspective, these tools follow different approaches. Cufflinks performs the inference of isoforms and the estimation of their abundances in two separate steps, where the inference is based on a minimum path cover formulation for the so-called *fragment graph* and the subsequent abundance estimation step employs maximum likelihood estimation. Traph uses a min-cost flow formulation and generates the set of isoforms by flow decomposition after a min-cost flow has been computed in the first step of the algorithm. The remaining approaches, except CLASS, which does not estimate abundance levels[2], solve the inference and abundance estimation problem simultaneously. iReckon implements a regularized expectation maximization approach to assign individual reads to candidate isoforms. The four approaches IsoInfer, IsoLasso, SLIDE, and MiTie are count based methods that determine for certain genomic regions, e.g., single exons, splice-junctions, or tuples of exons, the number of (paired-end) reads mapped to that region. Afterwards, they aim to infer isoforms with associated abundance estimates that explain these read counts as accurately as possible by minimizing some loss function to quantify the distance between observed and predicted counts, e.g., the residual sum of squares. While simply minimizing the loss function favors solutions with many predicted isoforms, IsoLasso, SLIDE, and MiTie perform a regularized estimation to balance between the accuracy of the predicted read counts and the number of predicted isoforms.

New algorithm

In this chapter, we will present our new algorithm CIDNE (Comprehensive Isoform Discovery and Abundance Estimation), which is based on a model similar to SLIDE and IsoLasso. The major advantage of CIDNE compared to these approaches is the ability to consider an enhanced set of candidate isoforms. SLIDE and IsoLasso both enumerate a set of candidate isoforms and use a regularized version of least squares, known as LASSO (least absolute shrinkage and selection operator) [Tibshirani, 1996], to infer a subset of expressed isoforms. For complex genes containing many exons, the number of candidate transcripts can become prohibitively large as it can theoretically grow to $2^n - 1$ for a gene with n exons. Therefore, IsoLasso, SLIDE and iReckon restrict the set of candidates to those isoforms that have all splice junctions covered by a certain number of mapped reads. In contrast, our approach incorporates an additional step into the infer-

[2] In the original publication introducing CLASS it is stated that CLASS itself does not perform abundance estimation. However, the current implementation provides abundance estimates but does not document how they are derived.

ence process, employing a delayed column generation algorithm (see Figure 4.2). In the first step (Phase I) we also apply the LASSO method to infer a subset of expressed isoforms considering only candidates with all splice junctions covered by mapped reads. Starting from this solution, in the second step (Phase II) we extend the search space to include also candidate isoforms that contain uncovered splice junctions due to their low abundance or read mapping coverage fluctuations. Using our new delayed column generation algorithm, we can generate new candidate isoforms that improve the solution of the first step without an exhaustive enumeration of this potentially huge set. Hence, CIDNE can recover expressed isoforms that are not detectable by the other tools. Since the delayed column generation approach can only be applied to linear programs, we formulate a piecewise-linear approximation of the quadratic least squares objective function.

4.2 Basic definitions and data structures

4.2.1 Definitions

We define a paired-end read $r = (r_1, r_2)$ as a pair of two reads r_1 and r_2, each having *read length* ℓ^r, that are separated by some *insert size* ℓ^s and originate from a single RNA *fragment* of *fragment length* $\ell^f = 2\ell^r + \ell^s$ (see Figure 4.3). In the remainder of this section, we present our model for the case of paired-end reads, which can easily be adapted for single-end read data.

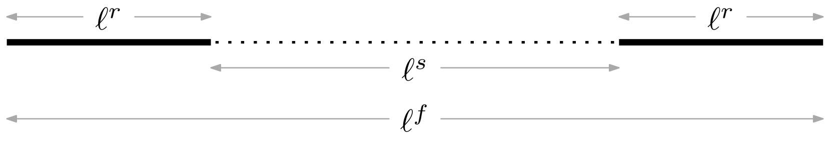


Figure 4.3: *Illustration of the definitions read length (ℓ^r), fragment length (ℓ^f) and insert size (ℓ^s) for a single fragment containing two sequenced reads (solid lines) and the insert region between them (dotted line).*

Segment

For a given set S' of exons, where every exon $e \in S'$ is defined as an interval of genomic positions, we generate the minimum cardinality set S of non-overlapping intervals containing all genomic positions in S', i.e.:

$$\forall e \in S' \; \exists S_e \subseteq S : e = \bigcup_{s \in S_e} s\,.$$

We refer to the elements of S as *segments* (see Figure 4.4).

Gene

We define a *gene* as a set S of segments, ordered according to their genomic position such that for all $1 \leq i < j \leq |S|$, segment s_i precedes segment s_j in $5' \rightarrow 3'$ direction (coding strand), also denoted as $s_i < s_j$.

Transcript and transcript sequence

For a given gene containing n segments $s_1, \dots, s_n$, we define a *transcript* $t = \langle s_{t_1}, \dots, s_{t_m} \rangle$ as a sequence of segments with $1 \leq t_1 < \cdots < t_m \leq n$. Please note that we will assume this property for every sequence of segments defined throughout this chapter. The associated *transcript sequence* denotes the

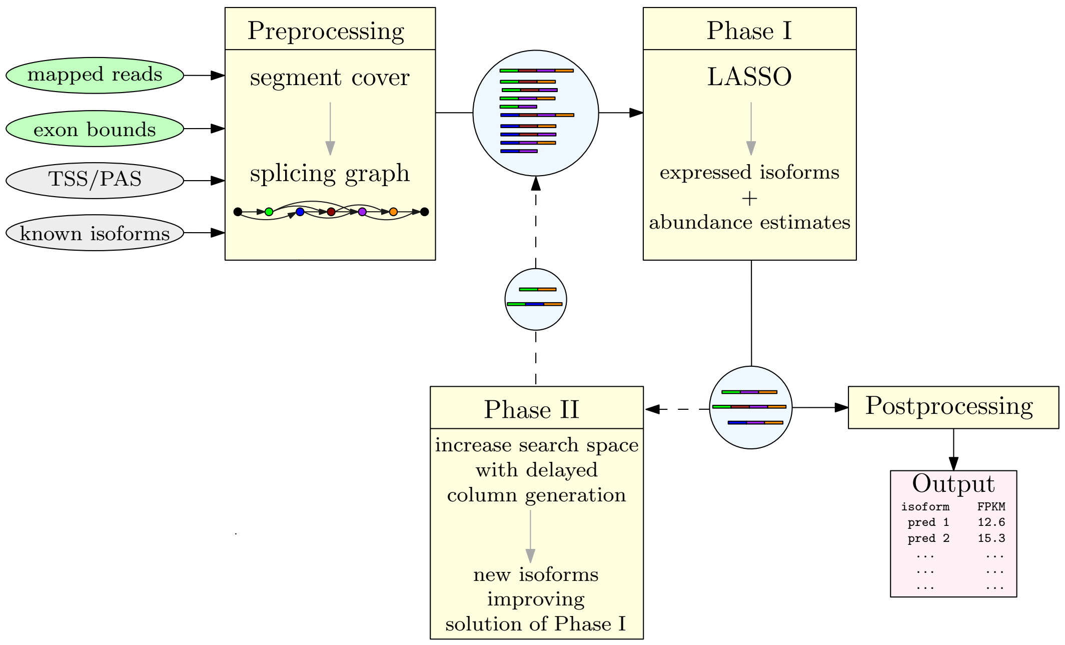


Figure 4.2: *General workflow of* CIDNE. *Mandatory inputs (mapped RNA-Seq reads, exon boundaries) and optional inputs (known transcripts, TSSs and PASs) are used to generate segment covers and a splicing graph. A restricted set of candidate isoforms is generated from the splicing graph and a subset of expressed isoforms with estimated abundances is computed by the LASSO method (Phase I). This set is the input of the optional second step (Phase II), where an extended candidate set is considered using delayed column generation. New candidates inferred in Phase II are then added to the initial candidate set and LASSO is again applied to compute the final solution. After postprocessing, including re-estimation of abundances and filtering, a list of isoforms with abundance estimates is returned.*

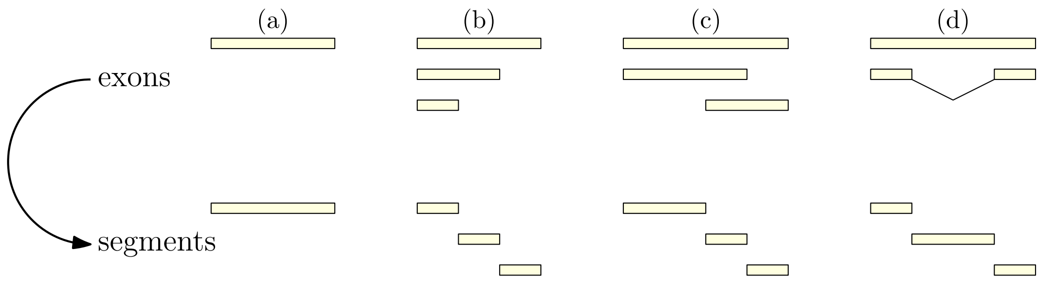


Figure 4.4: *Examples of exons and generated segments. Exons and segments are aligned according to their genomic position. (a) Only a single segment is generated. (b) Top exon has two variants with alternative 3' boundaries, resulting in three segments. (c) Top exon has one variant with alternative 3' boundary and one variant with alternative 5' boundary, also resulting in three segments. (d) Three segments are generated due to an intron retention.*

nucleotide sequence obtained by concatenation of the genomic sequence intervals $s_{t_1}, \ldots, s_{t_m}$.

Segment cover

Given the set of segments S and reads mapped to the genome, we define a *segment cover* $c = (c', c'', b)$ as a triple, where c' and c'' are sequences of segments and *segment cover count* b is the number of mapped paired-end reads *matching* (c', c''). Given a mapped read r' aligned to genomic positions $A(r') = a_1 < \cdots < a_k$, we say r' *matches* a sequence of segments $u = \langle s_{u_1}, \ldots, s_{u_m} \rangle$ if and only if the segments in u contain all aligned positions, each segment at least one, and if all gaps in the alignment are consistent with all segment boundaries in u, i.e.,

$$A(r) \cap s_{u_i} \neq \emptyset, 1 \leq i \leq m \quad \text{and} \quad A(r') = [a_1 \mathinner{..} a_k] \cap \bigcup_{1 \leq i \leq m} s_{u_i}\,.$$

Analogously, we say a mapped paired-end read $r = (r_1, r_2)$ matches (c', c'') if r_1 matches c' and r_2 matches c'' (see Figure 4.5). With a slight abuse of notation, in this case we will also say that r matches segment cover c. To avoid overcounting of multiple mapped reads, every paired-end read matching k different segment covers contributes a value of $1/k$ to each of the corresponding segment cover counts.

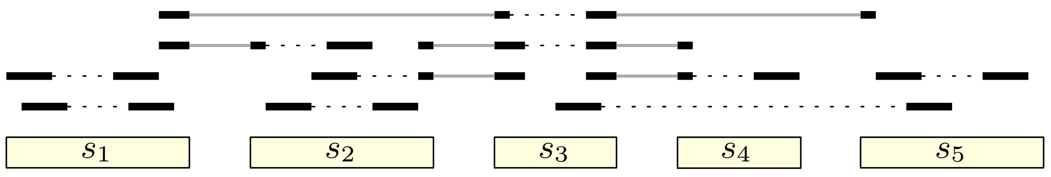


Figure 4.5: *Example of paired-end reads mapped onto segments. Solid lines denote splice junction spanning reads and dotted lines connect paired reads originating from the same fragment. The associated segment covers are given by:* $(\langle s_1\rangle, \langle s_1\rangle, 2)$, $(\langle s_1, s_2\rangle, \langle s_2\rangle, 1)$, $(\langle s_1, s_3\rangle, \langle s_3, s_5\rangle, 1)$, $(\langle s_2\rangle, \langle s_2\rangle, 1)$, $(\langle s_2\rangle, \langle s_2, s_3\rangle, 1)$, $(\langle s_2, s_3\rangle, \langle s_3, s_4\rangle, 1)$, $(\langle s_3, s_4\rangle, \langle s_4\rangle), 1)$, $(\langle s_3\rangle, \langle s_5\rangle, 1)$, *and* $(\langle s_5\rangle, \langle s_5\rangle, 1)$. *The segment covers imply an alternative splicing event with skipping of segment* s_2.

Segment cover length

Given a transcript t, segment cover $c = (c', c'', b)$ and fragment length ℓ^f, we define the *segment cover length* $\ell_{t,c}(\ell^f)$ as the number of possible starting positions for a fragment originating from transcript t, such that the associated mapped paired-end read matches (c', c''). Hence, $\ell_{t,c}(\ell^f)$ denotes the number of possible starting positions for a fragment in t in order to contribute to count b. If t does not contain c' or c'' as substrings, $\ell_{t,c}(\ell^f)$ has obviously value zero. Otherwise, we compute $\ell_{t,c}(\ell^f)$ in two steps: First, for a single read r of length ℓ^r, we calculate the leftmost and rightmost starting positions, *lpos* and *rpos*, of r in the transcript sequence of t such that r matches c' (analog for c''). Let s_l be the first segment in c', referred to as first(c'), and s_k be the last segment in c', referred to as last(c'). Further, let s_l and s_k span positions $[l_1 \,..\, l_2]$ and $[k_1 \,..\, k_2]$ in the transcript sequence of t respectively. Now, we can compute *lpos* and *rpos* as follows (see also Figure 4.6):

$$\begin{aligned} lpos &= \max\{l_1, k_1 - \ell^r + 1\}\,, \\ rpos &= \min\{l_2, k_2 - \ell^r + 1\}\,. \end{aligned}$$

Given the leftmost and rightmost starting positions $(lpos_1, rpos_1)$ for c' and $(lpos_2, rpos_2)$ for c'', we can now compute $\ell_{t,c}(\ell^f)$. We define Δ^{in} and Δ^{out} as follows (see Figure 4.7):

$$\begin{aligned} \Delta^{in} &:= lpos_2 - rpos_1\,, \\ \Delta^{out} &:= rpos_2 - lpos_1\,. \end{aligned}$$

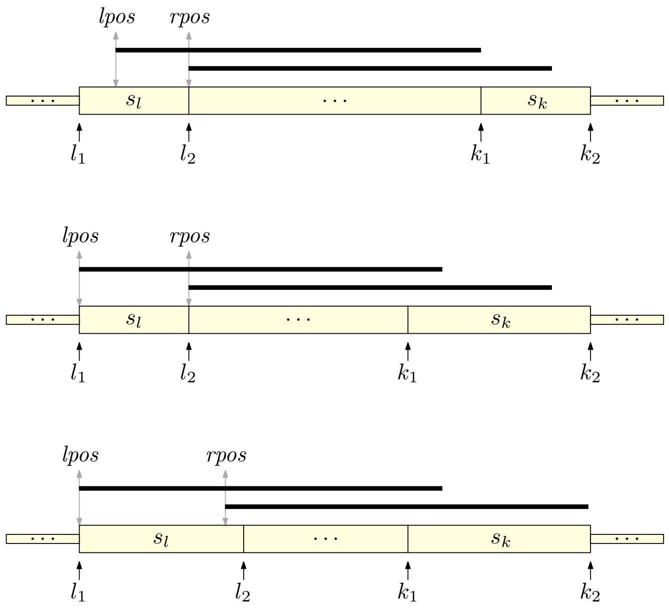


Figure 4.6: *Example scenarios for computation of values lpos and rpos for a given transcript containing a sequence of segments $\langle s_l, \ldots, s_k \rangle$ with segments s_l and s_k spanning positions $[l_1 \,..\, l_2]$ and $[k_1 \,..\, k_2]$ of the transcript sequence.*

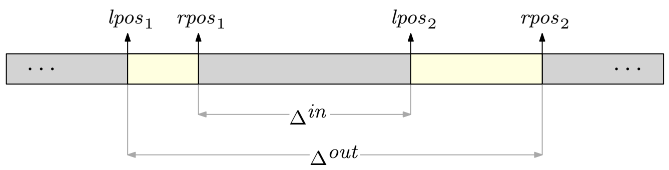


Figure 4.7: *Illustration of values Δ^{in} and Δ^{out} for a segment cover $c = (c', c'', b)$ with first and last starting positions $(lpos_1, rpos_1)$ and $(lpos_2, rpos_2)$ for a read to match c' and c'' respectively.*

Using the identity $\ell^s = \ell^f - 2\ell^r$, we finally obtain

$$\ell_{t,c}(\ell^f) = \max\{0, \min\{rpos_1 - lpos_1,\; rpos_2 - lpos_2,\; \ell^f - \ell^r - \Delta^{\text{in}},\; \Delta^{\text{out}} + \ell^r - \ell^f\} + 1\}\,.$$

Adjusted segment cover length

For experimental sequencing data, the fragment length is not constant but is assumed to follow a certain *fragment length distribution*. Therefore, for a given segment cover c, transcript t and fragment length distribution with probability mass function D, we compute the *adjusted segment cover length* $\bar{\ell}_{t,c}$, as follows:

$$\bar{\ell}_{t,c} := \sum_{f=\ell^f_{lb}}^{\ell^f_{ub}} D(f)\ell_{t,c}(f)\,, \tag{4.1}$$

where ℓ^f_{lb} and ℓ^f_{ub} are the minimum and maximum considered fragment lengths respectively. Hence, $\bar{\ell}_{t,c}$ can be interpreted as the expected segment cover length for the given fragment length distribution.

Splicing graph

For the task of isoform inference we require a compact data structure to represent all candidate isoforms for a single gene. The *splicing graph*, as introduced by Heber et al. [2002], offers such a representation by means of a graph in which every candidate isoform corresponds to a unique path. Given the set of segments S for a single gene, we construct the splicing graph $\mathcal{G}^S = (V, E)$ by generating a vertex $v_i \in V$ for every segment $s_i \in S$ (see Figure 4.5). Further, we add two additional vertices v^a and v^b to represent artificial start and end segments. For our algorithm, we construct the set of directed edges according to the following rules: Given the set C of segment covers, we add a directed edge (v_i, v_j) to E if $i < j$ and at least one of the following conditions is fulfilled:

(a) $s_i = [p_i \mathinner{..} q_i], s_j = [p_j \mathinner{..} q_j]$ and $p_j - q_i = 1$.

(b) $\exists (c', c'', b) \in C : \lceil b \rceil \geq \gamma \wedge (\langle s_i, s_j \rangle \preceq c' \vee \langle s_i, s_j \rangle \preceq c'')$.

(c) $j = i + 1 \wedge \exists (c', c'', b) \in C : \lceil b \rceil \geq \gamma \wedge s_i = \text{last}(c') \wedge s_j = \text{first}(c'')$.

where threshold parameter γ (default: $\gamma = 1$) denotes a lower bound for the number of mapped reads required to infer a splice junction edge in the splicing graph. Case (a) implies an edge between vertices corresponding to segments that are not separated by an intron and hence do not imply a splice junction. Case (b) implies an edge if a sufficient number of reads was mapped across the associated splice junction. Finally, case (c) implies an edge between vertices corresponding to consecutive segments s_i and s_{i+1}, if the associated splice junction is induced by enclosing paired-end reads. In the remainder, we will refer to all splice junctions (s_i, s_j) that fulfill either case (b) or (c) as *covered* splice junctions.

Every valid candidate isoform cannot start or end at arbitrary segments, but instead, it must start at a potential TSS and end at a potential PAS. Therefore, we define two sets $\mathcal{TSS}, \mathcal{PAS} \subseteq S$, containing all segments corresponding to a potential TSS and PAS respectively. We create these sets either based on annotated TSSs and PASs, additional data obtained from experiments like CAGE

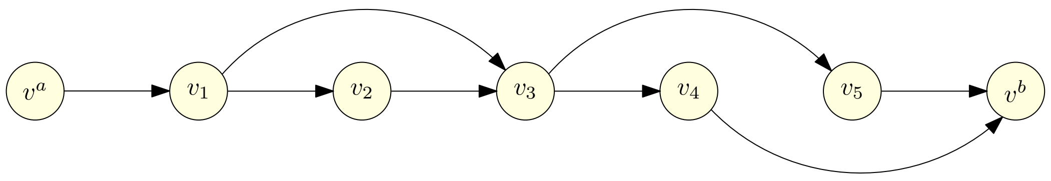


Figure 4.8: *Splicing graph generated for the segment covers shown in Figure 4.5. The graph encodes four transcripts: $t_1 = \langle s_1, s_2, s_3, s_4\rangle$, $t_2 = \langle s_1, s_2, s_3, s_5\rangle$, $t_3 = \langle s_1, s_3, s_4\rangle$ and $t_4 = \langle s_1, s_3, s_5\rangle$. However, the paired end information in the segment counts implies the existence of t_4.*

and *polyadenylation site sequencing*, or *ab initio*, based on the mapped reads. When solely based on mapped reads, we follow the same conservative strategy like CLASS and IsoLasso and consider only maximal candidate isoforms that are not a substring of any other candidate. Hence, we insert a segment s_i into set $\mathcal{TSS}$ (resp. $\mathcal{PAS}$) if vertex v_i has no incoming (resp. outgoing) edge. According to this strategy, isoforms starting or ending within existing exons of expressed isoforms can actually not be detected. However, to overcome this limitation, external methods to identify potential TSSs and PASs based on read mapping coverage [Behr et al., 2013] can be applied prior to our algorithm.

According to the definition of sets $\mathcal{TSS}$ and $\mathcal{PAS}$, for every candidate isoform $t = \langle s_i, \ldots, s_j\rangle$ we require $s_i \in \mathcal{TSS}$ and $s_j \in \mathcal{PAS}$. Hence, we add the following directed edges to the splicing graph:

$$(v^a, v_i), \forall s_i \in \mathcal{TSS} \quad \text{and} \quad (v_i, v^b), \forall s_i \in \mathcal{PAS}.$$

Assuming that for the set of expressed isoforms every splice junction is covered by at least γ mapped reads and all true TSSs and PASs are contained in the sets $\mathcal{TSS}$ and $\mathcal{PAS}$ respectively, then every expressed isoform corresponds to a unique v^a-v^b path in $\mathcal{G}^S$. On the other hand, if some expressed transcript contains an uncovered splice junction, then this transcript cannot be constructed as a v^a-v^b path in $\mathcal{G}^S$. From this it follows that we can generate the set of all maximal candidate isoforms that contain only covered splice junctions by enumerating all v^a-v^b paths in $\mathcal{G}^S$.

Faux segment cover

In addition to segment covers generated from observed read mappings, we also add *faux segment covers* to the model, which correspond to segment combinations not matched by any mapped paired-end read. These additional segment covers provide valuable information that can help to discriminate between candidates corresponding to true expressed isoforms and potential false positives. In our current implementation, we generate a faux segment cover $(c', c'', 0)$ if the following three conditions are fulfilled:

1. $\forall(c'_i, c''_i, b_i) \in C : c'_i \neq c' \vee c''_i \neq c''$.
2. $|c'| \leq 2 \wedge |c''| \leq 2$.
3. $\mathcal{G}^S$ contains a path $\langle v_i, \ldots, v_j \rangle$ with $s_i = \text{first}(c')$ and $s_j = \text{last}(c'')$.

Using above definitions, in the following two sections we present our algorithmic approach to the isoform inference and abundance estimation problem.

4.3 Core mathematical model

Given a set of candidate isoforms T and segment covers C, the central idea of our isoform inference algorithm is to compute an assignment of non-negative abundance estimates to all candidate isoforms in the set T that yields the best explanation of the segment cover counts.

For a given RNA molecule of length n, we refer to the nucleotides at positions $[1\,..\,n-\ell+1]$ as *potential starting points* for a sequenced fragment of length ℓ. Considering the complete set of RNA molecules in the analyzed sample, we assume that every potential starting point in this set has the same probability to be selected as starting point of a sequenced fragment. Based on this sampling uniformity assumption, the expected number of sequenced fragments originating from a particular isoform t and matching a given segment cover c is proportional to $\bar{\ell}_{t,c}$ and the abundance of t.

Our model quantifies isoform abundance in *expected fragments per base* (*FPB*), which is defined as the expected number of mapped fragments per base of transcript sequence. Hence, given the estimated abundance f_t of isoform t, the expected number of mapped fragments originating from t that match segment cover c is given by $f_t\bar{\ell}_{t,c}$. We denote by $supp(T, c)$ the subset of candidates in T that *support* segment cover c, i.e., $supp(T, c) := \{t \in T \mid \bar{\ell}_{t,c} > 0\}$. Finally, the expected number of fragments matching segment cover c is given by

$$\sum_{t \in supp(T,c)} f_t\bar{\ell}_{t,c}\,.$$

4.3.1 Linear least squares formulation

The core of our algorithmic approach to infer expressed isoforms and estimate their abundance is a linear least squares formulation, with segment cover counts as *observations* and isoform abundances as *unknown parameters* to be estimated. Accordingly, the problem has the following form:

$$\min_{f_t:t\in T} \sum_{\substack{c_i=(c'_i,c''_i,b_i):\\ c_i\in C}} \Big(b_i - \sum_{t\in supp(T,c_i)} f_t\bar{\ell}_{t,c_i}\Big)^2, \quad f_t \geq 0 \;\forall t\in T\,.$$

To account for significant differences in segment cover lengths and associated possible differences in the reliability of observed segment cover counts, we use a weighted least squares formulation, similar to IsoInfer [Feng et al., 2010]. For linear regression with unequal variances, the *best linear unbiased estimator* is obtained if the weight for each observation is reciprocal to its variance. As proposed by Feng et al., we use the observed segment cover count (b_i) as estimator for the variance of observation b_i. Since the term $(1/b_i)$ is undefined for faux covers, we use a lower bound of ϵ (default: $\epsilon = 1$) for the variance estimate. Hence, in our formulation we aim at minimizing the following term:

$$\sum_{\substack{c_i=(c'_i,c''_i,b_i):\\ c_i\in C}} \left(\frac{b_i - \sum_{t\in supp(T,c_i)} f_t\bar{\ell}_{t,c_i}}{\sqrt{\max\{\epsilon, b_i\}}}\right)^2, \quad f_t \geq 0 \;\forall t\in T\,. \tag{4.2}$$

4.3.2 Regularized linear least squares formulation

Depending on the structure of the splicing graph, the number of candidate isoforms can be very large compared to the number of segment covers. This can lead to unidentifiable instances without unique optimal solution or to overfitting by prediction of many isoforms with non-zero estimated abundances. Therefore, similar to IsoLasso, SLIDE, iReckon, and MiTie, we apply regularization to balance between the accuracy of predicted segment cover counts and the number of isoforms with non-zero estimated abundance. While the most desirable approach would be the application of a L_0-regularization, which directly penalizes the number of isoforms with non-zero estimated abundance, its non-convexity renders the associated optimization problem intractable for complex genes with many of candidate isoforms. Therefore, we adopt the approach used by IsoLasso and SLIDE and apply the LASSO method [Tibshirani, 1996], which performs L_1-regularization by adding the sum of estimated abundances to the objective function as a penalty. Finally, our core optimization problem is given by

$$\min_{f_t:t\in T} \sum_{\substack{c_i=(c'_i,c''_i,b_i):\\ c_i\in C}} \left(\frac{b_i - \sum_{t\in supp(T,c_i)} f_t\bar{\ell}_{t,c_i}}{\sqrt{\max\{\epsilon, b_i\}}}\right)^2 + \lambda\sum_{t\in T} f_t\,, \quad f_t \geq 0 \;\forall t\in T\,, \tag{4.3}$$

with regularization parameter λ to control the sparseness of the solution. Obviously, the isoform inference accuracy depends crucially on a good choice of the regularization parameter λ, as too small choices of λ will likely cause many false positive predicted isoforms. On the other hand, a too large choice of λ will drive many abundance estimates towards zero and generate very sparse solutions, possibly excluding correct isoforms. In Section 4.5 we briefly discuss the model selection strategy used in our implementation.

4.4 A delayed column generation approach

To enhance the sensitivity of our algorithm, we add an additional step to recover isoforms with uncovered splice junctions that cannot be included into the candidate set of the regularized least squares formulation due to their potentially very large number. Therefore, we apply delayed column generation to identify new candidate isoforms that improve the optimal solution of the regularized least squares problem without exhaustive enumeration of all possible candidates. As the delayed column generation method is limited to linear programming, we first formulate a piecewise linear approximation of the quadratic least squares objective function, as outlined in Section 4.4.1. The generation of new isoforms is then accomplished by means of an ILP formulation presented in Section 4.4.2.

4.4.1 Piecewise linear approximation of regularized linear least squares formulation

Before we can apply the delayed column generation method, we must formulate a linear program that is *almost* equivalent to the quadratic optimization problem (4.3). First, we rewrite (4.3) by introducing a slack error variable e_i for every segment cover $c_i \in C$, leading to the following quadratic problem:

$$\begin{aligned}
\min \quad & \sum_{(c'_i,c''_i,b_i)\in C} \left(\frac{e_i}{\sqrt{\max\{\epsilon, b_i\}}}\right)^2 + \lambda \sum_{t\in T} f_t & & \\
\text{subject to} \quad & \sum_{t\in supp(T,c_i)} f_t \bar{\ell}_{t,c_i} + e_i = b_i & & \forall c_i = (c'_i, c''_i, b_i) : c_i \in C \\
& f_t \in \mathbb{R}_+ & & \forall t \in T \\
& e_i \in \mathbb{R} & & \forall c_i \in C
\end{aligned} \tag{4.4}$$

Next, we approximate the quadratic objective function as follows: For every segment cover c_i, we introduce a variable $\tilde{e}_i$ to act as a proxy for the quadratic error e_i^2. Further, we define a set of *supporting points* $p_1^i, \ldots, p_k^i \in \mathbb{R}$. For every supporting point p, we compute the gradient of the approximated function, i.e., $g(x) = x^2$, and approximate $g(x)$ by the tangent line of g at point p, defined as

$$g^p(x) = g(p) + g'(p)(x - p) = p^2 + 2p(x - p)\,.$$

Due to the convexity of the approximated quadratic function, each of the linear approximations $g^{p^i_1}(x), \dots, g^{p^i_k}(x)$ constitutes a lower bound to $g(x)$. Hence, we select the tightest bound for proxy variable $\tilde{e}_i$ by enforcing $\tilde{e}_i \geq \max_{1\leq j\leq k} g^{p^i_j}(e_i)$. This leads to the following linear programming approximation of (4.3):

$$\begin{array}{llll}
\min & \displaystyle\sum_{(c'_i,c''_i,b_i)\in C} \left(\frac{\tilde{e}_i}{\max\{\epsilon, b_i\}}\right) + \lambda \sum_{t\in T} f_t & & \\
\text{subject to} & \displaystyle\sum_{t\in supp(T,c_i)} f_t \bar{\ell}_{t,c_i} + e_i = b_i & \forall c_i = (c'_i, c''_i, b_i) : c_i \in C & \\
 & 2p^i_j e_i - \tilde{e}_i \leq (p^i_j)^2 & 1 \leq i \leq |C|, 1 \leq j \leq k & \\
 & f_t \in \mathbb{R}_+ & \forall t \in T & (4.5)\\
 & e_i \in \mathbb{R} & \forall c_i \in C & \\
 & \tilde{e}_i \in \mathbb{R}_+ & \forall c_i \in C &
\end{array}$$

The approximation error of $g^p(x)$ is given by $(x - p)^2$. From this it follows that, for a predefined interval $[a_l, a_r]$ of values for $(e_i/\sqrt{\max\{\epsilon, b_i\}})$, we can approximate the term $(e_i/\sqrt{\max\{\epsilon, b_i\}})^2$ with a maximum error of $\mu > 0$ by defining $k = \lceil (a_r - a_l)/(2\sqrt{\mu \max\{\epsilon, b_i\}}) \rceil$ supporting points $p_1, \dots, p_k$, with $p_j = a_l + (2j - 1)\sqrt{\mu \max\{\epsilon, b_i\}}$

4.4.2 Pricing ILP for piecewise linear approximation

Given a solution of the piecewise linear approximation (4.5) with associated vector of Simplex multipliers $\boldsymbol{\rho}$, the pricing problem consists in the identification of a candidate isoform t with negative reduced cost $\bar{c}_t = \lambda - \boldsymbol{\rho}^\top \mathbf{A}_t$, where $\mathbf{A}_t$ is the column vector containing the coefficients of variable f_t for all constraints of formulation (4.5). Let the constraint matrix of (4.5) be ordered in a way that the first $|C|$ rows correspond to the equality constraints for all segment covers $c_1, \dots, c_{|C|}$. It follows that the i-th element of $\mathbf{A}_t$, with $i \leq |C|$, has value $\bar{\ell}_{t,c_i}$, if $t \in supp(T, c_i)$ and zero otherwise. All remaining elements of $\mathbf{A}_t$ are zero. Hence, to solve the pricing problem given the simplex multipliers $\rho_1, \dots, \rho_{|C|}$ for the first $|C|$ constraints, we must either identify a candidate isoform t with the following property:

$$\sum_{\substack{c_i\in C:\\ t\in supp(T,c_i)}} \rho_i \bar{\ell}_{t,c_i} > \lambda\,,$$

or prove that no such candidate exists. We will solve this problem by means of an ILP formulation to compute

$$\max_{t \in T} \sum_{\substack{c_i \in C: \\ t \in supp(T, c_i)}} \rho_i \bar{\ell}_{t,c_i} \,,$$

without exhaustively enumerating the complete set T. If the optimal cost of this pricing ILP is at most equal to λ, then it follows that the current solution of LP (4.5) is optimal and we can terminate.

Graph-theoretical formulation

Before we present our ILP formulation for the pricing problem, we reformulate it as a graph-theoretical problem based on a hypergraph $\mathcal{H}^S = (V^H, E^H)$. Like the splicing graph, $\mathcal{H}^S$ contains a vertex $v_i \in V^H$ for every segment $s_i \in S$. Further, $\mathcal{H}^S$ contains a hyperedge $e_i \in E^H$ for every segment cover $(c_i', c_i'', b_i) \in C$. Every hyperedge $e_i \in E^H$ contains the vertices for all segments in c_i' and c_i'', i.e., $e_i = \{v_j \in V^H \mid s_j \in c_i' \vee s_j \in c_i''\}$.

In order to introduce edge weights for $\mathcal{H}^S$, we first define $\bar{V}(e_i)$ as the set of vertices whose corresponding segments are located between the last segment in c_i' and the first segment in c_i'', i.e.:

$$\bar{V}(e_i) := \{v_k \in V^H \mid \text{last}(c_i') < s_k < \text{first}(c_i'')\} \,.$$

Using this definition, we assign a weight function $\omega_{e_i} : \mathcal{P}(\bar{V}(e_i)) \mapsto \mathbb{R}$ to each edge $e_i \in E^H$. For a set of vertices $\bar{V}' \subseteq \bar{V}(e_i)$, the associated weight $\omega_{e_i}(\bar{V}')$ is given by $\rho_i \bar{\ell}_{p,c_i}$ with $p = c' \cdot \alpha \cdot c''$, where α defines the sequence of segments corresponding to the vertices in $\bar{V}'$. Further, we define $\xi(e_i)$ as the set of vertices with associated segments being spanned by either c_i' or c_i'', i.e.:

$$\xi(e_i) := \{v_k \in V^H \mid \text{first}(c_i') \leq s_k \leq \text{last}(c_i') \vee \text{first}(c_i'') \leq s_k \leq \text{last}(c_i'')\} \,.$$

According to the construction of E^H, we can now formulate the pricing problem as a variant of the *maximum weight induced subgraph problem*, which consists in identifying a subset $V' \subseteq V^H$ that maximizes the total weight of all edges *strictly induced* by V'. We define the set $E^{ind}(V', E^H)$ of strictly induced edges by some subset V' as follows:

$$E^{ind}(V', E^H) := \{e \in E^H \mid \forall v \in e : v \in V' \wedge \forall v \in \xi(e) \setminus e : v \notin V'\} \,.$$

Hence, edge $e_i \in E^H$ is strictly induced by V' if the sequence of segments corresponding to all vertices in V' contains c_i' and c_i'' as substrings. Finally, the weight of selected subset V' is given by

$$\sum_{e \in E^{ind}(V', E^H)} \omega_e(V' \cap \bar{V}(e)) \,.$$

We capture this problem by the following ILP formulation: For every vertex $v_i \in V^H$, we introduce a binary variable x_i, which has value one if and only if vertex v_i is contained in the selected subset V'. In addition, for every pair of edge $e \in E^H$ and subset of vertices $\bar{V}'_j \subseteq \bar{V}(e)$, we introduce a binary variable $y_{e,j}$, which has value one if and only if the set of selected vertices contains every vertex in $\bar{V}'_j$ (constraint (4.9)) and strictly induces edge e (constraints (4.8) and (4.10)). The associated weight $\omega_{e,j}$ is given by $\omega_e(\bar{V}'_j)$.

$$\max \sum_{e \in E^H, \bar{V}'_j \subseteq \bar{V}(e)} \omega_{e,j} y_{e,j} \tag{4.6}$$

$$\text{subject to} \sum_{\bar{V}'_j \subseteq \bar{V}(e)} y_{e,j} \leq 1 \qquad \forall e \in E^H \tag{4.7}$$

$$y_{e,j} \geq \sum_{v_i \in e \cup \bar{V}'_j} x_i + \sum_{v_i \in \bar{V}(e) \setminus \bar{V}'_j} (1 - x_i) + \sum_{v_i \in \xi(e) \setminus e} (1 - x_i) + \\ - |\bar{V}(e)| - |\xi(e)| + 1 \qquad \forall e \in E^H, \bar{V}'_j \subseteq \bar{V}(e) \tag{4.8}$$

$$y_{e,j} \leq x_i \qquad \forall e \in E^H, \bar{V}'_j \subseteq \bar{V}(e), v_i \in e \cup \bar{V}'_j \tag{4.9}$$

$$y_{e,j} \leq 1 - x_i \qquad \forall e \in E^H, \bar{V}'_j \subseteq \bar{V}(e), \; v_i \in \{(\xi(e) \setminus e) \cup (\bar{V}(e) \setminus \bar{V}'_j)\} \tag{4.10}$$

$$x_i \in \{0, 1\} \qquad \forall v_i \in V^H \tag{4.11}$$

$$y_{e,j} \in \{0, 1\} \qquad \forall e \in E^H, \bar{V}'_j \subseteq \bar{V}(e) \tag{4.12}$$

Exon compatibility

To ensure that the set of selected segments induces only exons that are also present in the input set S', we add a binary variable z_j for every exon $s'_j \in S'$ to indicate whether exon s'_j is induced by the x-variables. We connect segment variables to exon variables by means of constraint (4.13).

$$x_i = \sum_{s'_j \supseteq s_i} z_j \qquad 1 \leq i \leq |S| \tag{4.13}$$

According to this constraint, every solution fulfills the following three properties: First, whenever some segment s_i is selected, exactly one exon s'_j containing segment s_i (i.e., $s_i \subseteq s'_j$) is also selected. Second, whenever some exon s'_j is selected, all contained segments are also selected. Finally, from every pair of overlapping, hence incompatible exons, at most one is selected, which avoids the creation of novel, expanded exons. Taken together, these three properties ensure that every solution corresponds to an isoform containing only exons from the input set S'.

TSS and PAS compatibility

As the selected set of segments is supposed to form a valid isoform, we must ensure that it starts at some segment $s_i \in \mathcal{TSS}$ and ends at some segment $s_j \in \mathcal{PAS}$. Therefore, we introduce binary variables ss_i and es_i to indicate whether segment s_i is selected as start or end segment of the generated isoform. Using these variables, we formulate constraints (4.14)-(4.19) to ensure that every solution selects exactly one segment from set $\mathcal{TSS}$ as start segment and exactly one segment from set $\mathcal{PAS}$ as end segment. Finally, we use constraints (4.20) and (4.21) to ensure that no selected segment precedes the selected start segment or follows the selected end segment.

$$\sum_{v_i \in V^H} ss_i = 1 \tag{4.14}$$

$$\sum_{v_i \in V^H} es_i = 1 \tag{4.15}$$

$$ss_i = 0 \qquad \forall s_i \notin \mathcal{TSS} \tag{4.16}$$

$$es_i = 0 \qquad \forall s_i \notin \mathcal{PAS} \tag{4.17}$$

$$x_i \geq ss_i \qquad \forall v_i \in V^H \tag{4.18}$$

$$x_i \geq es_i \qquad \forall v_i \in V^H \tag{4.19}$$

$$x_i \leq 1 - \sum_{j=i+1}^{|V|} ss_j \qquad \forall v_i \in V^H \tag{4.20}$$

$$x_i \leq 1 - \sum_{j=1}^{i-1} es_j \qquad \forall v_i \in V^H \tag{4.21}$$

Enforcing novel explanations

In order to prevent the inference of new isoforms that are only used to balance coverage fluctuations, we require that new isoforms can explain segment covers that are not explained by any transcript in the restricted set T' considered in the first step (Phase I). We denote this set of initially unexplained segment covers by $\tilde{C} := \{c \in C \mid supp(T', c) = \emptyset\}$. To reduce the impact of spurious read mappings, we also enforce a lower bound k^c on the total count of newly explained segment covers. Hence, we add the following constraint to the pricing ILP:

$$\sum_{(c_i', c_i'', b_i) \in \tilde{C}} b_i \sum_{\bar{V}_j' \subseteq \bar{V}(e_i)} y_{e_i,j} \geq k^c \tag{4.22}$$

Splicing graph compatibility

Finally, we add constraints to restrict the number of uncovered splice junctions contained in the predicted isoform to a given number k^e. For every pair of vertices $v_i, v_j \in V^H$ with $i < j$, we add a binary variable $u_{i,j}$, which has value one if the selected vertices induce an edge (v_i, v_j) in the splicing graph $\mathcal{G}^S = (V, E)$. Let E^{new} be a set of potential *new* edges, that are not contained in $\mathcal{G}^S$, i.e., $E^{\text{new}} \subseteq V \times V$ and $E^{\text{new}} \cap E = \emptyset$. We add constraints (4.23) and (4.24) to the pricing ILP to allow for the simultaneous selection of up to k^e edges in E^{new}. In addition, we add constraint (4.25) to ensure that every edge implied by the selected vertices is either contained in the splicing graph or belongs to the set E^{new}.

$$x_i + x_j - \sum_{i<k<j} x_k - u_{i,j} \leq 1 \qquad 1 \leq i < j \leq |S|, (v_i, v_j) \notin E \tag{4.23}$$

$$\sum_{(i,j) \in E^{\text{new}}} u_{i,j} \leq k^e \tag{4.24}$$

$$u_{i,j} = 0 \qquad 1 \leq i < j \leq |S|, (v_i, v_j) \notin E \cup E^{\text{new}} \tag{4.25}$$

$$u_{i,j} \in \{0, 1\} \qquad 1 \leq i < j \leq |S|, (v_i, v_j) \notin E \tag{4.26}$$

The idea behind constraint (4.23) is that whenever two vertices v_i and v_j with $i < j$ are selected and not connected by an edge in $\mathcal{G}^S$, then either a vertex v_k with $i < k < j$ is also selected or a new edge $(v_i, v_j) \in E^{\text{new}}$ is induced.

Space of y-variables

For every segment cover $c = (c', c'', b) \in C$, the pricing ILP contains a set of y-variables, each corresponding to a sequence of segments $p = c' \cdot \alpha \cdot c''$, with possibly empty substring α. Obviously, the worst case scenario for a gene with n segments occurs if c spans the complete gene, i.e., $\text{last}(c') = s_1$ and $\text{first}(c'') = s_n$. This scenario could give rise to the generation of up to 2^{n-2} y-variables, rendering the delayed column generation approach impractical for genes containing many segments. We avoid the generation of the complete set by adding a y-variable only if the adjusted segment cover length $\bar{\ell}_{p,c}$ for the associated sequence of segments p exceeds some predefined threshold. This strategy is equivalent to setting all coefficients for isoform variables in the complete constraint matrix of (4.5) that are below this threshold to zero. However, we need to apply some filtering strategy so that we can avoid the computation of $\bar{\ell}_{p,c}$ for each of the up to $2^{|n|-2}$ candidate substrings α to decide whether the corresponding y-variable must be added to the ILP.

According to the definition of the adjusted segment cover length (4.1), we only have to consider y-variables with

$$\max_{\ell^f_{lb} \leq f \leq \ell^f_{ub}} \{\ell_{p,c}(f)\} > 0\,.$$

Therefore, we require

$$\max_{\ell^f_{lb} \leq \ell \leq \ell^f_{ub}} \{\Delta^{out} + \ell^r - \ell + 1\} = \Delta^{out} + \ell^r - \ell^f_{lb} + 1 > 0\,,$$

and

$$\max_{\ell^f_{lb} \leq \ell \leq \ell^f_{ub}} \{\ell - \ell^r - \Delta^{in} + 1\} = \ell^f_{ub} - \ell^r - \Delta^{in} + 1 > 0\,.$$

Considering sequence p, assume that c' spans positions $[l_1 \,..\, l_2]$ of the associated transcript sequence with left- and rightmost starting positions $(lpos_1, rpos_1)$ and c'' spans positions $[k_1 \,..\, k_2]$ with left- and rightmost starting positions $(lpos_2, rpos_2)$. We define the terms $\Delta^l_1, \Delta^l_2, \Delta^k_1$ and Δ^k_2 as follows:

$$\Delta^l_1 := l_2 - lpos_1, \quad \Delta^l_2 := l_2 - rpos_1, \quad \Delta^k_1 := lpos_2 - k_1 \quad \text{and} \quad \Delta^k_2 := rpos_2 - k_1.$$

Note that these values are independent of substring α and can be computed solely from c' and c''. Let ℓ^α denote the transcript sequence length of α. Since

$$\Delta^{out} = \Delta^l_1 + \ell^\alpha + \Delta^k_2\,,$$

and

$$\Delta^{in} = \Delta^l_2 + \ell^\alpha + \Delta^k_1\,,$$

we only consider substrings α with

$$\ell^\alpha > \ell^f_{lb} - \ell^r - 1 - \Delta^l_1 - \Delta^k_2\,,$$

and

$$\ell^\alpha < \ell^f_{ub} - \ell^r - \Delta^l_2 - \Delta^k_1 + 1\,,$$

which we generate by means of a recursive traversal in $\mathcal{G}^S$ combined with pruning. We further restrict the generation of y-variables to sequences of segments that induce at most k^y new edges, which must originate from the set E^{new}.

4.5 Model selection

We solve the regularized least squares problem by means of an efficient *pathwise coordinate descent* algorithm proposed by Friedman et al. [2007]. This algorithm solves the least squares problem for a sequence of decreasing values of the regularization parameter λ (regularization path), where it exploits solutions for previous λ as warm-starts. For more details please refer to Friedman et al. [2007].

After the solutions for the complete path of values for λ have been computed, we select one with maximum *adjusted coefficient of determination* $\bar{R}^2$, defined as

$$\bar{R}^2 = R^2 - (1 - R^2)\frac{|T^*|}{|C| - |T^*| - 1},$$

where T^* is the set of candidate isoforms with non-zero estimated abundance and R^2 is the *coefficient of determination*, calculated as follows:

$$R^2 = 1 - \frac{\sum_{(c'_i, c''_i, b_i) \in C} (b_i - \hat{b}_i)^2}{\sum_{(c'_i, c''_i, b_i) \in C} (b_i - \bar{b})^2}, \qquad \bar{b} = \frac{1}{|C|} \sum_{(c'_i, c''_i, b_i) \in C} b_i\,,$$

with $\hat{b}_i$ denoting the predicted count for segment cover c_i according to the estimated isoform abundances.

4.6 Post-processing

Given the final set of inferred isoforms $T^* = t_1^*, \ldots, t_m^*$ with non-zero predicted abundances $f_{t_1^*}, \ldots, f_{t_m^*}$, we perform the following post processing steps: First, we re-estimate abundances by solving the un-regularized least squares problem for the set T^* in order to remove side effects caused by the L_1 penalty. Afterwards, we use the re-estimated abundances $f'_{t_1^*}, \ldots, f'_{t_m^*}$ as weights to compute a final assignment of mapped reads to isoforms using the following formula:

$$r(t_j^*) = \sum_{\substack{c_i = (c'_i, c''_i, b_i): \\ t_j^* \in supp(T^*, c_i)}} b_i \cdot \frac{\bar{\ell}_{t_j^*, c_i} f'_{t_j^*}}{\sum_{t_k^* \in supp(T^*, c_i)} \bar{\ell}_{t_k^*, c_i} f'_{t_k^*}},$$

where $r(t_j^*)$ is the number of reads assigned to isoform t_j^*. This assignment of reads to isoforms corrects overestimation or underestimation of the total number of mapped reads for the whole gene due to non-uniform read mapping coverage. For all isoforms $t_j^* \in T^*$ with $r(t_j^*) \geq k^r$ (default: $k^r = 10$), we convert $r(t_j^*)$ into unit FPKM (Fragments Per Kilobase of transcript per Million mapped fragments) and finally return all isoforms whose estimated abundance in FPKM is at least k^p-percent (default: $k^p = 10$) of the maximal estimated abundance for the same gene.

4.7 Experimental results

We implemented CIDNE in C++ using CPLEX (version 12.4) [IBM, 2011] as solver for the pricing ILP and the piecewise linear approximation LP. In addition, we used the `glmnet` software by Friedman et al. [2010] for efficient model selection as outlined in Section 4.5 and the SeqAn library [Döring et al., 2008] for efficient handling of read mapping and genome annotation data.

We compared the performance of our implementation to state-of-the-art tools in terms of recall, precision, and abundance estimation on transcript level with respect to a reference set, referred to as *ground truth*, containing the set of expressed (*true*) transcripts. In the absence of gold standard sets for RNA-Seq libraries and annotated expressed transcripts, we performed the analysis on realistic simulated RNA-Seq datasets, which we generated by means of the FluxSimulator software [Griebel et al., 2012], which has already been used for benchmarking in numerous studies. We scored a *true* transcript as *recovered* if the set of induced exon-intron boundaries exactly matches those of a predicted transcript. A *true* single-exon transcript was scored as recovered if it overlapped with a predicted single-exon transcript. Every predicted transcript was matched to at most one *true* transcript and vice versa. We used the following standard definitions for recall, precision and F-score:

$$\text{recall} := \frac{\#\text{recovered transcripts}}{\#\text{true transcripts}}, \qquad \text{precision} := \frac{\#\text{recovered transcripts}}{\#\text{predicted transcripts}},$$

$$\text{F-score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Whole genome study

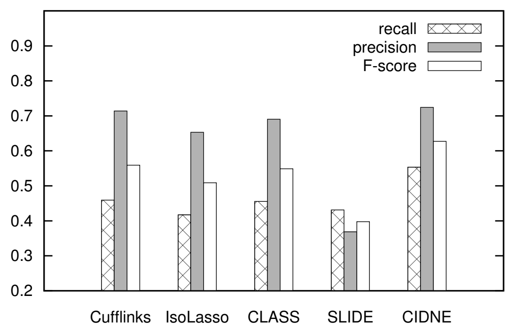
For our first benchmark, we simulated a whole genome scale RNA-Seq experiment, using the FluxSimulator software and the human UCSC-known-gene annotation, containing ∼78.000 known transcripts (Feb. (GRCh37/hg19)). After randomly assigning expression levels to all transcripts in the annotation, following a realistic distribution, the tool simulates individual stages of an RNA-Seq experiment to produce a certain number of single-end or paired-end reads. We simulated experiments with 40 million paired-end reads of length 75 bp and 100 bp without sequencing errors, GC-bias, and poly(A)-tails. Fragment lengths were simulated according to a normal distribution $N(250, 25)$ for the 75 bp reads and $N(300, 30)$ for the 100 bp reads respectively. For each set of paired-end reads, we used TopHat [Trapnell et al., 2009] to map the reads to the set of known transcripts, allowing no errors in the alignment. The ground truth contained the set of all known transcripts for which at least one paired-end read has been produced.

Similar to SLIDE, the current implementation of CIDNE is designed to work with provided exon boundary information and does not infer exon boundaries from the read mapping data. As current versions of Cufflinks[3] and CLASS[4] cannot be supplied with known exon boundary data (unless providing transcript annotations), we ran both tools without any input but the mapped reads, while we provided IsoLasso[5], SLIDE[6] and CIDNE with the set of exon boundaries for all known transcripts. Since the current implementations of CIDNE and IsoLasso do not estimate the fragment length distribution parameters and SLIDE did not compute reasonable estimates, we used the parameters estimated by Cufflinks for all three tools. We did not include iReckon into this first benchmark as it requires all known TSSs and PASs, which, as we believe, would be an unfair advantage.

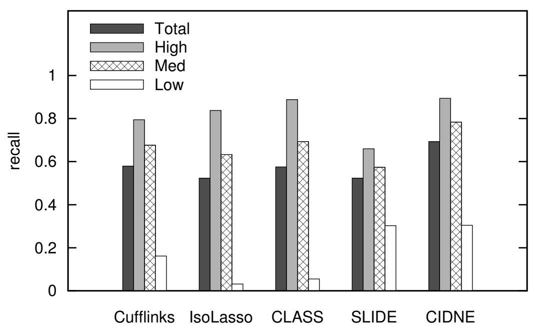
Figure 4.9 displays the recall, precision and F-score for each tool. For the 75 bp dataset, CIDNE achieved the highest recall value (∼55.3%), which was ∼10 percentage points higher than for Cufflinks (∼45.9%) and CLASS (∼45.5%), and more than 12 percentage points higher than for IsoLasso (∼41.7%) and SLIDE (∼43.1%). We obtained similar results for the 100 bp dataset, except that the recall achieved by CLASS was ∼3 percentage points higher than for Cufflinks. In terms of precision, the values obtained for CIDNE (∼72.4%), Cufflinks (∼71.4%) and CLASS (∼69.0%) were very similar for the 75 bp dataset, and higher than for IsoLasso (∼65.3%) and especially SLIDE (∼36.9%). Interestingly, for the 100 bp dataset the precision of SLIDE, Cufflinks and CLASS was decreased by ∼9 to ∼5 percentage points compared to the 75 bp dataset, while we observed no notable changes for IsoLasso and CIDNE.

Further, we analyzed the effect of the transcript abundance on the inference performance for each tool. We removed all transcripts with very low simulated abundance (<0.1 FPKM) from the set and split the remaining transcripts into three groups: Low, Med and High, where Low is the 20% fraction with lowest simulated abundance (in FPKM), High is the 5% fraction with the highest simulated abundance, and Med contains the remaining 75%. For each of the three sets we evaluated the recall separately as depicted in Figure 4.9. On the 75 bp dataset, CIDNE and SLIDE recovered almost twice as many lowly expressed isoforms (both ∼30%) as Cufflinks (∼16.1%), whereas CLASS and IsoLasso recovered only ∼5.5% and ∼3.1% respectively. On each of the three subsets CIDNE achieved the highest recall among all four tools. For the 100 bp dataset we obtained similar overall results.

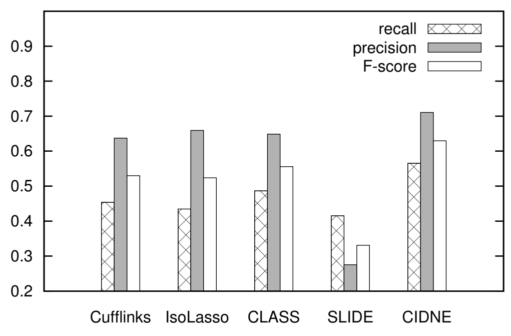
[3] Cufflinks version 2.2.1
[4] CLASS version 1.0.6
[5] IsoLasso version 2.6.1
[6] Adapted python script upon consultation with author.



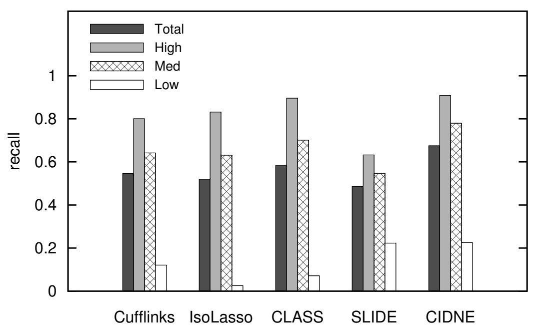
(a) *75 bp*



(b) *75 bp by abundance*



(c) *100 bp*



(d) *100 bp by abundance*

Figure 4.9: *Recall and precision on two simulated RNA-Seq experiments with: (a),(b) 75 bp paired-end reads and (c),(d) 100 bp paired-end reads. Plots (a) and (c) display recall, precision and F-score for the complete set of expressed transcripts. Plots (b) and (d) visualize the recall for all transcripts with simulated FPKM above 0.1* (Total), *which are separated into three groups* (Low, Med, High) *according to their simulated FPKM.*

Benchmark with partial annotation

As the average number of expressed isoforms per gene in the previous benchmark was relatively low (∼1.6), we performed a second simulation study on a selected set of genes containing between 2 and 8 annotated isoforms. In this study we analyzed the ability of CIDNE, Cufflinks and iReckon[7] to infer new isoforms when provided with an annotation of known transcripts, a use case supported by all three tools. This scenario is particularly interesting for practical application of these tools, as for many studied organisms at least a partial annotation of their transcriptome already exists, which can aid the identification process. For every gene considered in this benchmark, we removed at least one and at most

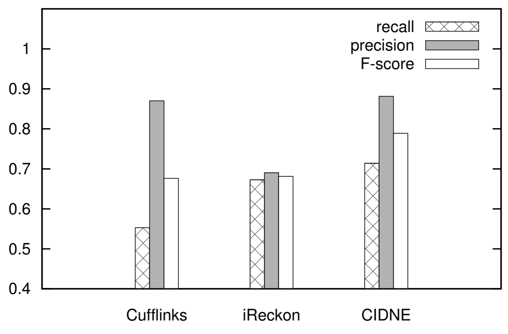
[7] iReckon version 1.0.8

50% of the known isoforms from the annotation, while ensuring that the reduced annotation still contained all exons present in the complete set of known isoforms. Our final benchmark set contained 1440 genes from chromosomes 1 and 2, with a total of $\sim$6300 transcripts of which we included $\sim$65% in the annotation provided to each tool (Annot). With the remaining $\sim$35% of transcripts (Hidden), we evaluated the ability of each tool to infer new isoforms in the presence of an incomplete annotation. Again, we used the FluxSimulator and generated 4 million paired-end 75 bp reads, which we mapped to the $\sim$6300 transcripts using TopHat with the same settings as in the previous benchmark.

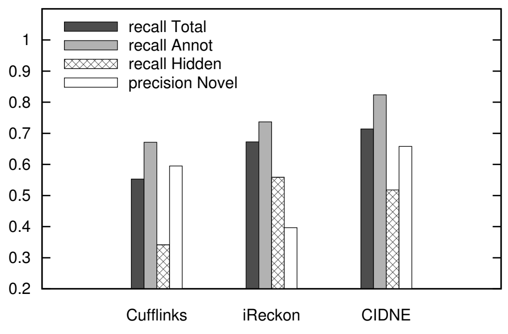
We integrated the incomplete annotation data into the model of CIDNE as follows: First, we integrated the information about known TSSs and PASs. Second, we assigned different regularization weights to annotated isoforms ($\lambda = 1$) and novel isoforms ($\lambda = 2$) in the candidate set. Finally, we only reported novel isoforms having an estimated abundance of at least 20% compared to the highest estimated abundance for the same gene, whereas for annotated isoforms we used the default threshold of 10%.

Cufflinks uses known transcripts to generate so-called *faux-reads* that are combined with the sequencing reads to aid the assembly of novel transcripts. iReckon uses the annotated TSSs and PASs exclusively and further incorporates information about annotated splice junctions.

The recall and precision values for each tool, together with the individual recall values for the two sets Annot and Hidden, are shown in Figure 4.10. Considering the complete set, CIDNE ($\sim$71.4%) achieved the highest recall value, which was $\sim$4 percentage points higher than for iReckon ($\sim$67.3%) and $\sim$16 percentage points higher than for Cufflinks ($\sim$55.3%). In terms of precision, the values for CIDNE ($\sim$88.1%) and Cufflinks ($\sim$87.0%) were 19 (resp. 18) percentage points higher than for iReckon ($\sim$69.0%). Note that since all isoforms in the set Annot had non-zero simulated abundance, the overall precision is determined by the precision of the predicted novel isoforms and the recall for set Annot. Focusing on the ability to correctly predict novel isoforms, CIDNE recovered only slightly less isoforms ($\sim$51.8%) than iReckon ($\sim$55.9%) and considerably more than Cufflinks ($\sim$34.1%). Regarding the precision for novel isoforms, i.e., the fraction of predicted novel isoforms matching an isoform in set Hidden, CIDNE ($\sim$65.7%) achieved a higher value than Cufflinks ($\sim$59.5%) and clearly outperformed iReckon ($\sim$39.6%). Note that we did not observe any significant changes in the performance for each tool for different numbers of simulated paired-end reads (2 million and 8 million).



(a)



(b)

Figure 4.10: *Recall and precision on simulated RNA-Seq experiment with incomplete annotation. (a) Recall, precision, and F-score for the complete set of simulated transcripts. (b) Individual recall for sets* Annot *and* Hidden *and precision of inferred novel transcripts* (precision Novel)*, i.e., fraction of predicted novel transcripts matching a transcript in set* Hidden*.*

Abundance estimation accuracy

In addition to the qualitative analysis in terms of recall and precision, we also analyzed the abundance estimation accuracy of CIDNE. To reduce the effect of the inference performance on the abundance estimation accuracy, we restricted the evaluation to the set Annot. For every transcript in this set, we compared the FPKM value predicted by each tool to the true FPKM value calculated from the number of simulated paired-end reads. For each tool, we treated transcripts not contained in the prediction as being predicted with FPKM value zero. Further, as very short transcripts often cause abundance estimation problems, we limited the analysis to transcripts with a minimal length of at least 500 bp.

We evaluated the correlation between predicted and true abundances in FPKM as shown in Figure 4.12. The calculated Pearson correlation coefficients were very similar for all three tools with values of 0.97 for CIDNE, 0.96 for Cufflinks and 0.98 for iReckon. In addition to the correlation analysis, we evaluated the abundance estimation accuracy in terms of the *relative error* defined as $|f_t - f'_t|/f_t$, where f_t and f'_t denote the true and predicted abundance in FPKM respectively. Note that this term was defined for all transcripts t in Annot as they all had a non-zero true abundance f_t. Considering for each tool the set of transcripts in Annot with non-zero predicted abundance, we calculated the fraction of transcripts having a relative error below a certain threshold, as presented in Figure 4.11(a). While this fraction was very similar for iReckon and CIDNE

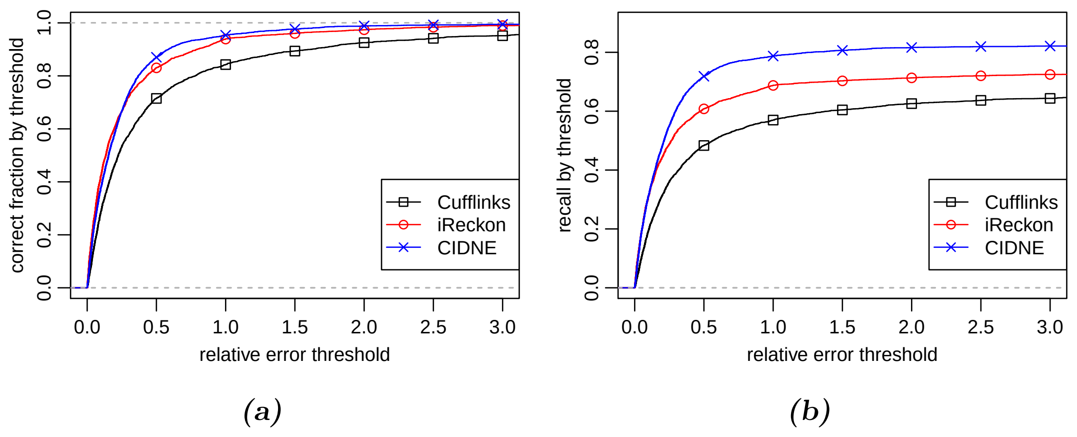
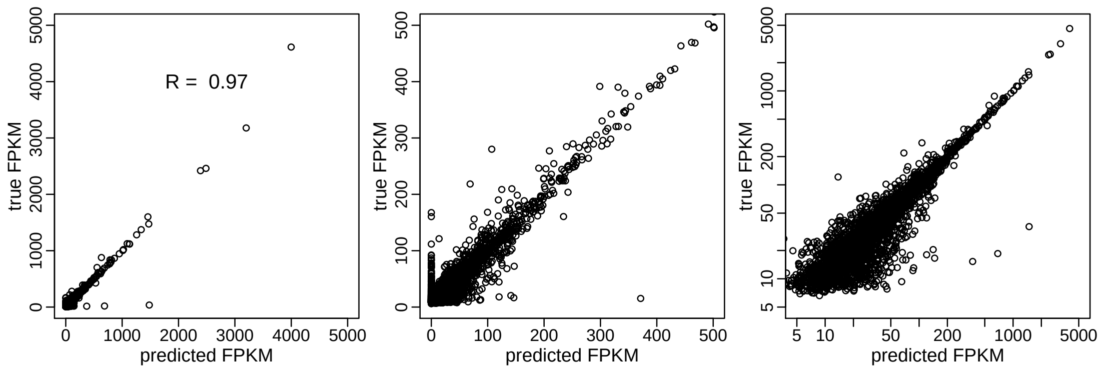
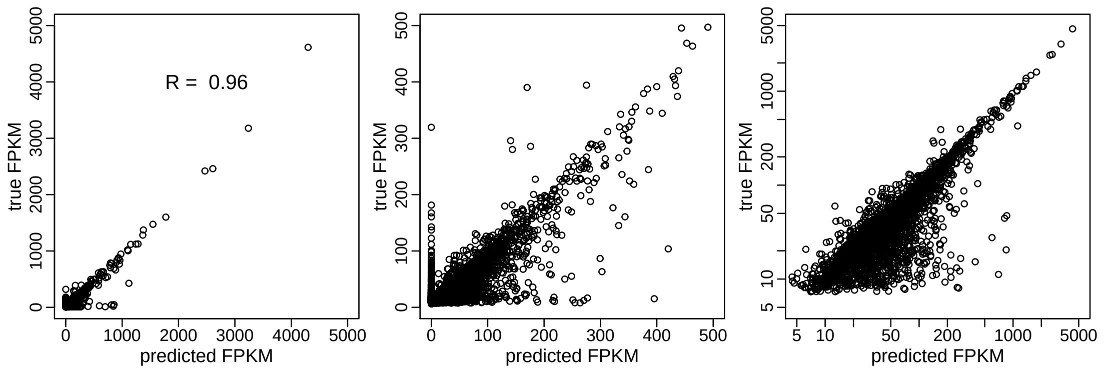


Figure 4.11: *Relative error of predicted transcript abundance in FPKM for* CIDNE, *iReckon, and Cufflinks. (a) Fraction of transcripts with non-zero estimated abundance having a relative error below threshold, for relative error thresholds in range* $[0, 3]$. *(b) Recall achieved when scoring a transcript as recovered only if the relative error of estimated abundance is below threshold, for relative error thresholds in range* $[0, 3]$.

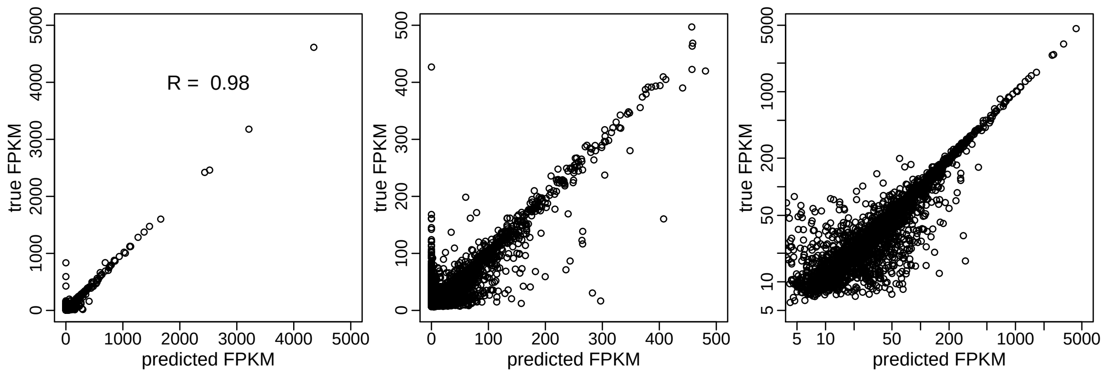
for all considered thresholds, both tools performed considerably better than Cufflinks. Further, we evaluated the recall for the set Annot, when a transcript is scored as recovered only if the relative error is below a certain threshold. As shown in Figure 4.11(b), for all error thresholds iReckon and CIDNE recovered considerably more transcripts than Cufflinks. For small error thresholds (≤ 0.1), iReckon and CIDNE achieved very similar recall values, whereas for increasing thresholds CIDNE recovered more transcripts than iReckon.



(a) CIDNE



(b) Cufflinks



(c) iReckon

Figure 4.12: *Correlation between true and predicted transcript abundance in FPKM for the set Annot. Left figures show plot of true FPKM against predicted FPKM (including predicted FPKM values of zero). Center figures show lower left rectangle of associated left figure with true and predicted FPKM value below 500. Right figures show plot of true FPKM against predicted FPKM in log-scale (excluding transcripts with predicted FPKM value of zero).*

Recovering transcripts with delayed column generation

In the previous two benchmarks, we evaluated the performance of our regularized least squares model on a restricted set of candidate isoforms, without additional delayed column generation step. In this benchmark, we demonstrate the potential of our delayed column generation algorithm to recover isoforms containing uncovered splice junctions. We used the same simulated expression data as in the second benchmark, but this time we simulated only 2 million 75 bp paired-end reads to obtain more expressed transcripts with at least one uncovered splice junction. In the complete set of ∼6300 expressed transcripts, 118 transcripts had at least one uncovered splice junction and could therefore not be constructed from the associated splicing graph $\mathcal{G}^S = (V, E)$. Like in the first benchmark, we only used known exon boundaries and the mapped reads as input for CIDNE.

Given the solution of the first step (Phase I) with set T^* of isoforms with non-zero estimated abundance and selected regularization parameter λ^*, we used the following protocol for the delayed column generation step:

- An edge (v_i, v_j) was added to the set E^{new} if its inclusion allowed for explanation of a segment cover $c_i = (c_i', c_i'', b_i)$ from the set $\tilde{C} \subseteq C$ of segment covers (excluding faux covers) not supported by any candidate isoform encoded in $\mathcal{G}^S$. In detail, (v_i, v_j) was added to E^{new} if the augmented splicing graph $\mathcal{G}' = (V, E \cup (v_i, v_j))$ contained a path $p = \langle v_k, \ldots, v_i, v_j, \ldots, v_l \rangle$ with associated sequence of segments p', such that $\bar{\ell}_{p', c_i} > 0$, for some segment cover $c_i \in \tilde{C}$ with $\text{first}(c_i') = s_k$ and $\text{last}(c_i'') = s_l$. With this set of potential new edges, we started the delayed column generation algorithm with initial basis T^*.
- To facilitate the generation of new candidate isoforms, we used a slightly reduced regularization parameter $\lambda = 0.9 \cdot \lambda^*$.
- Every new candidate isoform had to explain at least one segment cover in $\tilde{C}$ ($k^c = 1$) and was allowed to use at most two edges from the set E^{new} ($k^e = 2$).
- Once the optimal solution was found, we constructed a candidate superset by combining the set of candidates encoded in $\mathcal{G}^S$ and all new isoforms generated during the course of the delayed column generation algorithm (not only the final solution). With this enhanced candidate set we repeated the model selection to compute the final set of expressed isoforms. To enhance the selectivity of our method, new candidates were assigned a higher weight of $\lambda^{\text{new}} = 1.3 \cdot \lambda$ in the regularized least squares model.

For performance reasons, the delayed column generation algorithm was applied only for genes containing at most 50 segments.

Using this procedure, CIDNE successfully recovered ∼24.6% of the expressed transcripts with at least one uncovered splice junction. Cufflinks, without annotation, and IsoLasso, provided with exon boundaries, both did not recover a single isoform, while SLIDE recovered ∼5%, also using annotated exon boundaries.

When provided with the same incomplete annotation as in the previous benchmark, Cufflinks recovered ∼22% and iReckon recovered ∼28% of the transcripts. However, considering the subset of 42 transcripts not included in the annotation provided to both tools, iReckon and Cufflinks recovered only one and two transcripts respectively, whereas CIDNE recovered 17 (∼40%). We remark that for each of the three isoforms recovered by iReckon and Cufflinks, another isoform also containing the uncovered splice junction was found in the annotation provided to both tools. Thus, both tools did not recover any novel uncovered splice junction.

Looking at the complete set of ∼6300 transcripts, the recall was increased from ∼62.8% to ∼63.1% with an acceptable decrease in precision from ∼56.9% to ∼56.5%, compared to the solution of Phase I.

Running time

All benchmarks were performed on a machine equipped with 2 Intel Xeon CPU X5550 @2.67GHz Quad Core and 72 GB memory, and all tools with multithreading support (Cufflinks, iReckon, SLIDE) were allowed to use up to 16 threads.

For the 75 bp dataset with 40 million paired-end reads in the first benchmark, the running times of CIDNE (∼12 minutes), CuffLinks (∼21 minutes), IsoLasso (∼22 minutes) and CLASS (∼73 minutes) were all within minutes to less than two hours, while SLIDE required more than one week to complete. For the second benchmark with incomplete annotation, we observed running times of ∼2 minutes for CIDNE, ∼3 minutes for Cufflinks and ∼175 minutes for iReckon. In the last benchmark, the application of the delayed column generation algorithm increased the running time from ∼2 minutes to ∼15 minutes. Note that the current implementation of CIDNE uses only one thread, except for the pricing ILP solver, which could also use up to 16 threads.

4.8 Conclusion

In this chapter, we presented our approach to the isoform inference and abundance estimation problem using RNA-Seq data. We demonstrated that our delayed column generation algorithm can be used to recover isoforms containing splice junctions not covered by any mapped reads, without exhaustive enumeration of candidate isoforms. Using this approach, CIDNE was able to effectively recover unknown isoforms with uncovered splice junctions, clearly outperforming all other tools on this task. For datasets containing a greater fraction of transcripts with uncovered splice junctions, the impact of our delayed column generation step on the overall performance will very likely be increased.

Our algorithm directly exploits paired-end read information and it can incorporate additional sources of information into the inference process such as known or predicted TSSs and PASs and known transcript annotations, which can improve the identification and quantification performance. We remark that although the known exon boundary information provided to CIDNE constitutes an advantage over Cufflinks and CLASS in the first benchmark, CIDNE also clearly outperformed IsoLasso and SLIDE that were both provided with the same information. Further, in the second benchmark CIDNE also outperformed Cufflinks when both tools were provided with an incomplete annotation.

The presented ILP formulation for the pricing problem allowed us to implement several restrictions for the generation of new isoforms and it can be easily adapted to modify the column generation strategy. For instance, we could extend the search space to include also isoforms containing TSSs or TASs not included in the sets $\mathcal{TSS}$ or $\mathcal{PAS}$. In such a scenario, we could easily adapt the pricing ILP formulation to charge an extra penalty when using such a new TSS or PAS. Another possibility is to enforce certain properties of the new transcripts like, e.g., the presence of a certain set of segments, a certain upper bound or lower bound for the total number of exons or the transcript sequence length. These kinds of restrictions allow for a controlled extension of the search space and the integration of prior knowledge into the inference process.

Our current implementation employs a normal distribution of fragment lengths, however, the model can also use any other distribution. Thus far, CIDNE does not implement any correction of sequence-specific fragment biases due to *random hexamer priming* [Hansen et al., 2010] and *GC content* [Benjamini and Speed, 2012], or positional fragment biases [Bohnert and Rätsch, 2010], that produce non-uniform read mapping coverage along the transcript sequence. When solving the regularized least squares problem on a given set of isoform candidates, we can account for both types of biases by incorporating correction factors into the computation of the adjusted segment cover lengths. In the same way, we can also introduce correction factors for sequence-specific biases into the column generation step. However, the pricing ILP requires further adaptions to integrate positional bias correction also into the column generation algorithm.

In addition, there are further opportunities for improvement in the post-processing stage by using more sophisticated methods to compute the final abundance estimates for the set of predicted isoforms. A promising approach is the application of some expectation maximization method, like IsoEM [Nicolae et al., 2011], that operates on the level of individual mapped reads instead of segment cover counts. Further, we plan to extend our implementation by some statistical method to estimate confidence values to quantify the reliability of predicted isoforms and to obtain confidence intervals for the estimated abundances. Finally, we will further analyze how the efficiency of the pricing step can be enhanced by either improving the ILP formulation or by means of a different algorithm or efficient heuristics.

4.9 Contributions

The algorithmic approach presented in this chapter has been developed by the author and Dr. Stefan Canzar. Most of the implementation was performed by the author, Dr. Stefan Canzar implemented parts of the pricing ILP and the piecewise linear approximation model. Dr. David Weese provided some implementations for the preprocessing of gene annotation data and read alignment data. All benchmarks were conducted by the author.

Chapter 5 Solving The Duplication-Loss Alignment Problem With Branch And Cut

In this chapter, which is based on and in parts recites our recent journal publication [Andreotti et al., 2013], we present our branch and cut approach to a phylogenetic inference problem. The mathematical approach has been developed in a joint project with Dr. Stefan Canzar, where most of the implementation was performed by the author and a complexity proof was conducted by Dr. Stefan Canzar exclusively.

5.1 Background

In the course of evolution, genomes are continually modified by mutations ranging from single nucleotide mutations to genome-scale changes that can include the duplication of a complete genome. These genome-scale events can be divided into *rearrangement operations* that change the order of genes like inversions, transpositions and translocations and *content modifying operations* like insertions, deletions or duplications of single or multiple genes. Since gene duplications are a source of raw genetic material for the development of novel gene functions, they are assumed to be particularly important for the evolution of eukaryotic species [Hahn et al., 2007; Blomme et al., 2006; Cotton and Page, 2005; Ohno, 1970] and the generation of biodiversity [Lynch and Conery, 2000]. In contrast, gene loss through deletion or pseudogenization is a common fate of duplicated genes, where one of the two identical gene copies subsequently accumulates deleterious mutations [Hahn et al., 2007; Blomme et al., 2006; Cotton and Page, 2005; Ohno, 1970; Lynch and Conery, 2000]. In addition, gene loss is also considered an advantageous response to changing selective pressures according to the "less is more" hypothesis by Olson [1999], supported by results reported in [Greenberg et al., 2006; Wang et al., 2006; Koskiniemi et al., 2012]. The evolutionary importance of duplication and loss events is further underlined by the very high estimated average rate of 0.01 per gene per million years for eukaryotes such as *Homo sapiens, Mus musculus, Drosophila melanogaster, Caenorhabditis elegans, and Arabidopsis thaliana* [Lynch and Conery, 2000]. Recent availability of fully sequenced genomes of related species facilitates the study of large-scale differences in gene complements originating from gene duplication and loss at an unprecedented resolution. A prominent example of a gene family that is continuously duplicated and lost is transfer RNA (tRNA). In *Escherichia coli*, for example, the rate at which tRNA genes evolve by duplication and loss events has been estimated to be of the order of one per million years per genome per lineage [Withers et al., 2006], and for Drosophila it was estimated to be 3-4 times higher than for protein coding genes [Rogers et al., 2010]. At the same time, studying the evolution of tRNA is particularly challenging, as functionally equivalent tRNAs exhibit a very high sequence similarity that impedes the distinction between *orthologs*, i.e., genes descending from a common ancestral gene through a speciation event, and *paralogs*, i.e., genes created by a duplication event within the genome [Rogers et al., 2010; Withers et al., 2006].

Still, the evolutionary principles and implications of the high duplication and loss rate for tRNA genes and their considerable copy number variation, even between closely related species, are not well understood and will be subject to many future comparative genomics studies.

One key step towards this understanding is the inference of evolutionary histories from available genomes of related extant species. This requires methods to compare complete genomes and to infer an ancestral genome which, in the case of a parsimony approach, implies a minimum number of evolutionary operations. Given the genomes for a set of extant species and their phylogenetic tree (species tree), the problem of inferring ancestral genomes that minimize the overall number of evolutionary operations for the complete tree is known as the *small phylogeny problem*.

Genome comparison is frequently addressed by the genome rearrangement approach where a genome is represented by linear or circular sequences of genes. Often this sequence is signed such that every gene in the sequence is tagged by a "+″ or a "−″, which represents its orientation in the genome. The genomic distance between two genomes is then defined as the minimum number of rearrangement operations that are required to transform one genome into the other. Another assumption usually made is that every gene occurs exactly once in each genome, which can then be represented as a signed permutation. Although this is often not realistic in a whole genome context, this restriction allows for the application of polynomial time algorithms to compute the rearrangement distance between two signed permutations [Hannenhalli and Pevzner, 1995; 1999], subject to different rearrangement operations like *inversions* or the *double-cut-and-join* operation [Yancopoulos et al., 2005]. In contrast, the problem of computing the rearrangement distance between two genomes with duplicated genes is NP-hard for almost all studied distance measures [Fertin et al., 2009].

In their recent work, Holloway et al. [2012] proposed a model that, in contrast to previous models, considers only the two content modifying operations: duplication and loss. This restriction is particularly useful for the study of gene families that evolve predominantly by these operations and where a model including rearrangement operations may be inappropriate.

Due to this restriction to content modifying and gene order preserving operations, Holloway et al. proposed to formulate the problem of inferring the most parsimonious ancestor, i.e., the median of two genomes, in the duplication-loss model as an alignment problem. This so-called *duplication-loss alignment problem* is the problem of finding an optimal *labeled alignment* of two genomes. In a labeled alignment, every unaligned gene is labeled either as a loss or as the product of some duplication event in the evolutionary history from the common ancestor. An optimal labeled alignment is a labeled alignment with minimal summed cost of the implied evolutionary events.

As opposed to symmetrical distance measures like inversion distance or Hamming distance, the asymmetrical operations duplication and loss are unambiguously applied to one of the two sequences, which allows to infer an (almost[1]) unique ancestral genome from a labeled alignment. The duplication-loss alignment problem is naturally related to the classical sequence alignment problem, where only the symmetric operations substitution and insertion/deletion are allowed. While the pairwise sequence alignment problem without duplication events can be solved efficiently with the Needleman-Wunsch dynamic programming algorithm [Needleman and Wunsch, 1970], Holloway et al. pointed out that no algorithm is known to optimally solve the duplication-loss alignment problem, though the complexity of the problem was not shown. Therefore, Holloway et al. proposed an integer linear programming formulation.

We were able to prove that the duplication-loss alignment problem is NP-hard [Andreotti et al., 2013][2]. In another publication Benzaid et al. [2013] showed that also the related problem of optimally labeling a given fixed alignment of two genomes by duplications and losses is APX-hard and proposed a heuristic for the duplication-loss alignment problem.

[1] The only exception is discussed in Section 5.2.
[2] The NP-hardness proof was conducted by Dr. Stefan Canzar.

New algorithm

Building on the work of Reinert et al. [1997] and Althaus et al. [2005] for the *multiple sequence alignment* (MSA) problem, we present an exact branch and cut algorithm that outperforms the algorithm of Holloway et al. by several orders of magnitude. Our branch and cut algorithm is based on three classes of valid cuts that can be separated efficiently. One class, the *maximal clique inequalities* with duplication events, is closely related to the maximal clique inequalities for MSA as defined in [Reinert, 1999] and can be separated in a similar way. The other two classes of valid cuts, called *lifted duplication cycle inequalities* and *duplication island inequalities*, are based on our new insights into the combinatorial structure of duplication events.

Due to the immense performance gain, our algorithm can compute optimal labeled alignments for pairs of longer and more distantly related genomes than the algorithm by Holloway et al. Moreover, the performance of our algorithm enables us to extend the model in order to solve a slightly restricted variant of the *median-of-three problem*, asking for a genome that implies a minimal number of operations in the evolution from its immediate ancestor to its two immediate descendant genomes in a phylogenetic tree. This problem is of particular interest as it presents the key component of the widely used *Steinerization* method [Sankoff and Blanchette, 1997; Blanchette et al., 1997] to solve the small phylogeny problem towards a local optimum. After some initialization of the ancestral genomes in a phylogenetic tree, the Steinerization method proceeds by repeatedly replacing ancestral genomes with the median of the neighboring genomes in the tree, until it reaches a local optimum. In a proof of concept study, we will demonstrate that our median formulation, embedded in the Steinerization method, can reduce the number of implied duplication and loss events in a complete phylogenetic tree compared to the bottom-up heuristic proposed by [Holloway et al., 2012], which uses only the pairwise labeled alignment method. Further, on simulated instances, we can show that this reduced number of implied evolutionary operations is accompanied by an increased accuracy of predicted ancestral genomes.

5.2 The duplication-loss model for two species

5.2.1 Basic definitions

Throughout this chapter, we use the following basic definitions adopted from Holloway et al. [2012].

We define a *genome* as a string over an alphabet Σ, whose characters represent specific gene families. For two genomes G^1 and G^2, and a set O of evolutionary operations, an *evolutionary history*, $O_{G^1 \to G^2}$ is defined as a sequence of n operations from O transforming G^1 into G^2. Every operation o_i, denoting the i-th operation in $O_{G^1 \to G^2}$, is assigned a cost $c(o_i)$. Consequently, the cost of the complete evolutionary history is defined as $\sum_{i=1}^{n} c(o_i)$. The genome G^1 is called a *potential ancestor* of G^2 if there exists at least one evolutionary history $O_{G^1 \to G^2}$. In this case, the cost $c(G^1 \to G^2)$ for transforming G^1 to G^2 is defined as the minimal cost over all possible histories $O_{G^1 \to G^2}$. In the following section, we will present our approach to the small phylogeny problem restricted to two species, defined as follows:

Definition 5.1. Two Species Small Phylogeny Problem (2-SPP)
Given two genomes, G^1 and G^2, and a set of evolutionary operations O, determine the potential common ancestor G^* minimizing the cost $c(G^* \to G^1)+c(G^* \to G^2)$

Further, Holloway et al. restrict their evolutionary model to the following two content modifying operations:

- A *duplication* of size $k+1$ on genome G copies a substring $G[i \mathinner{..} i+k]$, called the *origin*, to a location $G[j \mathinner{..} j+k]$, called the *target* of the duplication. Origin and target intervals must be disjoint.
- A *loss* of size $k+1$ removes a substring $G[i \mathinner{..} i+k]$ from genome G.

As an additional restriction of the model, origin and target of a duplication must be present as contiguous blocks in the genomes. This leads to the following definitions of a *visible history* and a *visible ancestor*:

Definition 5.2. Visible history
An evolutionary history $O_{G^1 \to G^2}$ is called a visible history and G^1 a visible ancestor of G^2 if no duplication in $O_{G^1 \to G^2}$ is modified subsequently by inserting (by duplication) or deleting (by loss) genes from its origin or target.

With a slight abuse of notation, we will use the term visible history also for pairs of evolutionary histories $(O_{G^* \to G^1}, O_{G^* \to G^2})$, from a potential common ancestor G^* to two genomes G^1 and G^2, if *both* histories $O_{G^* \to G^1}$ and $O_{G^* \to G^2}$ are visible. In this case, we call the common ancestor G^* a visible ancestor of G^1 and G^2.

The restriction of the evolutionary model to order-preserving duplication and loss operations and visible histories allows to formulate the Two Species Small Phylogeny Problem of visible ancestors as an alignment problem, as shown in [Holloway et al., 2012].

To obtain an alignment of two genomes G^1 and G^2, gaps (“$-$”$\notin \Sigma$) can be introduced into both genomes such that the resulting strings have the same length.

Each of the strings then corresponds to a row of the two-dimensional alignment matrix $\mathcal{A}$. Every column of $\mathcal{A}$ either *matches* two (ortholog) genes from Σ, or aligns a gene $G^1[i]$ (resp. $G^2[i]$) with a gap. A gene $G^1[i]$ (resp. $G^2[i]$) that is aligned to a gap must be either the product of a duplication in the evolutionary history from a potential common ancestor G^* to G^1 (resp. G^2) or part of a loss in the evolutionary history from G^* to G^2 (resp. G^1).

For a given alignment $\mathcal{A}$, an interpretation of unmatched genes as products of a sequence of duplication and loss operations is called a *labeling* of $\mathcal{A}$. The alignment together with this labeling is then called a *labeled alignment*. As described by Holloway et al. and presented in Figure 5.1, there exists a one-to-one correspondence between a labeled alignment of G^1 and G^2 and visible ancestors of G^1 and G^2, with a single exception also depicted in Figure 5.1.

We are now ready to define the duplication-loss alignment problem as follows:

Definition 5.3. DUPLICATION-LOSS ALIGNMENT PROBLEM
Let the cost of a labeled alignment be the sum of the costs of the operations implied by the labeling. Given two genomes, G^1 and G^2, compute a labeled alignment of G^1 and G^2 with minimum cost.

5.3 Problem formulation and valid inequalities

We begin with a graph-theoretical representation of the duplication-loss alignment problem, which is then formulated as an ILP, similar to the one proposed by Holloway et al. Afterwards, we present three classes of valid inequalities that lead to an efficient branch and cut algorithm.

5.3.1 Graph-theoretical formulation

Given two genomes, G^1 and G^2, the associated *complete alignment graph* $\mathcal{G}' = (V^1 \cup V^2, E, H)$ is a complete bipartite graph with vertices $V^\ell = \{v_j^\ell \mid 1 \leq j \leq |G^\ell|\}$, $\ell \in \{1, 2\}$ (see Figure 5.2). A vertex $v_j^\ell \in V$ represents gene $G^\ell[j]$ and undirected edges $\{v_i^1, v_j^2\} \in E$ represent the alignment of genes $G^1[i]$ and $G^2[j]$. For some alignment $\mathcal{A}$ of genomes G^1 and G^2, an alignment edge $\{v_i^1, v_j^2\}$ is said to be realized if $\mathcal{A}$ aligns gene $G^1[i]$ with $G^2[j]$. Every edge $\{v_i^1, v_j^2\}$ is assigned a cost $c_{i,j}$, representing the cost to align the corresponding genes. For the evolutionary model in the duplication-loss alignment problem, the cost of alignment edges between genes from the same gene family is zero and alignment edges between genes of different gene families have infinite cost. The

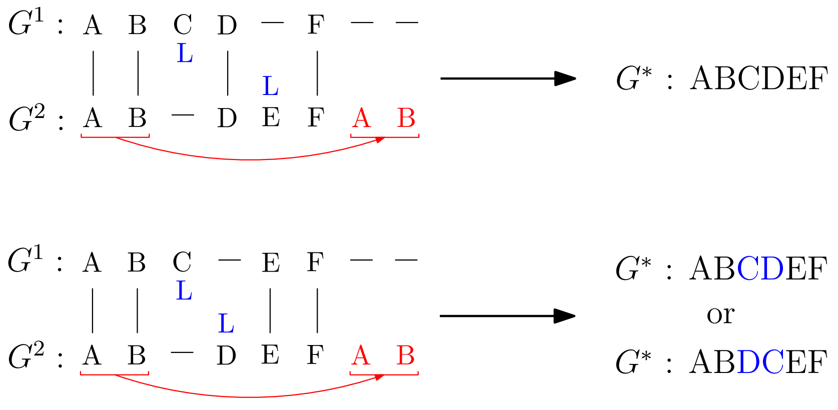
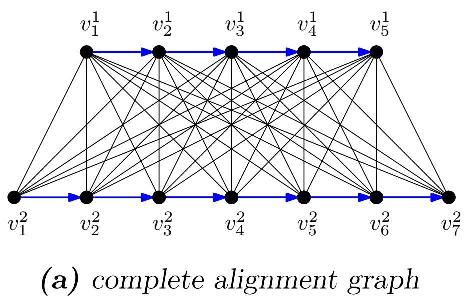


Figure 5.1: *Correspondence between a labeled alignment of genomes G^1 and G^2 and visible ancestor G^*. In the top labeled alignment, gene C in G^1 and gene E in G^2 are labeled as losses. This implies that a visible ancestor G^* of G^1 and G^2 contained both genes and that gene C was lost in the evolution from G^* to G^2, whereas gene E was lost in the evolution from G^* to G^1. Additionally, the rightmost occurrence of genes A and B in G^2 is labeled as the product of a duplication of the leftmost occurrence of genes A and B during evolution G^* to G^1. Hence, the rightmost genes A and B in G^2 are not contained in the ancestral genome, which can therefore be uniquely determined as ABCDEF. The bottom labeled alignment presents the only situation where no unique ancestral genome can be inferred. If two genes $G^1[i]$ and $G^2[j]$ are labeled as gene losses, their order in G^* can be uniquely determined only if the alignment contains a pair of matched genes $G^1[k]$ and $G^2[l]$, with either $k > i$ and $l < j$ or $k < i$ and $l > j$.*

set H of directed edges contains an edge (v_i^ℓ, v_{i+1}^ℓ) to connect every pair of consecutive genes $G^\ell[i]$ and $G^\ell[i+1]$ within the same genome G^ℓ. The subgraph $\mathcal{G} \subseteq \mathcal{G}'$, obtained by removing all alignment edges with infinite cost from $\mathcal{G}'$ is called the *alignment graph* for G^1 and G^2.

Further, we define D^ℓ as the set of all possible duplications, and L^ℓ as the set of all possible loss events for genome G^ℓ. The set of all possible duplications is generated by identification of all pairs of non-overlapping identical substrings in G^ℓ, each contributing two possible duplications with interchanged origin and target. For a duplication $d \in D^\ell$ with origin $G^\ell[i\,..\,i+k]$ and target $G^\ell[j\,..\,j+k]$, the functions origin(d) and target(d) are defined as origin$(d) := [i\,..\,i+k]$ and target$(d) := [j\,..\,j+k]$. Similarly, for a loss event $l \in L^i$, removing substring $G^i[j\,..\,j+k]$ from G^i, the function span(l) is defined as span$(l) := [j\,..\,j+k]$.



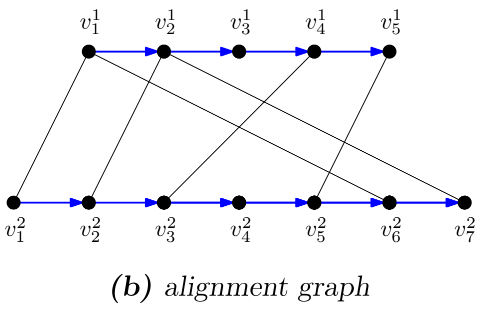


Figure 5.2: *Complete alignment graph and alignment graph for genomes $G^1 = ABCDF$ and $G^2 = ABDEFAB$. Blue (directed) edges connect vertices corresponding to consecutive genes in each genome, and (undirected) alignment edges connect vertices for genes in G^1 to vertices for genes in G^2. The complete alignment graph contains an alignment edge $\{v_i^1, v_j^2\}$ for all $1 \leq i \leq |G^1|$ and $1 \leq j \leq |G^2|$. For the duplication-loss model without substitutions, the alignment graph (b) contains an alignment edge $\{v_i^1, v_j^2\}$ only if $G^1[i] = G^2[j]$.*

The cost of every evolutionary operation is denoted by c_d for every duplication $d \in D^1 \cup D^2$ and c_l for every loss $l \in L^1 \cup L^2$.

Valid labeled alignments

In order to be valid, every labeled alignment must satisfy the following three conditions:

First, only certain pairs of alignment edges can be realized simultaneously. In a graphical representation of the alignment graph, as shown in Figure 5.2, a pair of alignment edges is *incompatible* if they share a common endpoint or if they are crossing. The first case implies a gene being aligned to two genes whereas the second case violates the co-linearity (order preserving) property of an alignment. Thus, two alignment edges $e_1 = \{v_i^1, v_j^2\}$ and $e_2 = \{v_k^1, v_l^2\}$ are incompatible if and only if either $i \leq k$ and $l \leq j$, or $i \geq k$ and $l \geq j$. The set of all pairs of incompatible alignment edges is denoted by $\mathcal{I}$.

Second, a reasonable biological interpretation of duplications in an evolutionary history imposes a (chronological) partial order "$\leq$" on the set of duplications. If the target of some duplication d_1 overlaps the origin of a duplication d_2 it must hold that d_1 occurred before d_2, and thus $d_1 \leq d_2$. Therefore, the antisymmetry of partial orders implies that every sequence of duplications $d_1, d_2, \ldots, d_k$ with overlapping targets and origins must not form a *duplication cycle* defined as follows:

Definition 5.4. DUPLICATION CYCLE
A set of duplication events $D \subseteq D^\ell$ forms a duplication cycle if and only if there exists a permutation $d_1, d_2, \ldots, d_k$ of the elements in D such that

$$\operatorname{origin}(d_i) \cap \operatorname{target}(d_{i-1}) \neq \emptyset,\ \forall\ 2 \le i \le k \quad \textit{and} \quad \operatorname{origin}(d_1) \cap \operatorname{target}(d_k) \neq \emptyset\,.$$

The third and last condition for a valid labeled alignment requires that every gene in genome G^ℓ is either aligned to a gene in the other genome, lies in the span of a loss $l \in L^\ell$, or lies in the target of a duplication $d \in D^\ell$. In the ILP formulation presented in the next section, each of these three conditions is captured by a class of constraints.

5.3.2 Initial ILP formulation

In the remainder of this chapter, we restrict the model such that loss events are limited to single genes like in the model proposed by Holloway et al. Thus, we can use a simplified notation, where for every vertex $v_i^\ell \in V^\ell$, the loss event $l_{v_i} \in L^\ell$ denotes the loss of the single gene $G^\ell[i]$.

The three conditions for a valid duplication-loss alignment of two genomes can be formulated in an ILP as follows. We introduce a binary variable:

- $x_{i,j}$ for every alignment edge $\{v_i^1, v_j^2\} \in E$,
- y_d for every possible duplication $d \in D^1 \cup D^2$,
- z_v for every possible loss $l_v \in L^1 \cup L^2$.

Further, we denote by D^* the set of all duplication cycles in D^1 and D^2. Using these definitions, an optimal solution to the duplication-loss alignment problem can be obtained by solving the ILP formulation (5.1)-(5.7).

$$\min \quad \sum_{\{v_i^1, v_j^2\} \in E} c_{i,j} x_{i,j} + \sum_{v \in V} c_v z_v \quad + \sum_{d \in D^1 \cup D^2} c_d y_d \tag{5.1}$$

$$\text{subject to} \quad x_{i,j} + x_{kl} \le 1 \qquad \forall \{\{v_i^1, v_j^2\}, \{v_k^1, v_l^2\}\} \in \mathcal{I} \tag{5.2}$$

$$\sum_{d \in D} y_d \le |D| - 1 \qquad \forall D \in D^* \tag{5.3}$$

$$z_{v_i^\ell} + \sum_{\{v_i^\ell, v_j^{\bar\ell}\} \in E} x_{i,j} + \sum_{\substack{d \in D^\ell \\ i \in \operatorname{target}(d)}} y_d = 1 \qquad \forall 1 \le i \le |G^\ell|,\ \ell \in \{1,2\},\ \bar\ell := 3 - \ell \tag{5.4}$$

$$x_{i,j} \in \{0,1\} \qquad \forall \{v_i^1, v_j^2\} \in E \tag{5.5}$$

$$y_d \in \{0,1\} \qquad \forall d \in D^1 \cup D^2 \tag{5.6}$$

$$z_v \in \{0,1\} \qquad \forall v \in V \tag{5.7}$$

Constraints (5.2), (5.3) and (5.4) capture the three conditions for a valid duplication-loss alignment in the same order.

A generalization of this ILP formulation to multi-gene loss events would contain a variable for each of the $\mathcal{O}(|G^1|^2 + |G^2|^2)$ potential loss events. Further, in the *cover constraint* (5.4), variable $z_{v_i^\ell}$ would be replaced with the sum of variables z^* for all loss events $l^* \in L^\ell$ with $i \in \text{span}(l^*)$.

The ILP (5.1)-(5.7) cannot be solved directly by a standard ILP solver on realistic instance sizes, as it contains one constraint (5.3) for every duplication cycle and the number of possible duplication cycles grows exponentially with the genome length in the worst case. Therefore, initially the ILP is relaxed by removing all duplication cycle constraints and subsequently adding violated duplication cycle inequalities as cutting planes during the ILP solving process. While this approach seems to be very similar to the strategy of Holloway et al., the difference is significant. Holloway et al. also drop the duplication cycle inequalities from the initial ILP before they iteratively solve the ILP, add violated duplication cycle inequalities, and re-solve it, until no more duplication cycle constraint is violated. The important difference between the two approaches is that in every iteration, Holloway et al. solve an ILP, which is computationally expensive. Further, in our approach we do not add *ordinary* violated duplication cycle inequalities, as defined by (5.3), but instead, we use a dominating class of cutting planes to obtain a stronger LP-relaxation. In combination with two other classes of valid cutting planes, which we introduce in the next section, we obtain a stronger LP relaxation that allows for pruning large parts of the branch and cut tree, resulting in a much better practical performance compared to the approach by Holloway et al.

5.4 Valid cuts and separation

In the following sections, we introduce three classes of inequalities that are valid for the duplication-loss alignment problem, and we describe how to efficiently solve the separation problem for each of them. In the remainder, we let P denote the convex hull of the feasible solutions to the ILP (5.1)-(5.7). Further, we define the *incompatibility graph* $\mathcal{H}$ as an undirected graph that contains a vertex for every binary variable and an edge for every pair of (incompatible) variables that cannot both have value 1 in a feasible solution.

In the following lemma we state that P has dimension $|E| + |D^1| + |D^2|$. This result will be used in Theorem 5.1.

Lemma 5.1. *The dimension* $\dim(P)$ *of* P *is* $|E| + |D^1| + |D^2|$.

Proof. As shown in [Nemhauser and Wolsey, 1988], the dimension of a polyhedron defined by a linear system $\mathbf{Ax} \leq \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{x} \in \mathbb{R}^n$ with equalities (explicit and implicit) $\mathbf{A}^{=}\mathbf{x} \leq \mathbf{b}^{=}$ and remaining inequalities $\mathbf{A}'\mathbf{x} \leq \mathbf{b}'$ has dimension $n - \text{rank}(\mathbf{A}^{=})$. An inequality $\mathbf{a}_i^\top \mathbf{x} \leq b_i$ is called an implicit equality if $\mathbf{a}_i^\top \mathbf{x}^* = b_i$ holds for every feasible solution $\mathbf{x}^*$.

Our ILP formulation for the duplication-loss alignment contains exactly one equality constraint (5.4) for every gene in G^1 and G^2, which belongs to the system $\mathbf{A}^{=}\mathbf{x} \leq \mathbf{b}^{=}$. Obviously, the corresponding rows in $\mathbf{A}^{=}$ are linearly independent as every row has a unique single non-zero entry for a loss variable. Hence, it holds that $\text{rank}(\mathbf{A}^{=}) \geq |L^1| + |L^2|$. It follows that $(|E| + |D^1| + |D^2| + |L^1| + |L^2|) - (|L^1| + |L^2|) = |E| + |D^1| + |D^2|$ is an upper bound for $\dim(P)$. We can easily show that this bound is tight by constructing a set of $|E| + |D^1| + |D^2| + 1$ affinely independent points that are feasible solutions to ILP (5.1)-(5.7). By setting exactly one duplication variable $y_d, d \in D^1 \cup D^2$ or exactly one alignment edge variable $x_e, e \in E$ to one and covering all remaining uncovered genes by the associated loss variable we obtain $|E| + |D^1| + |D^2|$ affinely independent feasible solutions. Together with the feasible solution where every gene is covered by its associated loss variable (no active alignment edge or duplication) we have $|E| + |D^1| + |D^2| + 1$ affinely independent points that are feasible for ILP (5.1)-(5.7), which proves the claim. □

5.4.1 Lifted duplication cycle inequalities

The *lifted duplication cycle inequalities* are a class of constraints that dominate the ordinary duplication cycle inequalities (5.3). Since the definition considers duplications in only one of the two genomes, let $D := D^\ell, \ell \in \{1, 2\}$. This class of constraints is based on a idea similar to the lifted mixed cycle inequalities for multiple sequence alignment, introduced in [Althaus et al., 2005].

Given a set of duplications $C \subseteq D$, which is partitioned into sets $C^1, \ldots, C^t$, the inequality

$$\sum_{d \in C} y_d \leq t - 1\,, \tag{5.8}$$

is valid for P if C satisfies the following two conditions:

(C1) For $r = 1, \ldots, t$, all duplications in C^r are pairwise incompatible.

(C2) Every set $\{d_1, \ldots, d_t\}$, where d_r is chosen arbitrarily from C^r for $r = 1, \ldots, t$, forms a duplication cycle according to Definition 5.4.

Note that the original duplication cycle inequalities (5.3) define a special case of (5.8) with every set C^r being a singleton. A constraint of type (5.8) is called a lifted duplication cycle inequality if the set C also satisfies a third condition:

(C3) C is maximal with respect to properties (C1) and (C2), i.e., C cannot be extended without violating (C1) or (C2).

Figure 5.3 shows an example of a fractional solution to the LP relaxation of the naïve ILP (5.1)-(5.7) that can be cut off by a lifted duplication cycle inequality (5.8).

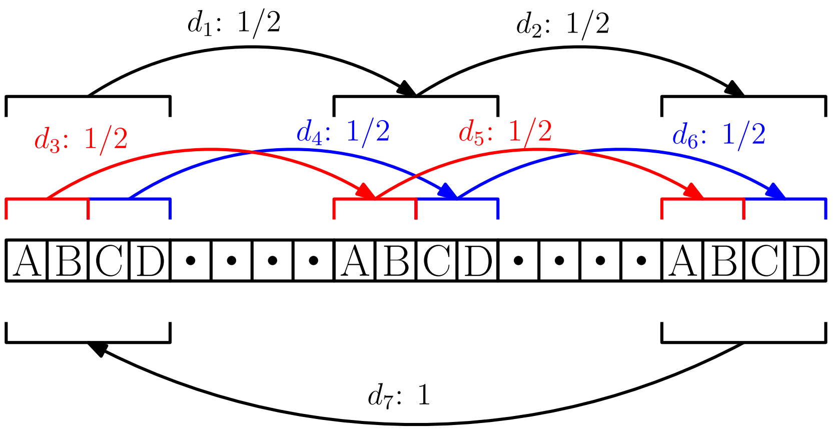


Figure 5.3: *A feasible fractional solution to the LP relaxation of* (5.1)-(5.7) *that is cut off by lifted duplication cycle inequalities. Numbers denote for every duplication the value of the associated variable. The set of duplications* $\{d_1, d_2, d_3, d_5, d_7\}$ *forms a lifted duplication cycle with violated inequality* $y_{d_1} + y_{d_2} + y_{d_3} + y_{d_5} + y_{d_7} \leq 2$ *for the partition* $C^1 = \{d_1, d_3\}$, $C^2 = \{d_2, d_5\}$, *and* $C^3 = \{d_7\}$. *The same holds for the set* $\{d_1, d_2, d_4, d_6, d_7\}$. *However, no ordinary duplication cycle inequality is violated.*

Even though the lifted duplication cycle inequalities lead to a tighter LP relaxation and a more efficient algorithm than ordinary duplication cycle inequalities, they are in general not facet defining for P. To show this, consider a lifted duplication cycle where C is partitioned into at least three components $C^1, \ldots, C^k$. Further assume there exists some duplication d' that bypasses C^1, meaning that d' is incompatible to every duplication $d_1 \in C^1$, and every set $\{d_2, \ldots, d_{k-1}\}$, where d_r is chosen arbitrarily from C^r, together with d' forms a duplication cycle. Every such duplication could be added to the lifted duplication cycle inequality associated to C, resulting in an inequality that is still valid for P and dominates the lifted duplication cycle inequality. However, the efficient separation for a slight relaxation of lifted duplication cycle inequalities, as presented below, makes this class of inequalities particularly useful for our branch and cut algorithm.

Separation of lifted duplication cycle inequalities

Our algorithm for efficient separation of a slight relaxation of the lifted duplication cycle inequalities is based on the following proposition:

Proposition 5.1. *An inequality of the form* (5.8) *with* $C = \bigcup_{i=1}^{t} C^i$, $C \subseteq D$ *is a lifted duplication cycle inequality if and only if there exists a sequence of non-empty intervals* $[a_1 \,..\, b_1], [a_2 \,..\, b_2], \ldots, [a_t \,..\, b_t]$ *such that for* $i = 1, \ldots, t$, *the following three properties hold:*

(P1) $\bigcap_{d \in C^i} \text{target}(d) = [a_{i+1} \,..\, b_{i+1}]$,

(P2) $\forall d \in C^i : \text{origin}(d) \cap [a_i \,..\, b_i] \neq \emptyset$,

(P3) $\forall d \in D \setminus C : \text{target}(d) \cap [a_{i+1} \,..\, b_{i+1}] \neq \emptyset \rightarrow$

$$\text{origin}(d) \cap [a_i \,..\, b_i] = \emptyset \vee \exists d' \in C^{i+1} : \text{target}(d) \cap \text{origin}(d') = \emptyset \,,$$

where $[a_{t+1} \,..\, b_{t+1}] := [a_1 \,..\, b_1]$ *and* $C^{t+1} := C^1$.

We omit the proof of this proposition, which can be found in [Andreotti et al., 2013].

Based on this characterization of lifted duplication cycles, we propose an algorithm to separate a relaxation of the lifted duplication cycle inequalities where all intervals $[a_i \,..\, b_i]$ in Proposition 5.1 are restricted to length 1. Let $\mathcal{C}$ denote the set of all lifted duplication cycles fulfilling this property. The central component of our algorithm is the construction of a directed, edge weighted graph $\mathcal{G}_{\mathcal{C}} = (V, A, w)$ that allows us to reduce the separation of all lifted duplication cycle inequalities in $\mathcal{C}$ to a series of shortest path computations.

Graph $\mathcal{G}_{\mathcal{C}}$ contains vertices $v_1, \ldots, v_n$, one for every gene in the genome of length n. For every pair of vertices v_i and v_j, we compute the set $\mathcal{D}(i,j) := \{d \in D \mid i \in \text{origin}(d) \wedge j \in \text{target}(d)\}$ of duplications whose origins contain the i-th gene and whose targets contain the j-th gene. If this set is non-empty, $\mathcal{G}_{\mathcal{C}}$ contains an edge (v_i, v_j) with weight $w((v_i, v_j))$, defined as follows:

$$w((v_i, v_j)) := 1 - \sum_{d \in \mathcal{D}(i,j)} y^*_d \,.$$

Thus, for every lifted duplication cycle $c \in \mathcal{C}$ with associated singleton target intervals $c_1, \ldots, c_t$, the violation of the corresponding lifted duplication cycle inequality equals

$$\sum_{j=1}^{t} \sum_{d \in C^j} y^*_d - t + 1 = 1 - \sum_{j=1}^{t} (1 - \sum_{d \in C^j} y^*_d) = 1 - \sum_{j=1}^{t} w((v_{c_j}, v_{c_{j+1}})) \,,$$

with $v_{c_{t+1}} := v_{c_1}$. This implies that every lifted duplication cycle in $\mathcal{C}$ with violated lifted duplication cycle inequality corresponds to some simple cycle in $\mathcal{G}_{\mathcal{C}}$ with a total edge weight strictly less than 1. Therefore, we identify a lifted duplication cycle in $\mathcal{C}$ with maximal violated lifted duplication cycle inequality by computing for every vertex $v \in \mathcal{G}_{\mathcal{C}}$, the shortest edge-weighted non-empty path to itself.

Obviously, the construction of $\mathcal{G}_{\mathcal{C}}$ and the computation of edge weights can be done in time $\mathcal{O}(|D|n^2)$ and space $\mathcal{O}(n^2)$. Since constraint (5.4) implies non-negativity of all edge weights, we can use Dijkstra's algorithm to compute the shortest paths. As graph $\mathcal{G}_{\mathcal{C}}$ has $\mathcal{O}(n)$ vertices, every call of Dijkstra's algorithm runs in time $\mathcal{O}(n^2)$. Therefore, by calling Dijkstra's algorithm once for each of the n vertices, the shortest cycle in $\mathcal{G}_{\mathcal{C}}$ can be computed in time $\mathcal{O}(n^3)$ and space $\mathcal{O}(n^2)$.

5.4.2 Maximal clique inequalities

The second class of valid inequalities, called *maximal clique inequalities*, is a generalization of constraints (5.2) from pairs of incompatible alignment edges to maximal cliques in the incompatibility graph $\mathcal{H}$. The definition and efficient separation is similar to the maximal clique inequalities defined by Althaus et al. [2005] for the multiple sequence alignment problem.

For the duplication-loss alignment problem, constraint (5.2) implies incompatibility between pairs of alignment edges while constraint (5.4) implies incompatibility between alignment edges, duplications and losses. Further, the duplication cycle constraints (5.3) can imply incompatibility between pairs of duplications if they constitute a duplication cycle of length 2. In the following, only maximal cliques in the incompatibility graph are considered that contain at least one alignment edge variable. Every such clique then corresponds to a set $K = K_E \cup K_D \cup K_L$, where $K_E \subseteq E$, $K_D \subseteq D^1 \cup D^2$ and $K_L \subseteq L^1 \cup L^2$, such that every two elements of this set are pairwise incompatible and $K_E \neq \emptyset$ (see Figure 5.4). If there exist no further alignment edges, duplications, or losses that are in conflict with every element of K, the clique is maximal.

Obviously, two duplications $d_1 \in D^1$ and $d_2 \in D^2$ cannot be pairwise incompatible, as they do not overlap or induce a duplication cycle. Hence, for every maximal clique K, either $K \cap D^1 = \emptyset$ or $K \cap D^2 = \emptyset$ holds.

Using this notation, the following maximal clique inequality is valid for P:

$$\sum_{e \in K_E} x_e + \sum_{d \in K_D} y_d + \sum_{l \in K_L} z_l \leq 1\,. \tag{5.9}$$

For every K with $K_L \neq \emptyset$, this constraint is trivially fulfilled with equality due to cover constraints (5.4). Therefore, we will restrict the following discussion about strength and separation to maximal clique inequalities that do not contain a loss variable, i.e., we assume $K_L = \emptyset$. The following theorem states that for every maximal clique with this property, the associated maximal clique inequality (5.9) is facet defining for P.

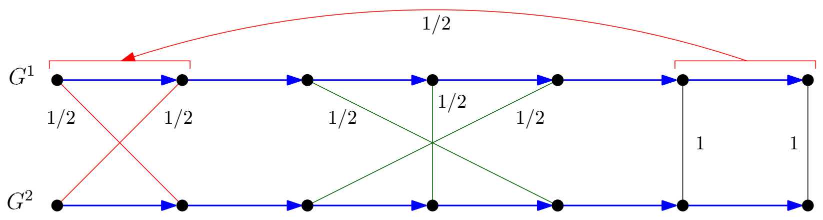


Figure 5.4: *Example of two violated maximal clique inequalities. Numbers denote values of the associated variables for the fractional solution. One maximal clique with associated violated inequality is given by the red alignment edges and the duplication event. A second maximal clique with violated associated inequality comprised of only alignment edges is given by the three green alignment edges. All cover constraints are fulfilled by loss variables wherever needed (not shown).*

Theorem 5.1. *For every maximal clique $K = K_E \cup K_D \cup K_L$ with $K_E \neq \emptyset$ and $K_L = \emptyset$, the maximal clique inequality* (5.9) *is facet defining for P.*

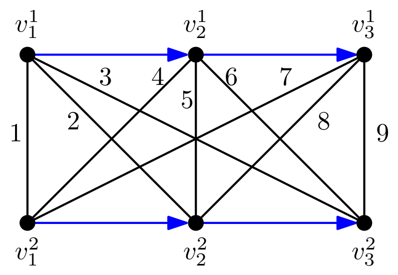
Proof. We use the *direct* method and show that every maximal clique inequality $\boldsymbol{\pi}^T(\mathbf{x}, \mathbf{y}, \mathbf{z}) \leq 1$ with $F := \{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in P \mid \pi^T(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 1\}$ defines a facet of P by constructing a set of $\dim(P) = |E| + |D^1| + |D^2|$ (Lemma 5.1) affinely independent points $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*) \in F$ as follows:

For every variable $k \in K_E \cup K_D$, we create a point where k is active, all remaining variables in K are inactive, and for all genes in G^1 and G^2 that are not covered by k, the corresponding loss variable is also active. These points are obviously in P, since no duplication cycle or alignment edge conflict can occur and all cover constraints are fulfilled.

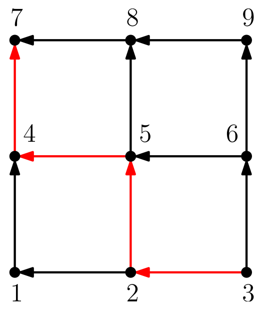
Since K is a maximal clique in $\mathcal{H}$, for every alignment edge $e \in E \setminus K$, there must exist some variable in K that is not incompatible to x_e. Otherwise, x_e would be incompatible to all variables in K, which would contradict the maximality of K. By the same argument, for every duplication variable $d \in D \setminus K$, there must exist some variable in K, such that the simultaneous activation of both variables does not induce a duplication cycle or violate any cover constraint. Therefore, for every alignment edge $e \in E \setminus K$ or duplication $d \in D \setminus K$, we can construct a feasible point by activating the corresponding variable together with exactly one non-incompatible variable in K. Again we activate the associated loss variables for all genes not covered by these two variables. Taken together, we obtain a set of $|D^1| + |D^2| + |E| = \dim(P)$ points that lie in F and are obviously affinely independent. As the activation of all loss variables (no active duplication or alignment edge variables) constitutes a feasible solution $(\mathbf{x}', \mathbf{y}', \mathbf{z}')$ with $\boldsymbol{\pi}^T(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0 < 1$, it follows that the maximal clique inequality defines a proper face of P, which finally proves the claim. $\square$

Separation of maximal clique inequalities

The separation problem of maximal clique inequalities can be solved in a similar fashion like for the multiple sequence alignment problem [Althaus et al., 2005]. As discussed above, the following separation procedure considers only maximal cliques containing no loss variable and at least one alignment edge variable.



(a) *alignment graph*



(b) *pairgraph*

Figure 5.5: *Example of a complete alignment graph for two genomes G^1 and G^2 of length three (a) and associated pairgraph (b). The pairgraph contains one vertex for every alignment edge in the complete alignment graph (vertex labels in (b) correspond to alignment edge labels in (a)). Every path in the pairgraph from the source (vertex 3) to the sink (vertex 7), e.g., $\langle 3, 2, 5, 4, 7\rangle$, represents a maximal clique of alignment edges in the incompatibility graph.*

Adopting notation from Althaus et al., let

$$\mathcal{E}(l_b \leftrightarrow l_e, m_b \leftrightarrow m_e)$$

denote the collection of all sets $S \subseteq E$ such that

(a) all edges in S are pairwise incompatible,

(b) for each edge $\{v_l^1, v_m^2\} \in S$, $l_b \leq l \leq l_e$ and $m_b \leq m \leq m_e$,

(c) S is maximal with respect to properties (a) and (b).

Similarly, for duplications we define

$$D^\ell(l \leftrightarrow m) := \{d \in D^\ell \mid [l \mathinner{..} m] \subseteq \text{target}(d) \wedge \ell \in \{1, 2\}\}\,.$$

Applying the same arguments as for the multiple sequence alignment problem, maximal cliques in $\mathcal{H}$ can be characterized by the following proposition:

Proposition 5.2. *A clique $K = K_E \cup K_D$ in $\mathcal{H}$ with sets $K_E \subseteq E$ (considering the complete alignment graph) and $K_D \subseteq D^1 \cup D^2$ is maximal if and only if*

$$K_E \in \mathcal{E}(l_b \leftrightarrow l_e, 1 \leftrightarrow |G^{\bar{\ell}}|), K_D = D^{\ell}(l_b \leftrightarrow l_e)\,,$$

for some $1 \leq l_b \leq l_e \leq |G^{\ell}|$, $\ell \in \{1,2\}, \bar{\ell} := 3 - \ell$.

Using above characterization of maximal cliques, we are now able to present a polynomial time algorithm to identify violated maximal clique inequalities. This algorithm separates maximal clique inequalities that involve duplications in $D := D^1$. A symmetric argument applies for cliques containing duplications in D^2. Let $n := |G^1|$ and $m := |G^2|$ and without loss of generality assume $m \leq n$. As described in [Althaus et al., 2005], the algorithm performs two steps to compute for all $1 \leq l_b < l_e \leq n$:

(a) $K_E \in \mathcal{E}(l_b \leftrightarrow l_e, 1 \leftrightarrow m)$ that maximizes $\sum_{e \in K_E} x^*_e$ and

(b) $\sum_{d \in D(l_b \leftrightarrow l_e)} y^*_d$.

The corresponding maximal clique inequality is violated if

$$\sum_{e \in K_E} x^*_e + \sum_{d \in D(l_b \leftrightarrow l_e)} y^*_d > 1\,.$$

Concerning (a), Reinert et al. [1997] introduced the pairgraph data structure. Given the alignment graph $\mathcal{G} = (V, E, H)$ for genomes G^1 and G^2, the corresponding pairgraph $\mathcal{G}_P = (\bar{V}, \bar{E})$ is a $n \times m$ directed grid graph. A vertex $\bar{v}_{p,q}$ in row p and column q of $\mathcal{G}_P$ corresponds to the (possibly missing) alignment edge connecting vertices v^1_p and v^2_q in $\mathcal{G}$. Every vertex $\bar{v}_{p,q}$ in $\mathcal{G}_P$ has up to two outgoing edges, one *vertical edge* to vertex $\bar{v}_{p+1,q}$ if $p < n$ and one *horizontal edge* to vertex $\bar{v}_{p,q-1}$ if $q > 1$. An example of a complete alignment graph with associated pairgraph is depicted in Figure 5.5.

In case of a non-complete alignment graph, only a subset of all vertices in $\mathcal{G}_P$, called *essential vertices*, corresponds to existing alignment edges in E. For every path $p = \langle \bar{v}_{l_b,m_e}, \ldots, \bar{v}_{l_e,m_b} \rangle$ in $\mathcal{G}_P$, the set of alignment edges associated to essential vertices in p corresponds to exactly one element of the set $\mathcal{E}(l_b \leftrightarrow l_e, m_b \leftrightarrow m_e)$. To identify the set $K_E \in \mathcal{E}(l_b \leftrightarrow l_e, 1 \leftrightarrow m)$ that maximizes $\sum_{e \in K_E} x^*_e$, we assign to every essential vertex $\bar{v}_{p,q}$ the weight $x^*_{\{v^1_p, v^2_q\}}$. Then, for each of the $n-1$ possible values of l_b, the longest path tree for source vertex $\bar{v}_{l_b,m}$ is computed using algorithm Dag-Longest-Path (Algorithm 1, see page 64), introduced in Section 3.2.3. Note that vertex weights can be easily transformed into edge weights such that algorithm DAG-LONGEST-PATH can be applied. Finally, the set $K_E \in \mathcal{E}(l_b \leftrightarrow l_e, 1 \leftrightarrow m)$ maximizing $\sum_{e \in K_E} x^*_e$ can be obtained by backtracking the longest path tree for source $\bar{v}_{l_b,m}$, starting at vertex $\bar{v}_{l_e,1}$. The weight of this path is exactly the value $\sum_{e \in K_E} x^*_e$. Since every longest path tree can be computed in time $\mathcal{O}(nm)$, the total time complexity of step (a) amounts to $\mathcal{O}(n^2 m)$.

The values (b) for all pairs $i := l_b$, $j := l_e$ can be computed in time $\mathcal{O}(n^2)$ by the following dynamic program: We define

$$D(i,j) := \{d \in D \mid \text{target}(d) = [i, .. \, j]\}, \quad \sigma_{i,j} := \sum_{d \in D(i \leftrightarrow j)} y^*_d, \quad \pi_{i,j} := \sum_{k=j}^{n} \sum_{d \in D(i,k)} y^*_d \,,$$

and observe that $\sigma_{i,j} = \sigma_{i-1,j} + \pi_{i,j}$. After computing the values $\pi_{p,q}$ for all $p = 1, \ldots, n$, $q = p, \ldots, n$ with the dynamic program outlined in Algorithm 3, we can use this recurrence to compute the matrix σ with another simple dynamic program depicted in Algorithm 4.

Algorithm 3 CALCULATE π

```
for $p = 1 \to n$ do
    $\pi_{p,n} = \sum_{d \in D(p,n)} y^*_d$
    for $q = n - 1 \to p$ do
        $\pi_{p,q} = \pi_{p,q+1} + \sum_{d \in D(p,q)} y^*_d$
    end for
end for
```

Algorithm 4 CALCULATE σ

```
for $q = 1 \to n$ do
    $\sigma_{1,q} = \pi_{1,q}$
end for
for $p = 2 \to n$ do
    for $q = p \to n$ do
        $\sigma_{p,q} = \sigma_{p-1,q} + \pi_{p,q}$
    end for
end for
```

Both dynamic programs have a time complexity of $\mathcal{O}(n^2)$, hence the time complexity to identify violated maximal clique inequalities is dominated by step (a) which is $\mathcal{O}(n^2 m)$. Since $m \leq n$, the separation problem of maximal clique inequalities can be solved in time $\mathcal{O}(n^3)$.

5.4.3 Duplication island inequalities

With the last class of constraints, referred to as *duplication island inequalities*, we can cut off fractional solutions, where every gene in a subset of one genome is exclusively covered by duplications originating from within this set, as displayed in Figure 5.6. As we show in Theorem 5.2, such a scenario implies a duplication cycle according to Definition 5.4 and hence cannot correspond to a valid solution. Therefore, in a valid labeled alignment, every set of genes must contain at least one gene that is either aligned, labeled as a loss, or labeled as a duplicate of some gene from outside the set. In the following, we consider only duplications in one of the genomes $G^\ell, \ell \in \{1,2\}$ and define $\bar{\ell} := 3 - \ell$, $n = |G^\ell|$, and $m = |G^{\bar{\ell}}|$. We present a formal definition and a separation algorithm for duplication island inequalities. Our separation algorithm is based on a graph $\hat{\mathcal{G}} = (V^1 \cup V^2, E, H, A)$, which we obtain by introducing a set of additional edges A into the alignment graph $\mathcal{G} = (V^1 \cup V^2, E, H)$ as follows:

For every duplication $d \in D^\ell$, we add an edge (u,v), where u is the vertex representing the i-th gene in origin(d) and v is the vertex representing the i-th gene in target(d), for all $i = 1, \dots, |\operatorname{origin}(d)|$. In addition, we define $\mathcal{D}(u,v)$ as the set of duplications d in D^ℓ that induce an edge $(u,v) \in A$. Now, for every set $S \subseteq V^\ell$, we denote by $\mathcal{D}(V^\ell \setminus S, S)$ the set of duplications that induce edges spanning the cut $(V^\ell \setminus S, S)$ in $\hat{\mathcal{G}}$, i.e.,:

$$\mathcal{D}(V^\ell \setminus S, S) := \bigcup_{\substack{(u,v) \in A: \\ u \in V^\ell \setminus S, v \in S}} \mathcal{D}(u,v)\,.$$

For every set $S \subseteq V^\ell$, we define the associated duplication island inequality as:

$$\sum_{v \in S} z_v + \sum_{v \in S} \sum_{k=1}^{m} x_{\{v, v_k^{\bar{\ell}}\}} + \sum_{d \in \mathcal{D}(V^\ell \setminus S, S)} y_d \geq 1\,. \tag{5.10}$$

The following theorem states that duplication island inequalities (5.10) are valid for P.

Theorem 5.2. *For every set $S \subseteq V^\ell$, the associated duplication island inequality* (5.10) *is valid for P.*

Proof. Assume, to the contrary, that the sum on the left hand side of inequality (5.10) is zero. We create graph $\hat{\mathcal{G}}'$ by removing from $\hat{\mathcal{G}}$ all alignment edges with associated x-variable having value zero and all edges $(u,v) \in A$ with $y_d = 0$, for all $d \in \mathcal{D}(u,v)$. Constraint (5.4) implies that every position in the genome must be covered. Since, by assumption $\sum_{v \in S} z_v + \sum_{v \in S} \sum_{k=1}^{m} x_{\{v, v_k^{\bar{\ell}}\}} = 0$, every vertex $v \in S$ must be incident to exactly one incoming edge in A. Due to the additional assumption $\sum_{d \in \mathcal{D}(V^\ell \setminus S, S)} y_d = 0$, each of these incoming edges must also originate at some vertex in S. Thus, after reverting all edges in $\hat{\mathcal{G}}'$, we can generate a walk of length greater than $|S|$ that starts at an arbitrary vertex in S and never leaves S. This implies a duplication cycle, and due to constraint (5.3) the corresponding solution must be infeasible. □

Separation of duplication island inequalities

We propose an algorithm to efficiently separate a slightly relaxed variant of constraint (5.10). Let $\alpha(d, S)$ denote the multiplicity of a duplication d in the cut-set of a cut $(V^\ell \setminus S, S)$, defined as:

$$\alpha(d,S) := |\{(u,v) \in A \mid u \in V^\ell \setminus S \wedge v \in S \wedge \mathcal{D}(u,v) \ni d\}| \,. \tag{5.11}$$

Now, we can define the relaxed variant of a duplication island inequality as follows:

$$\sum_{v \in S} z_v^* + \sum_{v \in S} \sum_{k=1}^{m} x^*_{\{v, v_k^{\bar\ell}\}} + \sum_{d \in \mathcal{D}(V \setminus S, S)} \alpha(d,S) \cdot y_d^* \geq 1 \,. \tag{5.12}$$

Our algorithm to determine whether a given (fractional) solution $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*) \in \mathbb{R}_+^{|E|+|D^1|+|D^2|+|V|}$ violates any relaxed duplication island inequality starts by selecting an arbitrary vertex $s \in V^\ell$. The next step is the construction of a graph $\mathcal{G}_s = (V^\ell, \tilde{A}, w)$ that contains the vertices for all genes in genome G^ℓ. Further, $\mathcal{G}_s$ has two subsets of directed edges $\tilde{A}_1$ and $\tilde{A}_2$ (i.e., $\tilde{A} = \tilde{A}_1 \cup \tilde{A}_2$), where $\tilde{A}_1$ contains an edge (u,v) of weight $w(u,v) := \sum_{d \in \mathcal{D}(u,v)} y_d^*$, for every pair of vertices $(u,v) \in V^\ell \times V^\ell$ with $\mathcal{D}(u,v) \neq \emptyset$. The subset $\tilde{A}_2$ contains an edge (s,v) of weight $w(s,v) := z_v^* + \sum_{k=1}^{m} x^*_{\{v, v_k^{\bar\ell}\}}$, for every $v \in V^\ell$ with $v \neq s$. According to the construction of graph $\mathcal{G}_s$, for every subset $S \subset V^\ell$ with $s \in V^\ell \setminus S$, the weight of the cut $(V^\ell \setminus S, S)$ in $\mathcal{G}_s$ equals the sum on the left

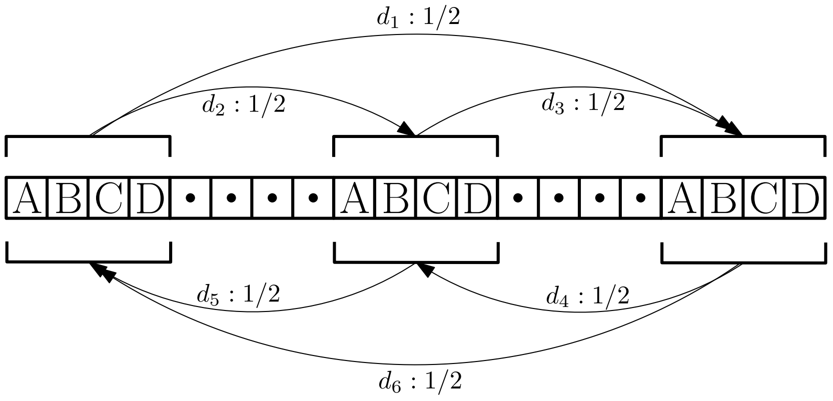


Figure 5.6: *A duplication island: The presented fractional values of the variables associated to duplications $d_1, \ldots, d_6$ define a feasible solution to the LP relaxation. In particular, no (lifted) duplication cycle inequality is violated. This solution is cut off by duplication island constraints.*

hand side of the relaxed duplication island inequality (5.12), as shown below:

$$\begin{aligned}
\sum_{\substack{(u,v)\in\tilde{A}_1\cup\tilde{A}_2:\\ u\in V^\ell\setminus S,\, v\in S}} w(u,v) &= \sum_{\substack{(u,v)\in\tilde{A}_1:\\ u\in V^\ell\setminus S,\, v\in S}} w(u,v) + \sum_{(s,v)\in\tilde{A}_2: v\in S} w(s,v)\\
&= \sum_{\substack{(u,v)\in\tilde{A}_1:\\ u\in V^\ell\setminus S,\, v\in S}} \sum_{d\in\mathcal{D}(u,v)} y_d^* + \sum_{v\in S}\left(z_v^* + \sum_{k=1}^{m} x^*_{\{v,v_k^{\bar{\ell}}\}}\right)\\
&\overset{(5.11)}{=} \sum_{d\in\mathcal{D}(V^\ell\setminus S,S)} \alpha(d,S)\cdot y_d^* + \sum_{v\in S} z_v^* + \sum_{v\in S}\sum_{k=1}^{m} x^*_{\{v,v_k^{\bar{\ell}}\}}\,.
\end{aligned}$$

Therefore, we can identify the set S^* that minimizes the left hand side of inequality (5.12) by computing a minimum s-t cut in $\mathcal{G}_s$, for all $s \in V^\ell$. We compute this minimum cut by selecting an arbitrary vertex $s \in V^\ell$ and solving $n-1$ maximum flow problems in $\mathcal{G}_s$ from source $s \in V^\ell$ to every other vertex in $V^\ell \setminus s$ and another $n-1$ maximum flow problems from every vertex $s' \in V^\ell \setminus s$ to vertex s in $\mathcal{G}_{s'}$. Each of the $2n-2$ maximum flow problems can be solved in time $\mathcal{O}(n^2\sqrt{|\tilde{A}|})$, using Goldberg-Tarjan's preflow push-relabel algorithm [Goldberg and Tarjan, 1988]. Since $|\tilde{A}| \le |D|n$, the complete separation algorithm runs in time $\mathcal{O}(n^{3.5}\sqrt{|D|})$ and space $\mathcal{O}(n^2)$.

5.5 The duplication-loss model for three species

In this section we propose an extension of our branch and cut algorithm to solve a slightly restricted version of the *median-of-three problem*. This problem is particularly interesting for the reconstruction of ancestral genomes in a phylogenetic tree based on the Steinerization method [Blanchette et al., 1997]. Starting from an initial assignment of ancestral genomes, this method iteratively improves the evolutionary history by traversing the phylogenetic tree and re-inferring ancestral genomes as the median of its immediate ancestor and its two descendants, until it reaches a local optimum.

We begin with a formal definition of the median-of-three problem considered throughout this section.

Definition 5.5. MEDIAN-OF-THREE PROBLEM
Let x be a vertex in the phylogenetic tree whose parent (immediate ancestor) is u and whose two children (immediate descendants) are v and w. Let $G(u), G(v)$ and $G(w)$ be the inferred or known genomes of $u, v,$ and w. Then, the median *problem is to infer the genome $G(x)$ minimizing*

$$c(G(u) \to G(x)) + c(G(x) \to G(v)) + c(G(x) \to G(w))\,.$$

In the remainder, we will refer to the genomes $G(u)$, $G(v)$, $G(w)$, and $G(x)$ as G^1, G^2, G^3, and G^m, respectively. A graphical representation of the median problem discussed throughout this section is depicted in Figure 5.7.

In contrast to the 2-SPP, every labeled alignment of G^1, G^2, and G^3 must correspond to an evolutionary history transforming G^1 into G^2 and G^3 along *two* generations.

Therefore, our duplication-loss model for the median-of-three problem employs additional constraints and variables to ensure that every solution corresponds to an evolutionary history and all induced duplications are consistent.

Since the inference of the evolutionary history from G^m to G^2 and G^3 represents an instance of the 2-SPP, we model it the same way using alignment variables $x_{\{v_i^2, v_j^3\}}$, loss variables $z_{v_j^i}$, $i \in \{2, 3\}$, and duplication variables y_d, $d \in D^2 \cup D^3$. To express the evolutionary history transforming G^1 into G^m, the model also contains variables:

- $x_{\{v_i^1, v_j^2\}}$ for alignment of genes $G^1[i]$ and $G^2[j]$,
- $x_{\{v_i^1, v_j^3\}}$ for alignment of genes $G^1[i]$ and $G^3[j]$,
- $\hat{y}_d$, $d \in D^m$ for duplication events in $O_{G^1 \to G^m}$,
- $\hat{z}_{v_j^1}$, $j = 1, \ldots, |G^1|$ for a loss of gene $G^1[j]$ in $O_{G^1 \to G^m}$.

Since we cannot directly determine a target region for a duplication $d \in D^m$ in the unknown genome G^m, we model d with origin in the ancestral genome G^1 and target in G^2 or G^3 (see duplication d in Figure 5.7).

This implies a limitation of our model, as it can only capture duplication events in the evolutionary history from G^1 to G^m if the source is present in G^1 and the target is conserved (visible) in either G^2 or G^3. Therefore, our model can only infer visible histories where all duplications fulfill this limitation, and hence, it solves only this restricted variant of the median-of-three problem optimally. However, this limitation does not render our ILP formulation infeasible on instances where the true evolutionary history contains such non-conserved duplications. Instead, the predicted median might just not be an optimal solution to the median-of-three problem. On the other hand, the model allows for a relaxation of the visible history restriction (see Definition 5.2) in a way that a duplication $d \in D^m$ can be inferred together with a subsequent modification of the origin of d during evolution from G^1 to G^m. This relaxation is possible, as the origin of such a duplication is fully preserved (and hence visible) in G^1, regardless of subsequent modifications.

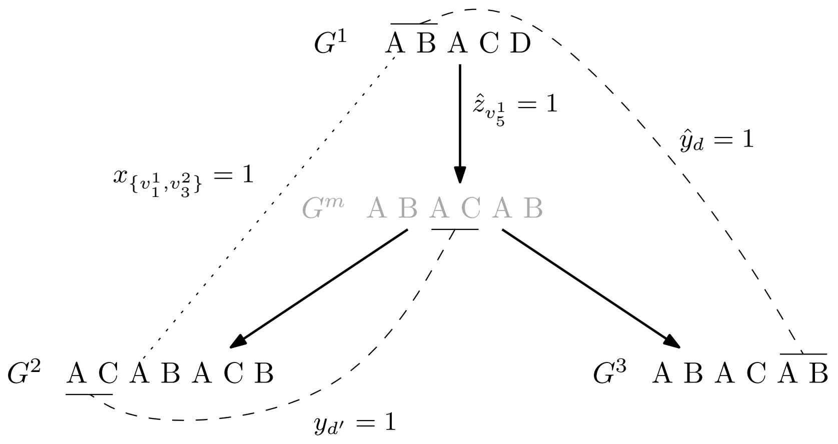


Figure 5.7: *Illustration of four different types of variables used to express the evolutionary history along two generations. Leftmost gene A in G^1 is retained in G^m, G^2, and G^3 (scenario (i)). Genes AB are duplicated from G^1 to G^m, modeled by duplication d with origin in G^1 and target in G^3 (scenario (ii)). The rightmost gene A in G^m is subsequently lost in G^2. Genes AC are duplicated from G^m to G^2 by duplication d (scenario (iii)). Gene D is lost from G^1 to G^m.*

5.5.1 Additional constraints

The pairwise alignment of genomes G^2 and G^3 is still correctly captured by constraints (5.2)-(5.7). Additionally, the initial occurrence of each gene in G^2 and G^3 must be modeled such that a gene $G^\ell[i]$, $\ell \in \{2, 3\}$ either

(i) has already been present in G^1 as $G^1[j]$,

(ii) arose from a duplication event $d \in D^m$ from G^1 to G^m, or

(iii) arose from a duplication event $d' \in D^2$ from G^m to G^ℓ.

Scenario (i) is expressed by an alignment of gene $G^\ell[i]$ to gene $G^1[j]$ and hence, $x_{\{v_j^1, v_i^2\}} = 1$. Scenario (iii) corresponds to a duplication event from the two genome model, and therefore this scenario is captured by setting $y_{d'} = 1$. Scenario *(ii)* can be captured in two ways, directly or indirectly. Assume a gene $G^\ell[i]$ that originates from gene $G^1[j]$, which was duplicated in the evolutionary history from G^1 to G^m. The direct way to capture this situation is to cover gene $G^\ell[i]$ by a duplication $d_1 \in D^m$ such that gene $G^1[j]$ lies in the origin of d_1 and is mapped to gene $G^\ell[i]$ in the target. Thus, the direct way is captured by setting $\hat{y}_{d_1} = 1$. The indirect way is to align gene $G^\ell[i]$ to some gene $G^{\bar{\ell}}[k]$, $\bar{\ell} := 4 - \ell$, which itself originates from $G^1[j]$ by a duplication $d_2 \in D^m$, directly captured by $\hat{y}_{d_2} = 1$. Therefore, the indirect way corresponds to setting $x_{\{v_i^\ell, v_k^{\bar{\ell}}\}} = 1$ and

$\hat{y}_{d_2} = 1$ simultaneously. To indicate this indirect relationship between gene $G^\ell[i]$ and gene $G^1[j]$, we introduce a new variable $\xi_{(v_i^\ell, v_k^{\bar{\ell}})}$ which corresponds to the product

$$x_{\{v_i^\ell, v_k^{\bar{\ell}}\}} \cdot \sum_{\substack{d \in D^m, \\ v_k^{\bar{\ell}} \in \text{target}(d)}} \hat{y}_d \, .$$

We express this product using the following three linear constraints:

$$\xi_{(v_i^\ell, v_k^{\bar{\ell}})} + 1 \geq x_{\{v_i^\ell, v_k^{\bar{\ell}}\}} + \sum_{\substack{d \in D^m, \\ v_k^{\bar{\ell}} \in \text{target}(d)}} \hat{y}_d \, , \qquad \xi_{(v_i^\ell, v_k^{\bar{\ell}})} \leq x_{\{v_i^\ell, v_k^{\bar{\ell}}\}} \qquad \text{and} \qquad \xi_{(v_i^\ell, v_k^{\bar{\ell}})} \leq \sum_{\substack{d \in D^m, \\ v_k^{\bar{\ell}} \in \text{target}(d)}} \hat{y}_d \, .$$

According to our model, for each gene $G^\ell[i]$ exactly one of the three cases (i), (ii), or (iii) must apply. Therefore, for every gene $G^\ell[i]$, our model contains the following constraint:

$$\sum_{j=1}^{|G^1|} x_{\{v_j^1, v_i^\ell\}} + \sum_{\substack{d \in D^m \\ v_i^\ell \in \text{target}(d)}} \hat{y}_d + \sum_{\substack{d \in D^\ell \\ v_i^\ell \in \text{target}(d)}} y_d + \sum_{j=1}^{|G^{\bar{\ell}}|} \xi_{(v_i^\ell, v_j^{\bar{\ell}})} = 1 \, . \tag{5.13}$$

For each gene in G^1 there exist only two possible cases. Either the gene was lost in the evolutionary history from G^1 to G^m or it is retained in G^m. The latter is only assumed if the gene also appears in at least one of G^2 or G^3, captured by a realized alignment edge. Therefore, for every gene $G^1[i]$, we introduce constraints

$$\hat{z}_{v_i^1} + \sum_{j=1}^{|G^2|} x_{\{v_i^1, v_j^2\}} + \sum_{j=1}^{|G^3|} x_{\{v_i^1, v_j^3\}} \geq 1 \, , \tag{5.14}$$

$$\hat{z}_{v_i^1} + \sum_{j=1}^{|G^2|} x_{\{v_i^1, v_j^2\}} \leq 1 \, , \tag{5.15}$$

$$\hat{z}_{v_i^1} + \sum_{j=1}^{|G^3|} x_{\{v_i^1, v_j^3\}} \leq 1 \, , \tag{5.16}$$

to ensure that exactly one of the two cases is realized. As a consequence, our model does not capture the scenario of a gene in G^1 that appears in G^m and is subsequently lost in both evolutionary histories from G^m to G^2 and G^3. However, since this scenario implies two loss operations, compared to a single loss operation in the evolutionary history from G^1 to G^m, it is always preferable for a median not to contain such a gene.

Further, we have to ensure consistency of the alignments between different pairs of genomes in the multiple alignment. In his polyhedral approach to the multiple sequence alignment problem, Reinert [1999] showed that a selection of alignment edges in the multiple sequence alignment is a trace (i.e., realizes a valid alignment) if and only if it does not induce a *mixed cycle* in the associated alignment graph $\mathcal{G} = (V, E, H)$. A mixed cycle in the alignment graph is defined as a cycle that contains at least one directed edge of set H. Building upon these results, Althaus et al. [2005] introduced the class of *lifted mixed cycle inequalities*, which dominate the mixed cycles inequalities and can be separated in time $\mathcal{O}(n_t^3)$, where n_t denotes the total length of all genomes.

Therefore, we add the violated lifted mixed cycle inequalities (5.17) to the model, which we separate as proposed by Althaus et al. [2005] (details omitted).

$$\sum_{\{v_i^j, v_l^k\} \in \mathcal{M}} x_{\{v_i^j, v_l^k\}} \leq |\mathcal{M} \cap E| - 1 \qquad \forall \text{ lifted mixed cycles } \mathcal{M} \in \mathcal{G} \tag{5.17}$$

In addition to mixed cycle inequalities, Althaus et al. [2005] also define a class of inequalities to enforce transitivity of alignment edges, meaning that for every pair of realized alignment edges $\{v_i^p, v_j^q\}$ and $\{v_j^q, v_k^r\}$, edge $\{v_i^p, v_k^r\}$ must be realized as well. Note, however, that for the considered case of alignment edges with zero cost and strictly positive cost for duplication and loss events, transitivity of the optimal solution is already implied after adding mixed cycle inequalities to the model.

Finally, we have to eliminate inconsistent selections of alignment edges and duplications $d \in D^m$ that imply non-consecutive target regions as depicted in Figure 5.8. This inconsistency is implied by simultaneous selection of a duplication $d \in D^m$ with target region $G^\ell[k \mathinner{..} l]$ and an alignment edge $\{v_i^1, v_j^{\bar{\ell}}\}$, such that gene $G^{\bar{\ell}}[j]$ lies between two genes $G^{\bar{\ell}}[j']$ and $G^{\bar{\ell}}[j'']$ that are both aligned to the interval $G^\ell[k \mathinner{..} l]$. Thus, $G^{\bar{\ell}}[j']$ and $G^{\bar{\ell}}[j'']$ are indirectly covered as products of the same duplication event in the evolution from G^1 to G^m. At the same time, the alignment edge $\{v_i^1, v_j^{\bar{\ell}}\}$ implies the conservation of gene $G^1[i]$ in the evolutionary history from G^1 to G^m where it would be located inside the target region of duplication d. This in turn implies the target region to be non-consecutive in G^m. Therefore, for every such pair of duplication $d \in D^m$ and alignment edge $\{v_i^1, v_j^{\bar{\ell}}\}$, we use the following inequality to cut off inconsistent solutions:

$$\sum_{r=1}^{j-1} x_{\{v_s^\ell, v_r^{\bar{\ell}}\}} + \sum_{r=j+1}^{|G^{\bar{\ell}}|} x_{\{v_t^\ell, v_r^{\bar{\ell}}\}} + \hat{y}_d + x_{\{v_i^1, v_j^{\bar{\ell}}\}} \leq 3 \qquad k \leq s < t \leq l\,. \tag{5.18}$$

5.6 Results

In the following two sections, we will evaluate the performance of our branch and cut approach for the Two Species Small Phylogeny problem as well as its

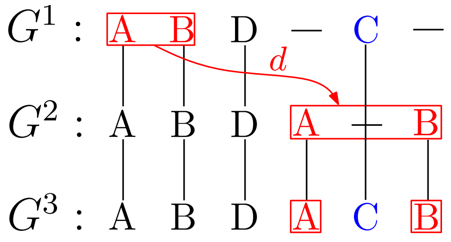


Figure 5.8: *Pair of inconsistent duplication $d \in D^m$ and alignment edge $\{v^1_4, v^3_5\}$. The presented solution corresponds to the median genome G^m = ABDACB, which implies a non consecutive duplication of AB enclosing the conserved gene C. Transitive alignment edges between G^1 and G^3 are not shown.*

extension to the MEDIAN-OF-THREE problem. All computations were performed on a machine equipped with 2 Intel Xeon CPU X5550 @2.67GHz Quad Core and 72 GB memory.

5.6.1 Two species duplication-loss alignment

First, we compare the practical performance of our branch and cut algorithm, referred to as DUPLOCUT, to the iterative ILP method by Holloway et al. [2012] We implemented DUPLOCUT in C++ using CPLEX (version 12.4) [IBM, 2011] as ILP solver, employing our problem specific cutting planes, and some graph data structures and graph algorithms provided by the SeqAn library [Döring et al., 2008], and the LEMON graph library [Dezs et al., 2011].

We compared our implementation to the implementation provided by the authors of Holloway et al. for two sets of real-world genome instances also provided by Holloway et al. The two sets contained the stable tRNA and rRNA contents of 10 *Bacillus* and 6 *Vibrionaceae* lineages that were preprocessed as described in [Holloway et al., 2012]. We applied both implementations to the set of all 45 *Bacillus* and 15 *Vibrionaceae* genome pairs. The average number of stable tRNA and rRNA genes was ∼120 for the *Bacillus* and ∼140 for the *Vibrionaceae* genomes. The average running time of the implementation by Holloway et al. on the *Bacillus* instances was around 69 seconds, while DUPLOCUT required less than 1.5 seconds; both implementations run with a single thread. For the set of *Vibrionaceae* pairs, DUPLOCUT was able to find the optimal solution within seconds on most instances using a single thread, whereas the implementation by Holloway et al. required a couple of hours when using up to 10 threads. For two pairs, it did not finish within two weeks of computation, whereas DUPLOCUT solved all instances in less than one hour.

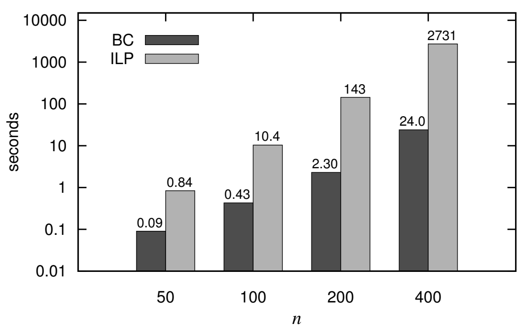
Simulation study

We performed a more detailed analysis on simulated data to evaluate the performance of the iterative ILP approach and DupLoCut for varying factors like genome length, evolutionary distance, and alphabet size. To eliminate side effects due to different programming language[3] and implementations of the ILP construction and the identification of violated duplication cycles, we compared DupLoCut against our own implementation of the iterative ILP approach. In addition, to reduce the immense running time of the iterative ILP approach, we used a slightly stronger formulation containing all lifted duplication cycle constraints for cycles of length two, and one duplication island constraint for every gene family g, containing all occurrences of g in the genome[4].

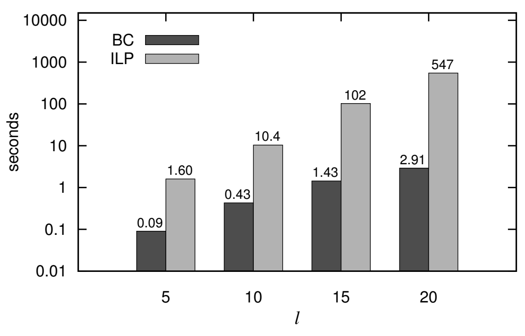
For the simulation of genome pairs, we followed the strategy of Holloway et al. and performed the following steps: First, we sampled a random sequence R of length n from an alphabet of size α, where the symbols at each position were independent and identically distributed. Second, we applied l random moves (single gene loss or duplication event) to R, where the length of every duplication was drawn from a Gaussian distribution with mean 5 and standard deviation 2. The starting positions of source and target regions of duplication events and the positions of loss events were uniformly distributed. The resulting sequence X was then used as ancestor genome, and two extant genomes were generated by applying l moves to X for each of them. Figure 5.9 shows average running times for different settings of parameters n, l, and α, with 200 simulated instances for each setting. For fixed ratios $2l/n = 0.2$ and $\alpha/n = 0.5$ (similar to the values observed in *Bacillus* data [Holloway et al., 2012]) and different genome lengths n, the running time of the iterative ILP increased drastically from less than one second for $n = 50$ to more than 2700 second for $n = 400$. While our branch and cut algorithm was roughly 10 times faster for length $n = 50$, the running time advantage increased to a factor of more than 100 for $n = 400$. For fixed values $n = 100$ and $\alpha = 50$, the running time of the iterative ILP increased from 1.6 seconds for $l = 5$ moves to 547 seconds for $l = 20$ moves. The running time advantage of the branch and cut algorithm increased from a factor of $\sim$18 for $l = 5$ to a factor of $\sim$188 for $l = 20$. The effect of decreasing the alphabet size, while keeping $n = 100$ and $l = 10$, was less pronounced for both algorithms. However, while halving the alphabet size from 60 to 30 caused the running time of the iterative ILP to increase by a factor of $\sim$16, the running time of the branch and cut algorithm exhibited only a modest increase by a factor of $\sim$3.6, yielding a running time advantage of factor $\sim$78 for $\alpha = 30$.

[3] Code provided by Holloway et al. written in Python

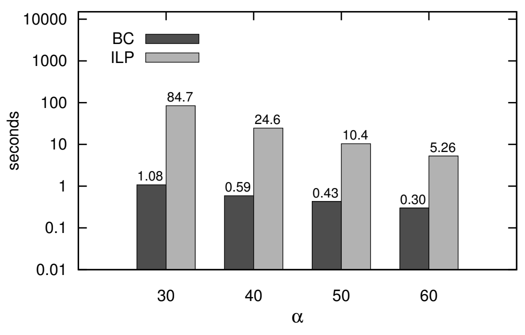
[4] We found similar constraints in the source code provided by Holloway et al.



***(a)** Effect of genome length*



***(b)** Effect of genome distance*



***(c)** Effect of alphabet size*

Figure 5.9: *Running time comparison between the iterative ILP method (ILP) and our branch and cut algorithm (BC) on simulated data. (a) Comparison for different genome lengths $n = 50, 100, 200, 400$ with ratios $2l/n$ and α/n fixed at 0.2 and 0.5. (b) Comparison for different number of moves $l = 5, 10, 15, 20$ using fixed $n = 100$ and $\alpha = 50$. (c) Comparison for different alphabet sizes $\alpha = 30, 40, 50, 60$ using fixed $n = 100$ and $l = 10$.*

5.6.2 Median-of-three

In addition to the running time analysis, we conducted another experiment to demonstrate how our median-of-three model can improve the quality of predicted ancestral genomes in a phylogenetic tree.

We generated 5 balanced phylogenetic trees, each with 128 extant species. For each internal node, the two descendant genomes were generated as in the pairwise alignment benchmark. First, we generated a random genome of length $n = 100$ and alphabet size $\alpha = 50$ at the root node. All other genomes along the tree were generated by applying 5 random moves (single gene loss or duplication event) to their immediate ancestor using the same model as in the previous section.

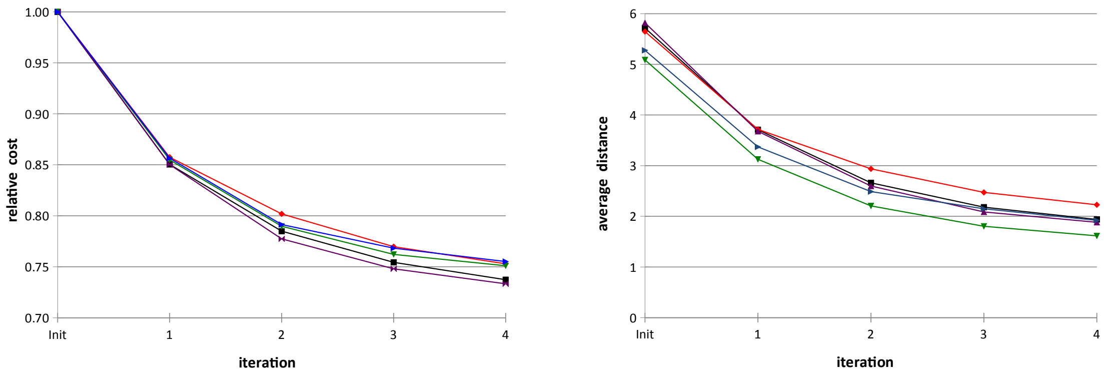
Following the Steinerization method, we first initialized the genomes of all internal nodes bottom up by computing the optimal labeled alignments of their two immediate descendants. Note that the so-called *SPP-heuristic* for the small phylogeny problem on a species tree, as proposed by Holloway et al., stops after this initialization step. Based on the initialized genomes, we applied 4 iterations of re-optimization where we replaced the genome of every internal node by the median of its three immediate neighbors. The root genome was re-estimated by computing the optimal labeled alignment of its two immediate descendants. Each round of re-optimization was also performed as a bottom-up traversal and up to 16 independent vertices at the same depth were processed in parallel. The running times for each tree were between 60 and 180 minutes.

Due to the limitation of our model, as discussed in Section 5.5, a newly derived median can have a higher cost than the solution from the previous iteration if, in the current iteration, the target of some duplication from G^1 to G^m has been disrupted in the descendent genomes. Deviating from the standard procedure, we replaced a genome by the newly computed median even if it implied a higher cost. Using this strategy, in our experiments we obtained more parsimonious solutions and a higher prediction accuracy. A possible explanation for this observation is that by our strategy we reduced the risk of getting stuck in a local minimum. The total cost of the complete tree after each iteration, with respect to the cost after initialization, is shown in Figure 5.10(a). For all 5 trees the total cost was reduced by $\sim$25% after 4 iterations of re-optimization.

To analyze whether the cost reduction also implies a more accurate reconstruction of ancestral genomes, we compared all predicted genomes to the true genomes after each iteration. For this comparison, we used the cost of an optimal duplication-loss alignment as a measure of distance. The average distance of all ancestral genomes after each iteration is depicted in Figure 5.10(b). Note that the total distance after initialization (*init*) corresponds to the (final) solution of the SPP-heuristic, as proposed by Holloway et al. For all trees, the average distance of the predicted genomes to the true genomes could be reduced by more than 60%.

5.7 Conclusion

In this chapter, we presented our approach to the duplication-loss alignment problem based on an efficient branch and cut algorithm, involving problem specific cutting planes and separation algorithms. In a running time benchmark on real and simulated data, we could demonstrate that our branch and cut algorithm outperforms the existing ILP-based approach by several orders of magnitude and that this running time advantage increases with the length and the evolutionary distance of the input genomes. Even for instances that are much harder than



(a) *Total cost* ***(b)*** *Average distance*

Figure 5.10: *Results of Steinerization method with median-of-three model on five simulated phylogenetic trees with 128 extant species (a) Total cost (number of implied duplication and loss operations for complete tree) after each iteration of re-optimization, normalized by the total cost after initialization. (b) Average distance (cost of optimal duplication-loss alignment) of all predicted genomes to the true genomes after each iteration.*

the *Bacillus* stable tRNA and rRNA content studied in [Holloway et al., 2012], our algorithm can compute optimal labeled alignments within a few seconds on a desktop PC. Hence, our algorithm offers the possibility of study more complex genomes under the duplication-loss model of evolution without need for a compute cluster.

In addition, the substantial performance improvement allowed us to formulate an extension of the model to three species. In a proof of concept study, we used this formulation with the Steinerization heuristic for the small phylogeny problem on simulated trees with 128 extant species. After only a few iterations, we observed a considerable cost reduction for the complete trees and more accurate predictions of ancestral genomes, compared to the SPP-heuristic proposed by Holloway et al. These results suggest that considering more than only two genomes at a time allows for much more accurate reconstructions of evolutionary histories. Hence, this approach can further improve our understanding of the evolutionary consequences and mechanisms of gene duplications and losses.

There are some open questions to be addressed in the future like, e.g., whether and under which conditions the duplication island inequalities are facet defining for our duplication-loss alignment ILP formulation. Another issue to be addressed is the computational complexity of our median-of-three formulation as it does not trivially follow from the hardness of the pairwise duplication-loss alignment problem.

Further, our formulation allows for several extensions to capture operations like inverted duplications and non-overlapping inversions. This requires a closer analysis on how much these extensions change the structure of the ILP formulation, how we must adapt our separation procedures, and whether we can identify new valid inequalities for the modified formulation. In addition, the possibility to assign individual costs to alignment edges, duplications, and losses, facilitates the introduction of prior knowledge about probabilities of particular evolutionary operations into the model. Moreover, due to the performance gain of our algorithm, it might be also possible to relax the visual history condition by considering also duplication events with a limited number of subsequent loss events within their origin or target. As in this case the origin and target are still present as contiguous regions the the genome, this involves only an increased number of duplication variables, each weighted by a score reflecting the number of implied loss events. This relaxation of the visible history restriction might be particularly useful to capture the frequently observed scenario of one gene copy being lost shortly after a duplication event.

Chapter 6

Closing Remarks

In this thesis, we demonstrated how linear programming and integer linear programming can be applied to solve important problems related to bioinformatics and computational biology, for which no efficient polynomial time algorithms are known. We could show that our approaches are capable of solving biologically relevant instances sufficiently fast for practical use and that they are at least competitive to existing state-of-the-art approaches.

While the algorithms presented in the previous chapters already tackle the most important use cases, another advantageous feature of our approaches is their flexibility and adaptivity to solve modified variants of the problem. Such adaptions can be used, for example, to introduce additional biological knowledge into the algorithm or to capture specific properties of the analyzed data. For each of our approaches, we already pointed out several interesting directions for future extensions and adaptions that can be realized with relatively low effort. Therefore, for each of the three problems, our approach is not merely just a final solution to the defined problem, but rather a good starting point also for other researchers to build upon in order to tackle related problems or even problems from different scientific domains.

While the success of our approaches is predominantly based on the algorithmic formulations, we also attached great importance to the implementation side by writing efficient, stable, and portable C++ code and always using appropriate data structures and algorithms for all data handling and manipulation tasks.

Finally, we remark that although the three studied problems originate from different research fields, recently emerging *multi-omics* experiments, integrating genome, transcriptome, and proteome data, will probably demand for combined algorithmic solutions to bridge the gap between peptide identification and isoform inference and abundance estimation in the near future. Again, for this task, the flexibility and extensibility of our algorithms may provide a major advantage over other existing methods.

Part III

APPENDIX

Appendix A Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Sandro Andreotti
February 4, 2015

Appendix B Curriculum Vitae

For privacy reasons, the curriculum vitae is not contained in the online version.

For privacy reasons, the curriculum vitae is not contained in the online version.

BIBLIOGRAPHY

Adams, M. D., Kelley, J. M., Gocayne, J. D., Dubnick, M., Polymeropoulos, M. H., Xiao, H., Merril, C. R., Wu, A., Olde, B., Moreno, R. F., et al. (1991). Complementary DNA sequencing: expressed sequence tags and human genome project. *Science*, 252(5013):1651–1656.

Aebersold, R. and Mann, M. (2003). Mass spectrometry-based proteomics. *Nature*, 422(6928):198–207.

Alberts, B. (2007). *Molecular Biology of the Cell.* Garland Science, New York, 5th edition.

Althaus, E., Caprara, A., Lenhof, H.-P., and Reinert, K. (2005). A branch-and-cut algorithm for multiple sequence alignment. *Mathematical Programming*, 105(2-3):387–425.

Althaus, E., Kohlbacher, O., Lenhof, H.-P., and Müller, P. (2002). A combinatorial approach to protein docking with flexible side chains. *Journal of Computational Biology*, 9(4):597–612.

Andonov, R., Malod-Dognin, N., and Yanev, N. (2011). Maximum contact map overlap revisited. *Journal of Computational Biology*, 18(1):27–41.

Andonov, R., Yanev, N., and Malod-Dognin, N. (2008). An efficient Lagrangian relaxation for the contact map overlap problem. In *Proceedings of the 8th International Workshop on Algorithms in Bioinformatics (WABI)*, pages 162–173. Springer-Verlag.

Andreotti, S. (2008). Fast de novo sequencing with mathematical programming. Master's thesis, Freie Universität Berlin, Germany.

Andreotti, S., Klau, G. W., and Reinert, K. (2012). Antilope - A Lagrangian relaxation approach to the de novo peptide sequencing problem. *IEEE Transactions on Computational Biology and Bioinformatics*, 9(2):385–394.

Andreotti, S., Reinert, K., and Canzar, S. (2013). The duplication-loss small phylogeny problem: from cherries to trees. *Journal of Computational Biology*, 20(9):643–59.

Backes, C., Rurainski, A., Klau, G. W., Müller, O., Stöckel, D., Gerasch, A., Küntzer, J., Maisel, D., Ludwig, N., Hein, M., Keller, A., Burtscher, H., Kaufmann, M., Meese, E., and Lenhof, H.-P. (2012). An integer linear programming approach for finding deregulated subgraphs in regulatory networks. *Nucleic Acids Research*, 40(6):e43.

Bafna, V. and Edwards, N. (2003). On de novo interpretation of tandem mass spectra for peptide identification. In *Proceedings of the Seventh Annual International Conference on Computational Molecular Biology, RECOMB*, pages 9–18. ACM Press.

Balev, S. (2004). Solving the protein threading problem by lagrangian relaxation. *Lecture Notes in Computer Science*, 2003:182–193.

Bartels, C. (1990). Fast algorithm for peptide sequencing by mass spectroscopy. *Biological Mass Spectrometry*, 19(6):363–368.

Bauer, M., Klau, G. W., and Reinert, K. (2005). Multiple structural RNA alignment with Lagrangian relaxation. In *Proceedings of the 5th Workshop on Algorithms in Bioinformatics (WABI)*, pages 303–314.

Bauer, M., Klau, G. W., and Reinert, K. (2007). Accurate multiple sequence-structure alignment of RNA sequences using combinatorial optimization. *BMC Bioinformatics*, 8(1):271.

Beadle, G. W. (1945). Biochemical Genetics. *Chemical Reviews*, 37(1):15–96.

Behr, J., Kahles, A., Zhong, Y., Sreedharan, V. T., Drewe, P., and Rätsch, G. (2013). MITIE: simultaneous RNA-Seq-based transcript identification and quantification in multiple samples. *Bioinformatics (Oxford, England)*, 29(20):2529–38.

Benjamini, Y. and Speed, T. P. (2012). Summarizing and correcting the GC content bias in high-throughput sequencing. *Nucleic Acids Research*.

Benzaid, B., Dondi, R., and El-Mabrouk, N. (2013). Duplication-loss genome alignment: complexity and algorithm. In *LATA*, volume 7810 of *Lecture Notes in Computer Science*, pages 116–127. Springer Berlin Heidelberg.

Berget, S. M., Moore, C., and Sharp, P. A. (1977). Spliced segments at the 5' terminus of adenovirus 2 late mRNA. *Proceedings of the National Academy of Sciences of the United States of America*, 74(8):3171–3175.

Bern, M. and Goldberg, D. (2006). De novo analysis of peptide tandem mass spectra by spectral graph partitioning. *Journal of Computational Biology*, 13(2):364–378.

Bertsch, A., Jung, S., Zerck, A., Pfeifer, N., Nahnsen, S., Henneges, C., Nordheim, A., and Kohlbacher, O. (2010). Optimal de novo design of MRM experiments for rapid assay development in targeted proteomics. *Journal of Proteome Research*, 9(5):2696–704.

Bertsch, A., Leinenbach, A., Pervukhin, A., Lubeck, M., Hartmer, R., Baessmann, C., Elnakady, Y. A., Müller, R., Böcker, S., Huber, C. G., and Kohlbacher, O. (2009). De novo peptide sequencing by tandem MS using complementary CID and electron transfer dissociation. *Electrophoresis*, 30(21):3736–3747.

Bertsimas, D. and Tsitsiklis, J. (1997). *Introduction to Linear Optimization*. Athena Scientific, 1st edition.

Blanchette, M., Bourque, G., and Sankoff, D. (1997). Breakpoint phylogenies. In *Genome Informatics*, pages 25–34. Univ. Academy Press.

Bland, R. G. (1977). New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2(2):103–107.

Blomme, T., Vandepoele, K., De Bodt, S., Simillion, C., Maere, S., and Van de Peer, Y. (2006). The gain and loss of genes during 600 million years of vertebrate evolution. *Genome Biology*, 7(5):R43.

Böck, A., Forchhammer, K., Heider, J., and Baron, C. (1991). Selenoprotein synthesis: an expansion of the genetic code. *Trends in Biochemical Sciences*, 16:463–467.

Bohnert, R. and Rätsch, G. (2010). rquant.web: a tool for RNA-Seq-based transcript quantitation. *Nucleic Acids Research*, 38(suppl 2):W348–W351.

Bouckaert, R. R. (2004). Bayesian network classifiers in weka. Technical report, University of Waikato, Department of Computer Science, Hamilton, New Zealand.

Brenner, S., Johnson, M., Bridgham, J., Golda, G., Lloyd, D. H., Johnson, D., Luo, S., McCurdy, S., Foy, M., Ewan, M., Roth, R., George, D., Eletr, S., Albrecht, G., Vermaas, E., Williams, S. R., Moon, K., Burcham, T., Pallas, M., DuBridge, R. B., Kirchner, J., Fearon, K., Mao, J., and Corcoran, K. (2000). Gene expression analysis by massively parallel signature sequencing (MPSS) on microbead arrays. *Nature Biotechnology*, 18(6):630–4.

Brett, D., Hanke, J., Lehmann, G., Haase, S., Delbrück, S., Krueger, S., Reich, J., and Bork, P. (2000). EST comparison indicates 38% of human mRNAs contain possible alternative splice forms. *FEBS Letters*, 474(1):83–86.

Canzar, S., Toussaint, N. C., and Klau, G. W. (2011). An exact algorithm for side-chain placement in protein design. *Optimization Letters*, 5(3):393–406.

Caprara, A. and Lancia, G. (2002). Structural alignment of large-size proteins via Lagrangian relaxation. In *Proceedings of the Sixth Annual International Conference on Computational Biology*, RECOMB, pages 100–108, New York, NY, USA. ACM.

Cawley, S., Bekiranov, S., Ng, H. H., Kapranov, P., Sekinger, E. A., Kampa, D., Piccolboni, A., Sementchenko, V., Cheng, J., Williams, A. J., Wheeler, R., Wong, B., Drenkow, J., Yamanaka, M., Patel, S., Brubaker, S., Tammana, H., Helt, G., Struhl, K., and Gingeras, T. R. (2004). Unbiased mapping of transcription factor binding sites along human chromosomes 21 and 22 points to widespread regulation of noncoding RNAs. *Cell*, 116(4):499–509.

Chen, M. and Manley, J. L. (2009). Mechanisms of alternative splicing regulation: insights from molecular and genomics approaches. *Nature Reviews. Molecular Cell Biology*, 10(11):741–754.

Chen, P., Lepikhova, T., Hu, Y., Monni, O., and Hautaniemi, S. (2011). Comprehensive exon array data processing method for quantitative analysis of alternative spliced variants. *Nucleic Acids Research*, 39(18):e123.

Chen, T., Kao, M. Y., Tepel, M., Rush, J., and Church, G. M. (2001). A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. *Journal of Computational Biology*, 8(3):325–337.

Chepelev, I., Wei, G., Tang, Q., and Zhao, K. (2009). Detection of single nucleotide variations in expressed exons of the human genome using RNA-Seq. *Nucleic Acids Research*, 37(16):e106.

Chi, H., Chen, H., He, K., Wu, L., Yang, B., Sun, R.-X., Liu, J., Zeng, W.-F., Song, C.-Q., He, S.-M., and Dong, M.-Q. (2013). pNovo+: de novo peptide sequencing using complementary HCD and ETD tandem mass spectra. *Journal of Proteome Research*, 12(2):615–25.

Chvátal, V. (1983). *Linear Programming*. W. H. Freeman and Company, New York.

Clark, T. A., Schweitzer, A. C., Chen, T. X., Staples, M. K., Lu, G., Wang, H., Williams, A., and Blume, J. E. (2007). Discovery of tissue-specific exons using comprehensive human exon microarrays. *Genome Biology*, 8(4):R64.

Collet, G., Andonov, R., Yanev, N., and Gibrat, J.-F. (2011). Local protein threading by mixed integer programming. *Discrete Applied Mathematics*, 159(16):1707–1716.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. The MIT Press, 2nd edition.

Cotton, J. A. and Page, R. D. M. (2005). Rates and patterns of gene duplication and loss in the human genome. *Proceedings of the Royal Society B: Biological Sciences*, 272(1560):277–283.

Croft, L., Schandorff, S., Clark, F., Burrage, K., Arctander, P., and Mattick, J. S. (2000). ISIS, the intron information system, reveals the high frequency of alternative splicing in the human genome. *Nature Genetics*, 24(4):340–1.

Dančík, V., Addona, T. A., Clauser, K. R., Vath, J. E., and Pevzner, P. (1999). De novo protein sequencing via tandem mass-spectrometry. *Journal of Computational Biology*, 6:327–341.

Datta, R. and Bern, M. (2009). Spectrum fusion: using multiple mass spectra for de novo peptide sequencing. *Journal of Computational Biology*, 16(8):1169–1182.

Davuluri, R., Suzuki, Y., Sugano, S., Plass, C., and Huang, T. (2008). The functional consequences of alternative promoter use in mammalian genomes. *Trends in Genetics*, 24(4):167–177.

de Queirós Vieira Martins, E., Pascoal, M. M. B., and Santos, J. L. E. D. (1999). Deviation algorithms for ranking shortest paths. *International Journal of Foundations of Computer Science*, 10(3):247–262.

Derti, A., Garrett-Engele, P., Macisaac, K. D., Stevens, R. C., Sriram, S., Chen, R., Rohl, C. A., Johnson, J. M., and Babak, T. (2012). A quantitative atlas of polyadenylation in five mammals. *Genome Research*, 22(6):1173–1183.

Dezs, B., Jüttner, A., and Kovács, P. (2011). Lemon - an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45.

Di Giammartino, D. C., Nishida, K., and Manley, J. L. (2011). Mechanisms and consequences of alternative polyadenylation. *Molecular Cell*, 43(6):853–866.

DiMaggio, P. A. and Floudas, C. A. (2007). De novo peptide identification via tandem mass spectrometry and integer linear optimization. *Analytical Chemistry*, 79(4):1433–1446.

Dittrich, M. T., Klau, G. W., Rosenwald, A., Dandekar, T., and Müller, T. (2008). Identifying functional modules in protein-protein interaction networks: an integrated exact approach. *Bioinformatics (Oxford, England)*, 24(13):i223–31.

Döring, A., Weese, D., Rausch, T., and Reinert, K. (2008). SeqAn An efficient, generic C++ library for sequence analysis. *BMC Bioinformatics*, 9(1):11.

Douglas, A. G. L. and Wood, M. J. A. (2011). RNA splicing: disease and therapy. *Briefings in Functional Genomics*, 10(3):151–64.

Eichner, J., Zeller, G., Laubinger, S., and Rätsch, G. (2011). Support vector machines-based identification of alternative splicing in *Arabidopsis thaliana* from whole-genome tiling arrays. *BMC Bioinformatics*, 12:55.

Eng, J. K., McCormack, A. L., and Yates, J. R. (1994). An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society for Mass Spectrometry*, 5(11):976–989.

Eppstein, D. (1998). Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673.

Eriksson, O., Zhou, Y., and Elofsson, A. (2001). Side chain-positioning as an integer programming problem. In *Proceedings of the First International Workshop on Algorithms in Bioinformatics*, pages 128–141. Springer.

Fagnani, M., Barash, Y., Ip, J. Y., Misquitta, C., Pan, Q., Saltzman, A. L., Shai, O., Lee, L., Rozenhek, A., Mohammad, N., Willaime-Morawek, S., Babak, T., Zhang, W., Hughes, T. R., van der Kooy, D., Frey, B. J., and Blencowe, B. J. (2007). Functional coordination of alternative splicing in the mammalian central nervous system. *Genome Biology*, 8(6):R108.

Feng, J., Li, W., and Jiang, T. (2010). Inference of isoforms from short sequence reads. In *Research in Computational Molecular Biology*, volume 6044 of *Lecture Notes in Computer Science*, pages 138–157. Springer Berlin Heidelberg.

Fenn, J., Mann, M., Meng, C., Wong, S., and Whitehouse, C. (1989). Electrospray ionization for mass spectrometry of large biomolecules. *Science*, 246(4926):64–71.

Fertin, G., Labarre, A., Rusu, I., Tannier, E., and Vialette, S. (2009). *Combinatorics of Genome Rearrangements*. Computational Molecular Biology. MIT Press.

Fischer, B., Roth, V., Roos, F., Grossmann, J., Baginsky, S., Widmayer, P., Gruissem, W., and Buhmann, J. M. (2005). NovoHMM: a hidden markov model for de novo peptide sequencing. *Analytical Chemistry*, 77(22):7265–7273.

Fisher, M. L. (1985). An applications oriented guide to Lagrangian relaxation. *Interfaces*, 15(2):10–21.

Forrest, A. R. and Carninci, P. (2009). Whole genome transcriptome analysis. *RNA Biology*, 6(2):107–112.

Frank, A. and Pevzner, P. (2005). PepNovo: de novo peptide sequencing via probabilistic network modeling. *Analytical Chemistry*, 77(4):964–973.

Frank, A. M. (2009). A ranking-based scoring function for peptide-spectrum matches. *Journal of Proteome Research*, 8(5):2241–52.

Friedman, J., Hastie, T., Höfling, H., and Tibshirani, R. (2007). Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22.

Gabow, H. N., Maheshwari, S. N., and Osterweil, L. J. (1976). On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, 2(3):227–231.

Garcia-Blanco, M. A., Baraniak, A. P., and Lasda, E. L. (2004). Alternative splicing in disease and therapy. *Nature Biotechnology*, 22(5):535–546.

Geer, L. Y., Markey, S. P., Kowalak, J. A., Wagner, L., Xu, M., Maynard, D. M., Yang, X., Shi, W., and Bryant, S. H. (2004). Open mass spectrometry search algorithm. *Journal of Proteome Research*, 3(5):958–964.

Goldberg, A. V. and Tarjan, R. E. (1988). A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940.

Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society*, 64:275–278.

Greenberg, A. J., Moran, J. R., Fang, S., and Wu, C.-I. (2006). Adaptive loss of an old duplicated gene during incipient speciation. *Molecular Biology and Evolution*, 23(2):401–410.

Griebel, T., Zacher, B., Ribeca, P., Raineri, E., Lacroix, V., Guigó, R., and Sammeth, M. (2012). Modelling and simulating generic RNA-Seq experiments with the flux simulator. *Nucleic Acids Research*, 40(20):10073–83.

Grötschel, M., Lovász, L., and Schrijver, A. (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197.

Guttman, M., Garber, M., Levin, J. Z., Donaghey, J., Robinson, J., Adiconis, X., Fan, L., Koziol, M. J., Gnirke, A., Nusbaum, C., Rinn, J. L., Lander, E. S., and Regev, A. (2010). Ab initio reconstruction of cell type-specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs. *Nature Biotechnology*, 28(5):503–10.

Haas, B. J., Papanicolaou, A., Yassour, M., Grabherr, M., Blood, P. D., Bowden, J., Couger, M. B., Eccles, D., Li, B., Lieber, M., Macmanes, M. D., Ott, M., Orvis, J., Pochet, N., Strozzi, F., Weeks, N., Westerman, R., William, T., Dewey, C. N., Henschel, R., Leduc, R. D., Friedman, N., and Regev, A. (2013). De novo transcript sequence reconstruction from RNA-Seq using the Trinity platform for reference generation and analysis. *Nature protocols*, 8(8):1494–512.

Hahn, M. W., Han, M. V., and Han, S. G. (2007). Gene family evolution across 12 Drosophila genomes. *PLoS Genetics*, 3(11).

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11:10–18.

Hannenhalli, S. and Pevzner, P. A. (1995). Transforming men into mice (polynomial algorithm for genomic distance problem). In *36th Annual IEEE Symposium on Foundations of Computer Science*, pages 581–592.

Hannenhalli, S. and Pevzner, P. A. (1999). Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1–27.

Hansen, K. D., Brenner, S. E., and Dudoit, S. (2010). Biases in Illumina transcriptome sequencing caused by random hexamer priming. *Nucleic Acids Research*, 38(12):e131.

Hardcastle, T. J. and Kelly, K. A. (2010). baySeq: empirical Bayesian methods for identifying differential expression in sequence count data. *BMC Bioinformatics*, 11:422.

Heber, S., Alekseyev, M., Sze, S. H., Tang, H., and Pevzner, P. A. (2002). Splicing graphs and EST assembly problem. *Bioinformatics (Oxford, England)*, 18 Suppl 1:S181–S188.

Hillenkamp, F., Karas, M., Beavis, R. C., and Chait, B. T. (1991). Matrix-assisted laser desorption/ionization mass spectrometry of biopolymers. *Analytical Chemistry*, 63(24):1193A–1203A.

Holloway, P., Swenson, K., Ardell, D., and El-Mabrouk, N. (2012). Evolution of genome organization by duplication and loss: an alignment approach. In *Research in Computational Molecular Biology*, volume 7262 of *Lecture Notes in Computer Science*, pages 94–112. Springer Berlin Heidelberg.

Huang, S., Zhang, J., Li, R., Zhang, W., He, Z., Lam, T.-W., Peng, Z., and Yiu, S.-M. (2011). SOAPsplice: genome-wide ab initio detection of splice junctions from RNA-Seq data. *Frontiers in genetics*, 2:46.

Huerta, M., Haseltine, F., Liu, Y., Downing, G., and Seto, B. (2000). NIH Working Definition of Bioinformatics and Computational Biology.

IBM (2011). IBM ILOG CPLEX Optimization Studio V12.4.

Jean, G., Kahles, A., Sreedharan, V. T., De Bona, F., and Rätsch, G. (2010). RNA-Seq read alignments with PALMapper. *Current Protocols in Bioinformatics*, Chapter 11.

Jia, W., Qiu, K., He, M., Song, P., Zhou, Q., Zhou, F., Yu, Y., Zhu, D., Nickerson, M. L., Wan, S., Liao, X., Zhu, X., Peng, S., Li, Y., Wang, J., and Guo, G. (2013). SOAPfuse: an algorithm for identifying fusion transcripts from paired-end RNA-Seq data. *Genome Biology*, 14(2):R12.

Johnson, J. M., Castle, J., Garrett-Engele, P., Kan, Z., Loerch, P. M., Armour, C. D., Santos, R., Schadt, E. E., Stoughton, R., and Shoemaker, D. D. (2003). Genome-wide survey of human alternative pre-mRNA splicing with exon junction microarrays. *Science*, 302(5653):2141–4.

Kampa, D., Cheng, J., Kapranov, P., Yamanaka, M., Brubaker, S., Cawley, S., Drenkow, J., Piccolboni, A., Bekiranov, S., Helt, G., Tammana, H., and Gingeras, T. R. (2004). Novel RNAs identified from an in-depth analysis of the transcriptome of human chromosomes 21 and 22. *Genome Research*, 14(3):331–42.

Kan, Z., Rouchka, E. C., Gish, W. R., and States, D. J. (2001). Gene structure prediction and alternative splicing analysis using genomically aligned ESTs. *Genome Research*, 11(5):889–900.

Kapranov, P., Cawley, S. E., Drenkow, J., Bekiranov, S., Strausberg, R. L., Fodor, S. P. A., and Gingeras, T. R. (2002). Large-scale transcriptional activity in chromosomes 21 and 22. *Science*, 296(5569):916–9.

Kato, Y., Sato, K., Hamada, M., Watanabe, Y., Asai, K., and Akutsu, T. (2010). RactIP: fast and accurate prediction of RNA-RNA interaction using integer programming. *Bioinformatics (Oxford, England)*, 26(18):i460–6.

Katz, Y., Wang, E. T., Airoldi, E. M., and Burge, C. B. (2010). Analysis and design of RNA sequencing experiments for identifying isoform regulation. *Nature Methods*, 7(12):1009–15.

Kececioglu, J. and Kim, E. (2006). Simple and fast inverse alignment. In *Research in Computational Molecular Biology*, volume 3909 of *Lecture Notes in Computer Science*, pages 441–455. Springer Berlin Heidelberg.

Kececioglu, J. D., Lenhof, H.-P., Mehlhorn, K., Mutzel, P., Reinert, K., and Vingron, M. (2000). A polyhedral approach to sequence alignment problems. *Discrete Applied Mathematics*, 104(1-3):143–186.

Keller, A., Purvine, S., Nesvizhskii, A. I., Stolyar, S., Goodlett, D. R., and Kolker, E. (2002). Experimental protein mixture for validating tandem mass spectral analysis. *OMICS: A Journal of Integrative Biology*, 6:207–212.

Kim, D. and Salzberg, S. L. (2011). TopHat-Fusion: an algorithm for discovery of novel fusion transcripts. *Genome Biology*, 12(8):R72.

Kim, H., Bi, Y., Pal, S., Gupta, R., and Davuluri, R. V. (2011). IsoformEx: isoform level gene expression estimation using weighted non-negative least squares from mRNA-Seq data. *BMC Bioinformatics*, 12(1):305.

Kingsford, C. L., Chazelle, B., and Singh, M. (2005). Solving and analyzing side-chain positioning problems using linear and integer programming. *Bioinformatics (Oxford, England)*, 21(7):1028–36.

Kodzius, R., Kojima, M., Nishiyori, H., Nakamura, M., Fukuda, S., Tagami, M., Sasaki, D., Imamura, K., Kai, C., Harbers, M., Hayashizaki, Y., and Carninci, P. (2006). CAGE: cap analysis of gene expression. *Nature Methods*, 3(3):211–22.

Kolman, P. and Pangrác, O. (2009). On the complexity of paths avoiding forbidden pairs. *Discrete Applied Mathematics*, 157(13):2871 – 2876.

Koskiniemi, S., Sun, S., Berg, O. G., and Andersson, D. I. (2012). Selection-driven gene loss in bacteria. *PLoS Genetics*, 8(6).

Laajala, E., Aittokallio, T., Lahesmaa, R., and Elo, L. L. (2009). Probe-level estimation improves the detection of differential splicing in Affymetrix exon array studies. *Genome Biology*, 10(7):R77.

Lancia, G. (2008). Mathematical programming in computational biology: an annotated bibliography. *Algorithms*, 1(2):100–129.

Lancia, G., Carr, R., Walenz, B., and Istrail, S. (2001). 101 optimal pdb structure alignments: A branch-and-cut algorithm for the maximum contact map overlap problem. In *Proceedings of the Fifth Annual International Conference on Computational Biology*, RECOMB, pages 193–202, New York, NY, USA. ACM.

Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25.

Lee, C. and Roy, M. (2004). Analysis of alternative splicing with microarrays: successes and challenges. *Genome Biology*, 5(7):231.

Lehninger, A., Nelson, D. L., and Cox, M. M. (2008). *Lehninger Principles of Biochemistry*. W. H. Freeman, 5th edition.

Li, B. and Dewey, C. N. (2011). RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics*, 12(1):323.

Li, H. and Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics (Oxford, England)*, 25(14):1754–60.

Li, J. J., Jiang, C.-R., Brown, J. B., Huang, H., and Bickel, P. J. (2011a). Sparse linear modeling of next-generation mrna sequencing (RNA-Seq) data for isoform discovery and abundance estimation. *Proceedings of the National Academy of Sciences*, 108(50):19867–19872.

Li, W., Feng, J., and Jiang, T. (2011b). IsoLasso: a LASSO regression approach to RNA-Seq based transcriptome assembly. *Journal of Computational Biology*, 18(11):1693–707.

Liu, C., Song, Y., Yan, B., Xu, Y., and Cai, L. (2006). Fast de novo peptide sequencing and spectral alignment via tree decomposition. In *Proceedings of the 11th Pacific Symposium on Biocomputing*, pages 255–266. World Scientific.

Lu, B. and Chen, T. (2003). A suboptimal algorithm for de novo peptide sequencing via tandem mass spectrometry. *Journal of Computational Biology*, 10(1):1–12.

Lu, W., Tamura, T., Song, J., and Akutsu, T. (2014). Integer programming-based method for designing synthetic metabolic networks by minimum reaction insertion in a boolean model. *PloS ONE*, 9(3).

Lynch, M. and Conery, J. S. (2000). The evolutionary fate and consequences of duplicate genes. *Science*, 290(5494):1151–1155.

Ma, B., Zhang, K., Hendrie, C., Liang, C., Li, M., Doherty-Kirby, A., and Lajoie, G. (2003). Peaks: Powerful software for peptide de novo sequencing by MS/MS. *Rapid Communications in Mass Spectrometry*, 17:2337–2342.

McPherson, A., Hormozdiari, F., Zayed, A., Giuliany, R., Ha, G., Sun, M. G. F., Griffith, M., Heravi Moussavi, A., Senz, J., Melnyk, N., Pacheco, M., Marra, M. A., Hirst, M., Nielsen, T. O., Sahinalp, S. C., Huntsman, D., and Shah, S. P. (2011). deFuse: an algorithm for gene fusion discovery in tumor RNA-Seq data. *PLoS Computational Biology*, 7(5).

Meneses, C. N., Lu, Z., Oliveira, C. A. S., and Pardalos, P. M. (2004). Optimal solutions for the closest-string problem via integer programming. *INFORMS Journal on Computing*, 16(4):419–429.

Mezlini, A. M., Smith, E. J., Fiume, M., Buske, O., Savich, G. L., Shah, S., Aparicio, S., Chiang, D. Y., Goldenberg, A., and Brudno, M. (2013). iReckon: simultaneous isoform discovery and abundance estimation from RNA-Seq data. *Genome research*, 23(3):519–529.

Michalski, A., Neuhauser, N., Cox, J., and Mann, M. (2012). A systematic investigation into the nature of tryptic HCD spectra. *Journal of Proteome Research*, 11(11):5479–91.

Mironov, A. A. (1999). Frequent alternative splicing of human genes. *Genome Research*, 9(12):1288–1293.

Moller-Levet, C. S., Betts, G. N. J., Harris, A. L., Homer, J. J., West, C. M. L., and Miller, C. J. (2009). Exon array analysis of head and neck cancers identifies a hypoxia related splice variant of LAMA3 associated with a poor prognosis. *PLoS Computational Biology*, 5(11).

Mortazavi, A., Williams, B. A., McCue, K., Schaeffer, L., and Wold, B. (2008). Mapping and quantifying mammalian transcriptomes by RNA-Seq. *Nature Methods*, 5(7):621–628.

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.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA.

Ner-Gaon, H. and Fluhr, R. (2006). Whole-genome microarray in Arabidopsis facilitates global analysis of retained introns. *DNA Research*, 13(3):111–21.

Nicolae, M., Mangul, S., Măndoiu, I. I., and Zelikovsky, A. (2011). Estimation of alternative splicing isoform frequencies from RNA-Seq data. *Algorithms for Molecular Biology: AMB*, 6(1):9.

Ohno, S. (1970). *Evolution by gene duplication*. Springer, Berlin, New York.

Okoniewski, M. J. and Miller, C. J. (2006). Hybridization interactions between probesets in short oligo microarrays lead to spurious correlations. *BMC Bioinformatics*, 7:276.

Olsen, J. V., Macek, B., Lange, O., Makarov, A., Horning, S., and Mann, M. (2007). Higher-energy C-trap dissociation for peptide modification analysis. *Nature Methods*, 4(9):709–12.

Olson, M. V. (1999). When less is more: gene loss as an engine of evolutionary change. *American Journal of Human Genetics*, 64(1):18–23.

Pan, Q., Shai, O., Lee, L. J., Frey, B. J., and Blencowe, B. J. (2008). Deep surveying of alternative splicing complexity in the human transcriptome by high-throughput sequencing. *Nature Genetics*, 40:1413–1415.

Pan, Q., Shai, O., Misquitta, C., Zhang, W., Saltzman, A. L., Mohammad, N., Babak, T., Siu, H., Hughes, T. R., Morris, Q. D., Frey, B. J., and Blencowe, B. J. (2004). Revealing global regulatory features of mammalian alternative splicing using a quantitative microarray platform. *Molecular Cell*, 16(6):929–41.

Perkins, D. N., Pappin, D. J., Creasy, D. M., and Cottrell, J. S. (1999). Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20(18):3551–3567.

Reinert, K. (1999). *A polyhedral approach to sequence alignment problems*. PhD thesis, Saarländische Universitäts- und Landesbibliothek, Postfach 151141, 66041 Saarbrücken.

Reinert, K., Lenhof, H.-P., Mutzel, P., Mehlhorn, K., and Kececioglu, J. D. (1997). A branch-and-cut algorithm for multiple sequence alignment. In *Proceedings of the First Annual International Conference on Computational Molecular Biology*, RECOMB, pages 241–250, New York, NY, USA. ACM.

Ritz, A., Bashir, A., and Raphael, B. J. (2010). Structural variation analysis with strobe reads. *Bioinformatics (Oxford, England)*, 26(10):1291–8.

Robinson, M. D., McCarthy, D. J., and Smyth, G. K. (2010). edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics (Oxford, England)*, 26(1):139–40.

Roepstorff, P. and Fohlman, J. (1984). Proposal for a common nomenclature for sequence ions in mass spectra of peptides. *Biomedical Mass Spectrometry*, 11(11):601.

Rogers, H. H., Bergman, C. M., and Griffiths-Jones, S. (2010). The evolution of tRNA genes in Drosophila. *Genome Biology and Evolution*, 2:467–77.

Royce, T. E., Rozowsky, J. S., and Gerstein, M. B. (2007). Toward a universal microarray: prediction of gene expression through nearest-neighbor probe sequence identification. *Nucleic Acids Research*, 35(15):e99.

Saliba, A.-E., Westermann, A. J., Gorski, S. A., and Vogel, J. (2014). Single-cell RNA-Seq: advances and future challenges. *Nucleic Acids Research*, 42(14):8845–8860.

Sankoff, D. and Blanchette, M. (1997). The median problem for breakpoints in comparative genomics. In *COCOON*, Lecture Notes in Computer Science, pages 251–264.

Schulz, M. H., Zerbino, D. R., Vingron, M., and Birney, E. (2012). Oases: robust de novo RNA-Seq assembly across the dynamic range of expression levels. *Bioinformatics (Oxford, England)*, 28(8):1086–92.

Shen, S., Warzecha, C. C., Carstens, R. P., and Xing, Y. (2010). MADS+: discovery of differential splicing events from Affymetrix exon junction array data. *Bioinformatics (Oxford, England)*, 26(2):268–9.

Siragusa, E., Weese, D., and Reinert, K. (2013). Fast and accurate read mapping with approximate seeds and multiple backtracking. *Nucleic Acids Research*, 41(7):e78.

Song, L. and Florea, L. (2013). CLASS: constrained transcript assembly of RNA-Seq reads. *BMC Bioinformatics*, 14 (suppl 5):S14.

Srinivasan, G., James, C. M., and Krzycki, J. A. (2002). Pyrrolysine encoded by UAG in Archaea: charging of a UAG-decoding specialized tRNA. *Science*, 296(5572):1459–62.

Steen, H. and Mann, M. (2004). The ABC's (and XYZ's) of peptide sequencing. *Nature Reviews. Molecular Cell Biology*, 5(9):699–711.

Stolc, V., Samanta, M. P., Tongprasit, W., Sethi, H., Liang, S., Nelson, D. C., Hegeman, A., Nelson, C., Rancour, D., Bednarek, S., Ulrich, E. L., Zhao, Q., Wrobel, R. L., Newman, C. S., Fox, B. G., Phillips, G. N., Markley, J. L., and Sussman, M. R. (2005). Identification of transcribed sequences in *Arabidopsis thaliana* by using high-resolution genome tiling arrays. *Proceedings of the National Academy of Sciences of the United States of America*, 102(12):4453–8.

Sturm, M., Bertsch, A., Gröpl, C., Hildebrandt, A., Hussong, R., Lange, E., Pfeifer, N., Schulz-Trieglaff, O., Zerck, A., Reinert, K., and Kohlbacher, O. (2008). OpenMS - an open-source software framework for mass spectrometry. *BMC Bioinformatics*, 9:163.

Syka, J. E. P., Coon, J. J., Schroeder, M. J., Shabanowitz, J., and Hunt, D. F. (2004). Peptide and protein sequence analysis by electron transfer dissociation mass spectrometry. *Proceedings of the National Academy of Sciences of the United States of America*, 101(26):9528–33.

Tanner, S., Shu, H., Frank, A., Wang, L. C., Zandi, E., Mumby, M., Pevzner, P. A., and Bafna, V. (2005). Inspect: identification of post translationally modified peptides from tandem mass spectra. *Analytical Chemistry*, 77(14):4626–4639.

Taylor, J. A. and Johnson, R. S. (1997). Sequence database searches via de novo peptide sequencing by tandem mass spectrometry. *Rapid Communications in Mass Spectrometry*, 11(9):1067–1075.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B: Methodological*, pages 267–288.

Tomescu, A. I., Kuosmanen, A., Rizzi, R., and Mäkinen, V. (2013). A novel min-cost flow method for estimating transcript expression with RNA-Seq. *BMC Bioinformatics*, 14 (suppl 5):S15.

Trapnell, C., Hendrickson, D. G., Sauvageau, M., Goff, L., Rinn, J. L., and Pachter, L. (2013). Differential analysis of gene regulation at transcript resolution with RNA-Seq. *Nature Biotechnology*, 31(1):46–53.

Trapnell, C., Pachter, L., and Salzberg, S. L. (2009). TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics (Oxford, England)*, 25(9):1105–11.

Trapnell, C., Williams, B. A., Pertea, G., Mortazavi, A., Kwan, G., van Baren, M. J., Salzberg, S. L., Wold, B. J., and Pachter, L. (2010). Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, 28(5):516–520.

Tyers, M. and Mann, M. (2003). From genomics to proteomics. *Nature*, 422(6928):193–197.

Velculescu, V. E., Zhang, L., Vogelstein, B., and Kinzler, K. W. (1995). Serial analysis of gene expression. *Science*, 270(5235):484–487.

Velculescu, V. E., Zhang, L., Zhou, W., Vogelstein, J., Basrai, M. A., Bassett, D. E., Hieter, P., Vogelstein, B., and Kinzler, K. W. (1997). Characterization of the yeast transcriptome. *Cell*, 88(2):243–251.

Wang, K., Singh, D., Zeng, Z., Coleman, S. J., Huang, Y., Savich, G. L., He, X., Mieczkowski, P., Grimm, S. A., Perou, C. M., MacLeod, J. N., Chiang, D. Y., Prins, J. F., and Liu, J. (2010). MapSplice: accurate mapping of RNA-Seq reads for splice junction discovery. *Nucleic Acids Research*, 38(18):e178.

Wang, X., Grus, W. E., and Zhang, J. (2006). Gene losses during human origins. *PLoS Biology*, 4(3).

Wang, Z., Gerstein, M., and Snyder, M. (2009). RNA-Seq: a revolutionary tool for transcriptomics. *Nature Reviews. Genetics*, 10(1):57–63.

Weese, D., Holtgrewe, M., and Reinert, K. (2012). RazerS 3: faster, fully sensitive read mapping. *Bioinformatics (Oxford, England)*, 28(20):2592–9.

Wells, J. M. and McLuckey, S. A. (2005). Collision-induced dissociation (CID) of peptides and proteins. *Methods in Enzymology*, 402:148–85.

Wilkins, M. R., Pasquali, C., Appel, R. D., Ou, K., Golaz, O., Sanchez, J. C., Yan, J. X., Gooley, A. A., Hughes, G., Humphery-Smith, I., Williams, K. L., and Hochstrasser, D. F. (1996). From proteins to proteomes: large scale protein identification by two-dimensional electrophoresis and amino acid analysis. *Biotechnology*, 14(1):61–65.

Withers, M., Wernisch, L., and dos Reis, M. (2006). Archaeology and evolution of transfer RNA genes in the *Escherichia coli* genome. *RNA*, 12(6):933–942.

Wohlers, I., Domingues, F. S., and Klau, G. W. (2010). Towards optimal alignment of protein structure distance matrices. *Bioinformatics (Oxford, England)*, 26(18):2273–80.

Wolsey, L. A. (1998). *Integer programming*. Wiley-Interscience, New York, NY, USA.

Xu, J., Li, M., Kim, D., and Xu, Y. (2003). RAPTOR: optimal protein threading by linear programming. *Journal of Bioinformatics and Computational Biology*, 01(01):95–117.

Xu, J., Li, M., and Xu, Y. (2004). Protein threading by linear programming: theoretical analysis and computational results. *Journal of Combinatorial Optimization*, 8(4):403–418.

Yancopoulos, S., Attie, O., and Friedberg, R. (2005). Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics (Oxford, England)*, 21(16):3340–3346.

Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716.

Zerbino, D. R. and Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829.

Zerck, A., Nordhoff, E., Lehrach, H., and Reinert, K. (2013). Optimal precursor ion selection for LC-MALDI MS/MS. *BMC Bioinformatics*, 14:56.

Zhao, S., Fung-Leung, W.-P., Bittner, A., Ngo, K., and Liu, X. (2014). Comparison of RNA-Seq and microarray in transcriptome profiling of activated T cells. *PloS ONE*, 9(1).

Zubarev, R. A., Kelleher, N. L., and McLafferty, F. W. (1998). Electron capture dissociation of multiply charged protein cations. A nonergodic process. *Journal of the American Chemical Society*, 120(13):3265–3266.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF NOTATIONS

$\langle a_1, a_2, \ldots, a_n \rangle$	sequence of symbols $a_1, \ldots, a_n$ from a given alphabet Σ
$\lvert X \rvert, \lvert s \rvert$	cardinality of a set X or length of a sequence s
$\preceq, \succeq$	substring relation for a pair of sequences
$\alpha \cdot \beta$	concatenation of sequences α and β
$[a \mathinner{..} b]$	set of integers $a, a+1, \ldots, b-1, b$
$\mathbf{c}^\top, \mathbf{A}^\top$	transpose of a vector $\mathbf{c}$ or a matrix $\mathbf{A}$
$(a_1, a_2, \ldots, a_n)$	n-dimensional column vector
$[\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n]$	$m \times n$ matrix with m-dimensional column vectors $\mathbf{A}_1, \ldots, \mathbf{A}_n$
$\mathcal{P}(S)$	power set of set S
$\mathcal{O}$	Landau symbol "Big O"

INDEX