École doctorale IAEM Lorraine

# Lagrangian Relaxation Solving $\mathcal{NP}$-hard Problems in Computational Biology via Combinatorial Optimization

Stefan Canzar

**Deutsch-Französische Doppelpromotion**
**THÈSE EN CO-TUTELLE**

zur Erlangung des Grades
pour l'obtention du

**des Doktors der Ingenieurswissenschaften (Dr.-Ing.)**
**der Naturwissenschaftlich-Technischen Fakultäten**
**der Universität des Saarlandes**

**Doctorat de l'Université Henri Poincaré – Nancy 1**
**Département de formation doctorale en informatique**
**UFR STMIA (spécialité informatique)**

**Max-Planck-Institut für Informatik**
**Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503**

**Date de Soutenance :** 14 Novembre 2008

**Composition du Jury**

| | |
|---|---|
| *Président :* | Prof. René Schott |
| *Rapporteurs :* | Prof. Dr. Knut Reinert |
| | Dr. Gregory Kucherov |
| | Prof. Dr. Ernst Althaus |
| *Directeurs :* | Prof. Dr. Ernst Althaus |
| | Dr. habil. Jens Gustedt |
| | Prof. Dr. Kurt Mehlhorn |

# Abstract

This thesis is devoted to two $\mathcal{NP}$-complete combinatorial optimization problems arising in computational biology, the well-studied *multiple sequence alignment* problem and the new formulated *interval constraint coloring* problem. It shows that advanced mathematical programming techniques are capable of solving large scale real-world instances from biology to optimality. Furthermore, it reveals alternative methods that provide approximate solutions.

In the first part of the thesis, we present a *Lagrangian relaxation* approach for the multiple sequence alignment (MSA) problem. The multiple alignment is one common mathematical abstraction of the comparison of multiple biological sequences, like DNA, RNA, or protein sequences. If the weight of a multiple alignment is measured by the sum of the projected pairwise weights of all pairs of sequences in the alignment, then finding a multiple alignment of maximum weight is $\mathcal{NP}$-complete if the number of sequences is not fixed. The majority of the available tools for aligning multiple sequences implement heuristic algorithms ; no current exact method is able to solve moderately large instances or instances involving sequences exhibiting a lower degree of similarity.

We present a branch-and-bound (B&B) algorithm for the MSA problem. We approximate the optimal integer solution in the nodes of the B&B tree by a Lagrangian relaxation of an ILP formulation for MSA relative to an exponential large class of inequalities, that ensure that all pairwise alignments can be incorporated to a multiple alignment. By lifting these constraints prior to dualization the Lagrangian subproblem becomes an *extended pairwise alignment* (EPA) problem : Compute the longest path in an acyclic graph, that is penalized a charge for entering "obstacles". We describe an efficient algorithm that solves the EPA problem repetitively to determine near-optimal *Lagrangian multipliers* via subgradient optimization. The reformulation of the dualized constraints with respect to additionally introduced variables improves the convergence rate dramatically. We account for the exponential number of dualized constraints by starting with an empty *constraint pool* in the first iteration to which we add cuts in each iteration, that are most violated by the convex combination of a small number of preceding Lagrangian solutions (including the current solution). In this *relax-and-cut* scheme, only inequalities from the constraint pool are dualized.

The interval constraint coloring problem appears in the interpretation of experimental data in biochemistry. Monitoring hydrogen-deuterium exchange rates via mass spectroscopy is a method used to obtain information about protein tertiary structure. The output of these experiments provides aggregate data about the exchange rate of residues in overlapping fragments of the protein backbone. These fragments must be re-assembled in order to obtain a global picture of the protein structure. The interval constraint coloring problem is the mathematical abstraction of this re-assembly process.

The objective of the interval constraint coloring problem is to assign a color (exchange rate) to a set of integers (protein residues) such that a set of constraints is satisfied. Each constraint is made up of a closed interval (protein fragment) and requirements on the number of elements in the interval that belong to each color class (exchange rates observed in the experiments).

We introduce a polyhedral description of the interval constraint coloring problem, which serves as a basis to attack the problem by integer linear programming (ILP) methods and tools, which perform well in practice. Since the goal is to provide biochemists with all possible candidate solutions, we combine related solutions to equivalence classes in an improved ILP formulation in order to reduce the running time of our enumeration algorithm. Moreover, we establish the polynomial-time solvability of the two-color case by the integrality of the linear programming relaxation polytope $\mathcal{P}$, and also present a combinatorial polynomial-time algorithm for this case. We apply this algorithm as a subroutine to approximate solutions to instances with arbitrary but fixed number of colors and achieve an order of magnitude improvement in running time over the (exact) ILP approach.

We show that the problem is $\mathcal{NP}$-complete for arbitrary number of colors, and we provide algorithms that, given an instance with $\mathcal{P} \neq \emptyset$, find a coloring that satisfies all the coloring requirements within $\pm 1$ of the prescribed value. In light of our $\mathcal{NP}$-completeness result, this is essentially the best one can hope for. Our approach is based on polyhedral theory and randomized rounding techniques.

In practice, data emanating from the experiments are noisy, which normally causes the instance to be infeasible, and, in some cases, even forces $\mathcal{P}$ to be empty. To deal with this problem, the objective of the ILP is to minimize the total sum of absolute deviations from the coloring requirements over all intervals. The combinatorial approach for the two-color case optimizes the same objective function. Furthermore, we use this combinatorial method to compute, in a Lagrangian way, a bound on the minimum total error, which is exploited in a branch-and-bound manner to determine all optimal colorings. Alternatively, we study a variant of the problem in which we want to maximize the number of requirements that are satisfied. We prove that this variant is $\mathcal{APX}$-hard even in the two-color case and thus does not admit a polynomial time approximation scheme (PTAS) unless $\mathcal{P} = \mathcal{NP}$. Therefore, we slightly (by a factor of $(1+\epsilon)$) relax the condition on when a requirement is satisfied and propose a *quasi-polynomial time approximation scheme* (QPTAS) which finds a coloring that "satisfies" the requirements of as many intervals as possible.

# Zusammenfassung

Die vorliegende Dissertation widmet sich zwei $\mathcal{NP}$-vollständigen kombinatorischen Optimierungsproblemen aus der Bioinformatik, dem intensiv erforschten Problem des *Multiplen Sequenzalignments* (englisch: *multiple sequence alignment*) sowie dem neu formulierten *Intervallinduzierten Färbungsproblem* (englisch: *interval constraint coloring*). Sie zeigt, dass mithilfe von fortgeschrittenen Methoden der mathematischen Programmierung durchaus auch höherdimensionale Probleminstanzen der Biologie exakt gelöst werden können. Darüberhinaus beschreibt sie alternative Ansätze, die es erlauben, approximative Lösungen (mit und ohne Garantie bzgl. Approximationsgüte) zu bestimmen.

Im ersten Teil dieser Dissertation stellen wir einen Algorithmus für das Multiple Sequenzalignment (MSA) vor, der dem Konzept einer *Lagrange-Relaxierung* folgt. Das Multiple Alignment ist eine gebräuchliche mathematische Abstraktion des Vergleichs von mehreren biologischen Sequenzen, wie etwa DNA, RNA oder Proteinsequenzen. Berechnet sich das Gewicht eines Multiplen Alignments als die Summe der Gewichte aller projezierten Sequenzpaare in dem Alignment, so ist die Bestimmung eines Multiplen Alignments maximalen Gewichtes $\mathcal{NP}$-vollständig, falls die Anzahl der Sequenzen nicht als fest vorgegeben betrachtet wird. Die Mehrzahl der verfügbaren Programme verfolgt deshalb einen heuristischen Ansatz; keine bisher vorgestellte Methode ist in der Lage, moderat große Instanzen oder Instanzen, die Sequenzen von einem niedrigen Ähnlichkeitsgrad umfassen, exakt (optimal) zu lösen.

Wir stellen einen exakten Algorithmus für das Problem des Multiplen Sequenzalignments vor, der auf dem Branch-and-Bound (B&B) Prinzip beruht. Wir approximieren die optimale ganzzahlige Lösung in den Knoten des B&B Baums durch eine Lagrange-Relaxierung einer ILP Formulierung von MSA. Wir dualisieren eine Klasse von Ungleichungen exponentieller Größe, die sicherstellt, dass alle paarweisen Alignments konfliktfrei zu einem Multiplen Alignment zusammengesetzt werden können. Durch eine vorangehende Verstärkung dieser Ungleichungen wird das Lagrange-Subproblem zum *Erweiterten Paarweisen Alignment* (EPA) Problem: berechne den längsten Pfad in einem azyklischen Graph, der für das Betreten von "Hindernissen" bestraft wird. Wir beschreiben einen effizienten Algorithmus, der das EPA Problem wiederholt löst, um annähernd optimale *Lagrange-Multiplikatoren* mittels Subgradienten-Verfahren zu bestimmen. Die Umformulierung der dualisierten Ungleichungen bezüglich zusätzlich ein-

geführter Variablen verbessert die Konvergenzrate dramatisch. Wir begegnen dem Problem der exponentiellen Größe der Menge von dualisierten Ungleichungen, indem wir, beginnend mit einem leeren *Constraint-Pool* in der ersten Iteration, in jeder weiteren Iteration diesem Pool Ungleichungen hinzufügen, die von der Konvexkombination einer kleinen Zahl von vorhergenden Lagrange-Lösungen (einschließlich der aktuellen Lösung) am stärksten verletzt werden. Dem *Relax-and-Cut* Schema folgend werden jeweils nur im Constraint-Pool befindliche Ungleichungen dualisiert.

Das Intervallinduzierte Färbungsproblem taucht in der Biochemie bei der Interpretation von experimentellen Messdaten auf. Die Beobachtung von Wasserstoff-Deuterium-Austauschraten mittels Massenspektrometrie gibt Hinweis auf die tertiäre Struktur von Proteinen. Die experimentell ermittelten Daten geben Aufschluss über die Verteilung der Austauschraten innerhalb von sich überlappenden Fragmenten der Proteinsequenz. Ziel ist es, den einzelnen Residuen Austauschraten so zuzuordnen, dass sie mit den beobachteten Verteilungen konsistent sind. Nur solch hinreichend feingranulare Informationen lassen den Schluss auf die Lösungsmittel-Zugänglichkeit von Molekülabschnitten, und damit auf Eigenschaften der tertiären Struktur, zu. Das Intervallinduzierte Färbungsproblem ist die mathematische Abstraktion dieses Verfeinerungsprozesses.

Das Intervallinduzierte Färbungsproblem verlangt nun, einer Menge von positiven ganzen Zahlen (den Residuen), unter Einhaltung von gewissen Bedingungen, eine Farbe (Austauschrate) zuzuordnen. Dabei gilt eine Bedingung als erfüllt, wenn sich eine erforderlich Anzahl von Elementen jeder Farbklasse (den experimentell beobachteten Austauschraten) in einem gegebenen geschlossenen Interval (Proteinfragment) befinden.

Wir formulieren das Problem als ganzzahliges lineares Programm (ILP) und wenden einen impliziten Aufzählungsalgorithmus an, der typische Probleminstanzen aus der Praxis sehr effizient löst. Da man in der Biochemie an allen möglichen Kandidatenlösungen interessiert ist, fassen wir in einer überarbeiteten ILP Formulierung "ähnliche" Lösungen in Äquivalenzklassen zusammen und verbessern dadurch das Laufzeitverhalten unseres Aufzählungsalgorithmus. Gleichzeitig begründen wir die Lösbarkeit des Zweifarbenfalls in polynomieller Zeit durch die Ganzzahligkeit des durch die lineare Relaxierung beschriebenen Polytops $\mathcal{P}$ und beschreiben für eben diesen Fall einen kombinatorischen Algorithmus mit polynomieller Laufzeit. Dieser Algorithmus dient als Baustein, um Lösungen von Instanzen mit beliebiger aber fester Anzahl von Farben zu approximieren (ohne Garantie bzgl. Approximationsgüte). Gegenüber dem (exakten) ILP-Ansatz erzielen wir dadurch eine Laufzeitverbesserung die im Bereich einer Größenordnung liegt.

Wir beweisen, dass dieses Problem, gegeben eine beliebige Anzahl von Farben, $\mathcal{NP}$-vollständig ist und beschreiben einen Algorithmus zur Bestimmung einer Färbung, vorausgesetzt $\mathcal{P} \neq \emptyset$, die von allen Farbanforderungen jeweils um höchstens $\pm 1$ abweicht. Im Angesicht der $\mathcal{NP}$-Vollständigkeit dieses Problems ist mit einem sehr viel stärkeren Resultat nicht zu rechnen. Unser Ansatz beruht auf Erkenntnissen der Polyedertheorie und nutzt randomisierte Rundungstechniken.

In der Praxis sind experimentell ermittelte Daten jedoch fehlerbehaftet, was meist zur Unlösbarkeit des Problems führt und gelegentlich sogar $\mathcal{P} = \emptyset$ bewirkt. Wir begegnen diesem Problem zum einen mit der Modellierung der absoluten Abweichung von den

Farbanforderungen mithilfe zusätzlicher Variablen in dem ganzzahligen linearen Programm, deren Summe über alle Intervalle in der Zielfunktion minimiert wird. Der kombinatorische Ansatz für den Zweifarbenfall optimiert dabei den gleichen Zielfunktionswert. Darüberhinaus erlaubt uns eine geeignete Lagrange-Relaxierung des Problems, mithilfe dieses kombinatorischen Algorithmus eine Schranke für den minimal möglichen Fehler zu berechnen, der in einem B&B Schema verwendet wird, um alle optimalen Färbungen zu bestimmen. Als Alternative dazu formulieren wir eine Problemvariante, in der wir die Anzahl der Intervalle, deren Farbanforderungen erfüllt werden, zu maximieren versuchen. Wir zeigen, dass diese Problemvariante bereits im Zweifarbenfall $\mathcal{APX}$-hart ist und damit kein polynomielles Approximationsschema (PTAS) zulässt, es sei denn $\mathcal{P} = \mathcal{NP}$. Deshalb relaxieren wir die Erfüllung einer Farbanforderung geringfügig (um einen multiplikativen Faktor $(1+\epsilon)$) und führen ein Approximationsschema mit quasi-polynomieller Laufzeit ein, das die Anforderungen von so vielen Intervallen wie möglich "erfüllt".

# Résumé

Cette thèse est dédiée à la résolution de deux problèmes d'optimisation combinatoire $\mathcal{NP}$-complets surgissant en bioinformatique, à savoir le problème classique d'alignement de séquences, ainsi qu'un problème nouvellement formalisé, le problème de *coloration par contraintes d'intervalles* ou *Interval Constraint Coloring Problem* (*ICP*). Nous montrons dans cette thèse qu'il est possible de résoudre des instances réelles et de grande taille de ces problèmes apparaissant en biologie, et ceci par des méthodes de programmation mathématiques avancées. Nous démontrons également l'existence de méthodes plus efficaces, permettant d'obtenir des solutions approchées pour ces mêmes problèmes.

Dans la première partie de la thèse, nous présentons un algorithme pour la solution du problème classique de l'*alignement de séquences*, basé sur la *relaxation lagrangienne*. Le problème de l'alignement de séquences est une abstraction mathématique courante du problème de comparaison de séquences biologiques, comme l'ADN, l'ARN ou les séquences de protéines. Si le poids d'un alignement séquentiel multiple est calculé comme la somme des poids des paires de séquence projetées de l'alignement considéré, alors le problème de déterminer un alignement de poids maximal est $\mathcal{NP}$-complet, tant que le nombre de séquences n'est pas fixé. La plupart des logiciels disponibles pour la résolution du problème d'alignement de séquences se focalisent donc sur des approches heuristiques. Aucune méthode n'est actuellement capable de résoudre efficacement des instances de taille moyenne, ou des instances comportant des séquences d'un faible taux de similitude, de ce problème de manière exacte.

Nous présentons un nouvel algorithme pour la résolution du problème de l'alignement de séquences, basé sur la technique de *séparation et évaluation* (*branch-and-bound* ou B&B). Nous approchons la solution optimale en nombres entiers dans les nœuds de l'arbre B&B par une relaxation lagrangienne de la formulation en tant que PLNE du problème d'alignement de séquences multiples. Un nombre exponentiel d'inégalités supplémentaires doit alors être vérifié afin de garantir que l'alignement de séquences multiples peut être reconstruit sans conflits à partir des alignements individuels. En renforcant ces inégalités avant de les dualiser, le sous-problème lagrangien devient le *problème d'alignement par paires étendu* : il s'agit alors de trouver le plus long chemin dans un graphe acyclique, auquel sont ajoutés des pénalités lors du passage à travers certaines zones "obstacles". Nous introduisons un algorithme efficace permettant de résoudre ce

problème de manière répétitive, afin de trouver une bonne approximation des *multiplicateurs de Lagrange* via optimization du sous-gradient. La reformulation des inégalités dualisées par rapport à des variables supplémentaires améliore de manière significative le taux de convergence de l'algorithme. Nous adressons le problème du nombre exponentiel d'inégalités par une approche itérative. L'ensemble des contraintes est vide au début. Après chaque itération, nous rajoutons à cet ensemble ces inégalités qui sont le plus violées par la combinaison convexe d'un petit nombre des solutions lagrangiennes précédentes (y compris la solution courante). Conformément au schéma *relax-and-cut*, nous dualisons exclusivement les inégalités présents dans l'ensemble des contraintes décrit précédemment.

L'ICP est un problème qui se pose lors de l'analyse et l'interprétation de données expérimentales en biochimie. L'analyse du taux d'échange hydrogène/deutérium par spectrométrie de masse est une des méthodes utilisées pour obtenir des informations sur la structure tertiaire des protéines. Les résultats de ces expériences se présentent sous forme de taux d'échange des résidus dans les segments chevauchés de la protéine. Ces segments doivent être recollés afin d'obtenir une vision globale de la structure de la protéine. L'ICP est l'abstraction mathématique de ce process de recombinaison.

L'objectif de l'ICP est d'attribuer une couleur (taux d'échange) à un ensemble d'entiers (résidus de protéines) de telle manière à ce qu'un ensemble de contraintes est vérifié. Chaque contrainte représente un intervalle fermé (segment d'une protéine) ainsi qu'un ensemble d'exigences supplémentaires concernant le nombre d'éléments qui doivent appartenir à chacune des catégories de couleur (taux d'échanges observés lors des expériences).

Nous montrons que le problème peut être résolu par des méthodes de programmation linéraire en nombres entiers (PLNE), et nous utilisons un algorithme d'énumération implicite qui s'avère efficace pour la plupart des problèmes qu'on rencontre dans la pratique. Puisque notre motivation est de fournir aux biochimistes une liste exhaustive des solutions potentielles, une version améliorée de notre approche PLNE consiste à regrouper des solutions similaires dans des classes d'équivalence, ceci afin d'établir une version améliorée et plus performante de la procédure d'énumération. Nous démontrons ensuite la solvabilité du cas particulier à deux couleurs par la contrainte de solution en nombres entiers du polytope $\mathcal{P}$, défini à travers la relaxation linéaire, tout en proposant un algorithme de résolution de complexité polynomielle pour ce cas précis. Cet algorithme sert ensuite de base pour l'établissement de solutions approchés d'instances de dimension quelconque mais fixe (pour le moment sans garantie sur la qualité de la solution obtenue). Nous obtenons ainsi une amélioration d'un ordre de grandeur en termes de performance par rapport à la solution exacte, basée sur l'approche PLNE.

Nous démontrons que ce problème est $\mathcal{NP}$-complet pour un nombre arbitraire de couleurs. Nous établissons ensuite un algorithme qui, étant donné $\mathcal{P} \neq \emptyset$, est capable de déterminer une coloration satisfaisante toutes les contraintes données avec un écart maximal de $\pm 1$ des valeurs cible. Vue la complexité en $\mathcal{NP}$ du problème, il ne semble pas possible d'obtenir des solutions d'une qualité sensiblement supérieure. Notre approche est essentiellement basée sur la théorie des polyèdres et des techniques d'arrondissage randomisés.

Les données obtenues lors des expériences biologiques réelles sont souvent bruitées, ce qui entraine le plus souvent l'insolvabilité du problème, voire même $\mathcal{P} = \emptyset$. Afin d'adresser ce problème, l'objectif de la PLNE est de minimiser la somme des déviations absolues des contraintes de coloration sur l'intégralité des intervalles. L'approche particulière à deux couleurs optimise en effet cette même fonction. En outre, nous utilisons cette approche combinatoire pour déterminer, d'une façon lagrangienne, une borne sur l'erreur minimale, qui sera ensuite utilisée dans un algorithme de type branch-and-bound pour déterminer toutes les colorations optimales. Nous proposons une variante du problème précédent, dans laquelle nous essayons de maximiser le nombre contraintes qui peuvent être satisfaits en même temps. Nous démontrons que ce problème est $\mathcal{APX}$-dur et qu'il ne permet donc pas de schéma d'approximation polynomial sauf si $\mathcal{P} = \mathcal{NP}$. C'est pourquoi nous relaxons légèrement le critère de satisfiabilité (par un facteur $(1 + \epsilon)$) et décrivons par la suite un schéma d'approximation d'une complexité quasi-polynomiale, permettant de "satisfaire" le plus grand nombre de contraintes possible.

# Acknowledgements

I could not have written this thesis without the help of many people. Most notably, I thank my supervisor Ernst Althaus for his guidance and helpful advice throughout my doctoral study. I am also much obliged to Kurt Mehlhorn for giving me the opportunity to join the Algorithms and Complexity group at the Max-Planck-Institut für Informatik. I also want to thank Jens Gustedt (LORIA) for his support and his instant commitment to become a second supervisor of my bi-nationally supervised doctoral thesis.

The research I did with my collaborative colleagues Ernst Althaus, Carsten Ehrler, Khaled Elbassioni, Andreas Karrenbauer, Julián Mestre, Rajiv Raman (all at MPI Saarbrücken), Naveen Garg (Indian Institute of Technology), and Jan Remy (ETH Zürich) was a real pleasure. I owe them hours of inspiring and fruitful discussions. Additionally, I would like to thank our collaborative partners Mark R. Emmett, Alan G. Marshall, Huimin Zhang (all at National High Magnetic Field Laboratory, FSU), and Anke Meyer-Baese (FAMU-FSU College of Engineering) for the dynamic interaction, which enabled us to develop a precise mathematical model of the underlying biological problem, and for providing us with valuable data collected in elaborate experiments.

I am grateful and proud that with Gregory Kucherov (LIFL) and Knut Reinert (FU Berlin) two leading experts in this field agreed to co-examine my thesis.

I would also like to express my gratitude to my friends and colleagues who contributed to this thesis in many ways: Deepak Ajwani (MADALGO), Kanela Kaligosi (MPI Saarbrücken), Sören Laue (FSU Jena), Domagoj Matijevic (University of Osijek), Antonina Mitrofanova (New York University), Daniel Szer.

The work on this thesis was financially supported through CNRS (Centre National de la Recherche Scientifique), Région Lorraine, the German Academic Exchange Service (DAAD), and through a Ph.D. position at the Max-Planck-Institut für Informatik. I consider it an honor and privilege to have had the possibility to work at such stimulating places like the MPI in Saarbrücken and the LORIA in Nancy.

Finally, I am deeply grateful to my parents for their invaluable support they gave me during my entire studies.

# Table of Contents

# List of Figures

# List of Tables

Chapter 1

# Introduction

> "What's new?" is an interesting and broadening eternal question, but one which, if pursued exclusively, results only in an endless parade of trivia and fashion, the silt of tomorrow. I would like, instead, to be concerned with the question "What is best?", a question which cuts deeply rather than broadly, a question whose answers tend to move the silt downstream.
>
> Robert M. Pirsig
> *Zen and the Art of Motorcycle Maintenance* (1974)

Challenging mathematical problems that arose in the world in which *Diophantus of Alexandria* (A.D. c.200 - c.284), often known as "the father of algebra", lived, found their way into *Arithmetica*. In his major work, he collected 130 problems giving numerical solutions of determinate and indeterminate algebraic equations, potentially in the domain of integer numbers. Most of the problems the work considers lead to quadratic equations. Diophantus was always satisfied with *any* positive rational solution; he considered negative or irrational solutions "useless", or even "absurd" (how could a problem lead to $-4$ books?).

In the work *Liber abbaci* (1202), *Leonardo of Pisa* (A.D. c.1170 - c.1240), known as *Fibonacci* (from "filio Bonacci"), provided problems and puzzles inspired by everyday life that often lead to linear equations, for example: "A man buys 30 birds: partridges, doves and sparrows. A partridge costs three silver coins, a dove two and a sparrow 1/2. He pays with 30 silver coins. How many birds of each type did he buy?" Assuming positive integers, the only possible solution is 3, 5, and 22, respectively. For centuries, especially in contexts like mechanics, astronomy, or economics, linear algebraic equations and diophantine equations were the tool by which problems could be solved. Frequently these problems had unique solutions, in other words, no "choice" was involved.

In the more recent past, however, problems arising in mathematics, the biological and chemical sciences, engineering, and management, allow for a large set of alternative solutions, non of which is "useless" or even "absurd". Nevertheless, they may be of different

*value*, which naturally leads to the question "What is best?". Informally speaking, we call such problems, in which we seek to optimize the quality of a solution, an *optimization problem*. Especially *computational biology* is a great source of important and difficult optimization problems, that frequently involve huge but *discrete* solution spaces, so-called *combinatorial optimization problems*.

This thesis is devoted to two $\mathcal{NP}$-complete combinatorial optimization problems (and variants of them) arising in computational biology, the *multiple sequence alignment* problem, and in biochemistry, the *interval constraint coloring* problem. In both problems we obtain bounds on the best value of a solution by exploiting, in a *Lagrangian relaxation* scheme, the special structure embedded in the problem.

## 1.1 Lagrangian Relaxation A Combinatorial Alternative to Linear Programming

The first step in the canonical approach to $\mathcal{NP}$-hard combinatorial optimization problems is to find a "good" integer linear programming (ILP) formulation. From a "good" formulation one can derive a problem that can be solved effectively, yet yielding a strong bound on the objective value of the original problem. Since it is the restriction of (a subset of) variables to take integral values that destroy the convexity of the feasible region and thus make the problem seemingly harder to solve, the *linear programming (LP) relaxation* removes this restriction.

In a second step, a solution to the ILP is obtained by using a general-purpose ILP solver. Commercial ILP solvers usually rely on implicit enumeration techniques like *branch-and-bound*, using LP relaxation to approximate the optimal solution. However, dropping the integrality condition might alter the structure of the feasible region so significantly, that the LP solution provides a rather weak bound on the optimal value of the integer solution. Therefore, one often tries to "cut" from the linear programming polytope by adding hyperplanes (constraints), so that it closely approximates, at least in the proximity of the optimal solution, the *convex hull* of all feasible solutions to the original problem.

The pioneering work in the area of research on how to describe the convex hull of all feasible solutions by linear inequalities, called *polyhedral combinatorics*, was done by Dantzig, Fulkerson and Johnson (1954) [DFJ54]. They proposed a method for the *traveling salesman problem* (TSP), the archetypical problem in combinatorial optimization. In this problem, we want to determine the order, in which a "salesman" has to visit a number of "cities", such that all cities are visited exactly once, and such that the total length of the tour is minimized. They iteratively added valid inequalities (inequalities that are satisfied by all feasible integer points) that "cut off" the fractional linear programming solution, i.e. hyperplanes that *separate* the fractional point from the polyhedron described by the convex hull of all feasible points. Solving a 49-city instance to optimality at that time, demonstrates the power of their *cutting-plane* methodology.

An alternative approach to approximate the optimal integer solution is based on the observation, that the structure of many hard integer programming problems allows the problems to be interpreted as an "easy" to solve subproblem, complicated by additional constraints. However, relaxing these "complicating" constraints may result in weak bounds on the optimal value. *Lagrangian relaxation* resolves this difficulty by penalizing the violation of the complicating constraints with an associated *Lagrangian multiplier* in the objective function. The Lagrangian relaxation method, as it exists today, was first applied to the traveling salesman problem in 1970 by Held and Karp [HK70]. They produced a sequence of minimal 1-*trees*, a slight variant of spanning trees, that increasingly resemble tours. The sequence of 1-trees is computed via an iterative approach to determine Lagrangian multipliers that yield a best possible lower bound on the optimal TSP solution (the *Lagrangian dual* problem), now known in the literature as *subgradient optimization*. Their technique has served as a basis for many successful branch-and-bound methods to solve TSP to optimality.

To solve the *multiple sequence alignment* problem and the *interval constraint coloring* problem, introduced in the next two sections, we employ a similar methodology. We identify "easy" (combinatorial) to solve subproblems, a *longest path* problem in an acyclic graph in the former, and a *minimum cost network flow* problem in the latter problem, and relax the remaining complicating constraints. The resulting Lagrangian bounds are exploited in a branch-and-bound fashion to obtain an optimal solution. For a comprehensive coverage of the field of *molecular biology* including an introductory exposition of basic necessary terminology, the reader is referred to one of the standard text books in this field, such as [WGWZ92].

## 1.2 Multiple Sequence Alignment (MSA)

In Chapter 2 of this thesis, we present a Lagrangian relaxation approach for the *multiple sequence alignment* problem. A multiple alignment is one common mathematical abstraction of the comparison of multiple biological sequences, like DNA, RNA, or protein sequences. From detected commonalities of a set of sequences (so-called "homologous regions"), one might be able to infer evolutionary trees or deduce structural and functional properties from proteins with similar sequence.

**Problem 1.1** (MULTIPLE SEQUENCE ALIGNMENT (MSA))**.** *Given a set of strings* $\mathcal{S} = \{s^1, s^2, \ldots, s^k\}$ *over an alphabet* $\Sigma$*, an* alignment *of* $\mathcal{S}$ *is a family* $\mathcal{A} = \{\bar{s}^1, \bar{s}^2, \cdots, \bar{s}^k\}$*, where* $\bar{s}^i$ *is obtained by inserting gap symbols "–"* $\notin \Sigma$ *into string* $s^i$*, such that all strings in* $\mathcal{A}$ *have equal length and no column consists entirely of gap symbols. Given an objective function for alignments, the problem calls for a maximum weight alignment.*

Alternatively, instead of maximizing the weight of an alignment, which reflects the *similarity* of the strings, one can also formalize the problem with the objective to minimize the *distances* between the sequences. In the context of DNA sequences, alphabet $\Sigma$ usually contains the four letters `A`, `C`, `G`, and `T`, representing the four different nucleotide bases *adenine*, *cytosine*, *guanine*, and *thymine*, respectively. For protein sequences, alphabet

$\Sigma$ contains 20 letters, representing the 20 naturally occurring amino acids. The following is an example of an alignment of fragments of five hepatitis proteinase sequences.

```
GLVRKNLVQFGVGEKNGSVRWVMNALGVKDDWLLVPSHAYKFEKDYEMMEFYFNRG---
KYPYNTIGNVFVK---GQTSATGVLIGK--NTVLTNRHIAKFANGD-PSKVSFRPSINT
ANTVPYQVSLNS----GYHFCGGSLINS--QWVVSAAHCYK-------SGIQVLGE---
---IAGGEAITT---GGSRCSLGFNVVA---HALTAGHCT-------NISAWSIG----
ALKLEADRLFDVKNEDGDVIGHALAMEG---KVMKPLHVK---------GTIDHP----
```

To be able to find "the best" among all possible alignments, an objective function expresses the quality of an alignment as a numerical value. One of the most commonly used quality measures for the multiple sequence alignment is the so-called *sum-of-pairs* (SP) score (see Carrillo *et al.* [CL88]), which sums the projected pairwise scores of all pairs of sequences in the multiple alignment. It is known that finding a multiple alignment of $k$ sequences with optimal SP score is $\mathcal{NP}$-hard [WJ94]. The problem remains $\mathcal{NP}$-hard if internal gaps are forbidden and sequences can only be shifted relative to each other [Jus01]. It can be approximated within a factor $2 - l/k$, for fixed $l < k$ [BLP97]. It is unknown, whether it admits a polynomial time approximation scheme for some metric objective function. For a nonmetric function that does not satisfy the triangle inequality it has been proven that MSA with SP score is MAX-$\mathcal{SNP}$-hard [Jus01].

A consequence of the hardness of MSA is the predominant use of heuristic methods to align multiple sequences. One of the most successful exact approaches so far was developed by Althaus *et al.* [ACLR06]. They proposed a *branch-and-cut* algorithm, a variant of the branch-and-bound method, where the bound in each node of the enumeration tree is obtained by a cutting plane approach. In the experiments on real-world instances, they observed, however, that in almost all cases the enumeration tree degenerates to a single node (the root node), since the optimal LP solution in the root node was either integer, or its computation time exceeded a given time limit. On the positive side, the three classes of valid inequalities considered define facets for the convex hull of feasible integer solutions while being separable in polynomial time. However, even for a relatively small number of sequences solving the linear programs becomes intractable due to the large number of variables and constraints involved (the considered classes of inequalities are exponentially large). Especially on instances where the sequences exhibit only a moderate degree of similarity, the bounds are too weak to allow for the elimination of a significant number of variables in a preprocessing step. The degree of difficulty of the linear programs is indicated by the fact, that it was more efficient to solve the LPs using the barrier (or interior point) algorithm "from scratch" than reoptimizing using the *dual simplex* after adding violated inequalities to the LP. Even more, the underlying LP solver CPLEX [ILO06] crashed on one of the instances.

This motivates the application of an alternative method to approximate the optimal integer solution in the nodes of the branch-and-bound tree. In fact, the ILP formulation [Rei99] used in their branch-and-cut algorithm exhibits a structure that lets a Lagrangian relaxation approach appear particularly favourable. It can be viewed as an alignment between all pairs of sequences, further complicated by inequalities that

ensure that all pairwise alignments can be integrated into a multiple alignment. The pairwise alignment problem can be solved efficiently by a longest path computation in an acyclic graph. Dualizing a lifted version of the complicating constraints has a major impact on the solution process of the Lagrangian relaxation subproblem, the *extended pairwise alignment* (EPA) problem; the longest path in the acyclic graph is penalized a charge for entering certain "obstacles". Nevertheless, we describe an efficient algorithm to solve the EPA problem, which is of great importance, since we have to solve these subproblems repetitively to obtain near-optimal Lagrangian multipliers via subgradient optimization. To cope with the problem of dualizing an exponential number of inequalities, we adopt the idea that is underlying polyhedral cutting-plane algorithms. In the *relax-and-cut* framework [Luc92], we identify and dualize cuts that are violated by the current Lagrangian solution. Transferred to our context, we start with an empty set of complicating constraints and add (dualize) most violated inequalities (among the exponentially many constraints available) from iteration to iteration to a dynamic *constraint pool*. Instead of taking into account only the current Lagrangian solution, we dualize constraints that are most violated by the *convex combination* of the current Lagrangian solution and a small number of preceding solutions, which increases the convergence rate dramatically. The separation problem reduces to a shortest path problem on a graph with positive edge weights and can, therefore, be solved efficiently. This work is published in part in [AC07, AC08b, AC08a].

Experiments show that the strength of the Lagrangian bounds obtained in the nodes of the branch-and-bound tree combined with the effectiveness of our solution method enables us to solve larger instances to optimality, even if the sequences exhibit a lower degree of similarity.

## 1.3 Interval Constraint Coloring

In Chapter 3 of this thesis, we introduce the *interval constraint coloring problem*, a problem arising in the interpretation of experimental data in biochemistry. One possible method to gain information about the tertiary structure of a protein is to monitor its *hydrogen-deuterium exchange rate* in a $D_2O$ diluted solution via mass spectrometry. These experiments provide aggregate data of the exchange rate of residues within overlapping fragments of the protein. We need to find an assignment of exchange rates to individual residues that is consistent with the observed experimental data to obtain information about the solvent accessibility of various parts of the protein. The interval constraint coloring problem is the mathematical abstraction of this information refinement process.

**Problem 1.2** (IntervalColoring)**.** *Given a set of intervals* $\mathcal{I} \subseteq \{\,[i,j] \mid 1 \leq i \leq j \leq n\}$ *defined on the set* $V = \{1,\ldots,n\}$*, a set of color classes* $[k] = \{1,\ldots,k\}$*, and a requirement function* $r : \mathcal{I} \times [k] \to \mathbb{Z}_+$ *with* $\sum_{c\in[k]} r(I,c) = |I|$ *for all* $I \in \mathcal{I}$*, compute a coloring* $\chi : V \to [k]$*, if one exists, such that for every* $I \in \mathcal{I}$ *we have*

$$\big|\{i \in I \mid \chi(i) = c\}\big| = r(I,c) \text{ for all } c \in [k].$$

For an explanation of notation see next section. A closely related problem is *broadcast scheduling*, where a server must decide which data item to broadcast at each time step in order to satisfy client requests. The literature in broadcast scheduling is vast, and many variations of the problem have been studied (see [CEGK08, GKPS06] and references therein). In the variant we are concerned with here, a client request is specified by a time window $I$ and a data type $A$. The request is satisfied if $A$ is broadcast *at least once* in $I$. The similarities between the two problems should be clear with time steps, time windows and data types in broadcast scheduling playing the respective roles of positions, intervals and colors in interval constraint coloring. There are, however, important differences. First, whereas in broadcast scheduling it does not hurt to broadcast an item more times than the prescribed number, in our problem it does. Second, an interval is satisfied only if *all* the requirements for that interval are satisfied *exactly*, which, undoubtedly, makes our problem significantly harder.

We show that deciding whether a coloring $\chi$ as stated in Problem 1.2 exists is $\mathcal{NP}$-complete if the number of colors $k$ is considered to be part of the input. Following the canonical approach outlined above, we formulated the problem as an integer linear program and solved it using a branch-and-bound algorithm based on LP relaxation. The goal, however, is to provide biochemists with *all* possible candidate solutions, that can be validated in a second step by protein structure prediction tools and by manual inspection. We therefore combine related solutions to equivalence classes in an improved ILP formulation to reduce the running time of our enumeration algorithm. From the total unimodularity of the constraint matrix in the two-color case we conclude that the LP relaxation polytope $\mathcal{P}$ is integral and provide a combinatorial polynomial-time algorithm for this case. However, the complexity status of problem INTERVALCOLORING for three or more colors remains open. For $k \geq 3$ the total unimodularity of the constraint matrix is destroyed, that is, there are instances with fractional vertices. If the length of the intervals is bounded by a constant, an optimal coloring can be determined in polynomial time by a dynamic programming algorithm.

The mathematical model of assigning colors to elements in a set $V$ defines an abstraction of a re-assembly process on peptic fragments. It is based on data (especially, the requirement function $r$), that are obtained only indirectly through monitoring an increase in mass of each of the fragments as the deuteron is added. Physical experimental limitations like deuterium for hydrogen back-exchange, as well as the discretization of the experimental data, e.g. the definition of exchange rates, require the instance data to be approached with caution. Furthermore, the interpretation of the final results with respect to the tertiary structure of the protein is subject to a certain degree of inaccuracy. This motivated the development of algorithms that find solutions that are "close to" the optimal solution.

Provided the LP relaxation polytope $\mathcal{P}$ is not empty, we show how to round a fractional solution to produce a coloring where *all* the requirements are satisfied within a mere additive error of one. In light of our result that INTERVALCOLORING is $\mathcal{NP}$-complete, this is essentially the best one can hope for. Alternatively, we propose a pure combinatorial approach that is based on the polynomial-time algorithm for the two-color case. Due to its heuristic nature, it does not give any guarantee on the quality of the solution. Nev-

ertheless, for instances with arbitrary but fixed number of colors, it provides solutions that approximate the optimal solution well in practice. Compared to the (exact) ILP approach, it achieves an order of magnitude improvement in running time.

Since the data collected in the real experiments usually contain some noise, the problem instance might be infeasible or polytope $\mathcal{P}$ might be even empty. This again leads to the question of "what is the best" coloring that we can get, i.e. what is the coloring exhibiting a minimal possible error? To deal with this problem, we capture the error of a solution by means of additional variables and constraints in the ILP formulation and seek a solution with minimum error. The combinatorial approach for the two-color case optimizes the same objective function. Furthermore, in a Lagrangian relaxation approach for the general case of an arbitrary number of colors, we dualize constraints in such a way, that the resulting Lagrangian subproblem can be solved by the combinatorial approach for the two-color case. Exploiting the resulting bounds in a branch-and-bound algorithm yields exact (optimal) solutions to the general ($\mathcal{NP}$-complete) problem involving an arbitrary number of colors.

Alternatively, we study a variant of the problem where we are asked to find a coloring satisfying the requirements of as many fragments as possible. We prove that this variant is $\mathcal{APX}$-hard even in the two-color case and thus does not admit a polynomial time approximation scheme (PTAS) unless $\mathcal{P} = \mathcal{NP}$. The existence of a *quasi-polynomial time approximation scheme* (QPTAS) would imply that $\mathcal{NP} \subseteq \text{DTIME}[n^{\text{polylog}(n)}]$, which is widely thought not to be the case. We, therefore, allow a coloring to violate the requirement $r$ by a factor $(1+\epsilon)$ and introduce a QPTAS that finds such a coloring for any $\epsilon > 0$ in quasi-polynomial time. This work is published in part in [ACE+08a, ACE+08b].

## 1.4 Notation

We use $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ to denote the sets of natural numbers, integers, and real numbers, respectively. With $M_+$ we refer to the set of nonnegative numbers in $M$ for $M \in \{\mathbb{Z}, \mathbb{R}\}$. The cardinality of a finite set $S$ is denoted by $|S|$. The set that contains the first $n$ natural numbers is denoted by $[n]$, that is,

$$[n] = \{1, 2, \ldots, n\}.$$

For any integer numbers $a$ and $b$, we define the closed interval $[a, b]$ by

$$[a, b] = \{x \in \mathbb{Z} \mid a \leq x \leq b\}.$$

We denote by $\langle v_i \rangle_0^n$ a sequence of $n+1$ elements $v_i$, ordered from $i = 0$ to $i = n$.

For an arbitrary set $R$, $n \in \mathbb{N}$, we use the notation

$$R^n$$

to indicate the set of all *n-tuples* with components from $R$. We always consider a $n$-dimensional *vector* $\mathbf{x} = (x_i)_{i=1,\ldots,n} \in \mathbb{R}^n$, usually denoted by a lower case boldface

character, to be a *column vector*, that is,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}.$$

Additionally, we will also use the more convenient notation $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. The *transpose* of a columns vector is a *row vector*, which will be denoted by $\mathbf{x}'$.

If $\mathbf{x}$ and $\mathbf{y}$ are two vectors in $\mathbb{R}^n$, then we call

$$\mathbf{y}'\mathbf{x} := \sum_{i=1}^{n} x_i y_i$$

the *inner product* of $\mathbf{x}$ and $\mathbf{y}$. We use $\mathbf{0}$ to denote the zero vector and $\mathbf{1}$ to denote the vector with all components equal to 1, i.e.

$$\mathbf{0} = \begin{bmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{1} = \begin{bmatrix} 1 \\ \vdots \\ \vdots \\ \vdots \\ 1 \end{bmatrix}.$$

The dimension of the vectors $\mathbf{0}$ and $\mathbf{1}$ will be clear from the context.

For a vector $\mathbf{x}$, we denote by $\|\mathbf{x}\|$ its Minkowski $\mathscr{L}_2$-norm, i.e. $\|\mathbf{x}\| = \sqrt{x'x}$. For two vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, their Minkowski distance of order 1 is defined as:

$$\|\mathbf{x} - \mathbf{y}\|_1 = \sum_{i=1}^{n} |x_i - y_i|.$$

For a set $R$ and $m, n \in \mathbb{N}$, we denote by

$$R^{m \times n}$$

the set of $m \times n$ *matrices* ($m$ rows, $n$ columns) with entries from $R$. We will denote a matrix $\mathbf{A} \in R^{m \times n}$ by upper case boldface characters and refer to its $(i, j)$th entry by $a_{ij}$ such that $\mathbf{A}$ is of the following form:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

We use the notation $\mathbf{a}_i$ to indicate the vector formed by the entries of the $i$th row of matrix $\mathbf{A}$, that is, $\mathbf{a}_i = (a_{i1}, a_{i2}, \ldots, a_{in})$. The *transpose* of a $m \times n$ matrix $\mathbf{A}$ will be denoted by $\mathbf{A}'$. We denote by $\mathbf{I}$ the *identity matrix*, a square matrix whose diagonal entries are equal to one and whose entries are equal to zero elsewhere.

Chapter 2

# A Lagrangian Relaxation Approach for MSA

## 2.1 Introduction

The importance of *multiple sequence comparison* (of DNA, RNA, or protein sequences) in computational biology is evidenced by the large number of programs that have been developed for the multiple sequence *alignment* (MSA) problem, one common formalization (see next section) of the multiple sequence comparison.

Multiple alignment programs may detect biologically important, yet subtle, similarities in a set of sequences, that might not be apparent when comparing two sequences alone. From these commonalities one might be able to infer evolutionary trees or to group proteins into structurally or functionally related families.

On the other hand, multiple alignment can be associated with solving problems that are inverse to the ones addressed by pairwise sequence comparisons [Gus97]. Pairwise alignments are mostly used to find sequences in a database that share certain patterns with a query sequence, but which might not be known to be biologically related. The inverse problem is to deduce common patterns from known biological relationships.

The alignment method we propose is guided by an objective function that *scores* a multiple alignment by summing the similarity measures of all induced pairwise alignments. This is known as the *sum of pairs (SP)* scoring scheme and is formally defined in the next section. The goodness of a pairwise alignment is measured by summing the scores of opposing characters in the alignment, obtained by a pairwise *scoring matrix*. Taking into account the construct of *gaps*, which is a maximal, consecutive run of characters in one sequence that are not aligned with any character in the other sequence (formal definition below), the model of the *multiple sequence alignment* problem becomes more powerful. The approach we present is able to deal with *affine gap costs*, one of the most commonly used gap cost models in the molecular biology literature. This model charges

a penalty for the existence of a gap (*gap opening cost*) and a further constant penalty for the extension of a gap by one character (*gap extension cost*).

Computing an alignment of $k$ sequences that is optimal in the described model quickly becomes computationally intractable as $k$ increases. For example, dynamic programming algorithms find an optimal alignment of sequences with mean length $n$ in time and space $\mathcal{O}\left(n^k\right)$ [GKS95b, LR00, CL88]. Even more, these methods encounter difficulties when using *truly* affine gap costs and thus consider a simplification instead, the so-called *quasi-affine* gap costs. More complex gap cost functions (nonlinear) add a polylog factor to this complexity [Epp90, LS90].

If the number $k$ of sequences is not fixed, it has been proven by Wang and Jiang [WJ94] by a reduction from *shortest common supersequence* [GJ79b] that the problem of finding the multiple alignment of $k$ sequences with optimal $SP$ score is $\mathcal{NP}$-complete. Hence, it is unlikely that polynomial time algorithms exist and, depending on the problem size, various heuristic approaches have been applied to solve the problem approximately (see, e.g., [NHH00, THG94b, BCG$^+$03]).

In [ACLR06], Althaus *et al.* proposed a (exact) branch-and-cut algorithm for the multiple sequence alignment problem based on an integer linear programming (ILP) formulation of MSA. Since solving the LP relaxation is by far the most expensive part of the algorithm and becomes intractable even for moderately large instances, we propose a Lagrangian relaxation approach to approximate the integer linear program and utilize the resulting bounds on the optimal value in a branch-and-bound framework. We shall assume that the reader is familiar with the Lagrangian relaxation approach to approximate integer linear programs and refer to the seminal work of Held and Karp [HK70] for a detailed exposition. A detailed description of the branch-and-bound technique can, for example, be found in [Wol98].

This chapter is organized as follows. After introducing a formal definition of the multiple sequence alignment problem in the following subsection, we review the polyhedral description of the MSA problem in Section 2.2. In Section 2.3 we relax a slightly modified ILP formulation in a Lagrangian manner in order to obtain upper bounds on the optimal alignment value by solving an *extended pairwise alignment* problem for each pair of sequences. After giving preliminaries concerning the pairwise alignment problem in Section 2.4, we describe in Section 2.5 how to solve the extended pairwise alignment problem via dynamic programming (DP). We reduce the complexity of the DP graph in two steps and proof the correctness of the final algorithm, that uses a *bypass graph*. In Section 2.6 we improve the upper bounds obtained from the Lagrangian relaxation by a modified subgradient optimization method, which we finally exploit in a branch-and-bound algorithm to solve real-world instances in Section 2.7. Finally, in Section 2.8, we conclude our work by a comparison with existing methods and point out future work.

### 2.1.1 Formal Problem Definition

Although we motivated the multiple alignment problem by a comparison of biological sequences (DNA, RNA, or protein sequence), we will generally use the term "string" instead of "sequence", since pure computer science issues are discussed. A *string* is an ordered list of characters. For a string $s$, we denote the substring of $s$ that starts at position $i$ and ends at position $j$ by $s_i, \ldots, s_j$. A prefix of a string $s$ is a substring that starts at position 1.

Let $\mathcal{S} = \{s^1, s^2, \ldots, s^k\}$ be a set of $k > 2$ strings over an alphabet $\Sigma$ and let $\bar{\Sigma} = \Sigma \cup \{-\}$. Given a string $s$, $\|s\|$ denotes the number of characters in the string and $s_l$ the $l$th character of $s$, for $l = 1, \ldots, \|s\|$. We define $n := \sum_{i=1}^{k} \|s^i\|$.

A (global) *multiple alignment* of $\mathcal{S}$ is a set $\mathcal{A} = \{\bar{s}^1, \bar{s}^2, \cdots, \bar{s}^k\}$ of strings over the alphabet $\bar{\Sigma}$, where each string can be interpreted as a row of a two dimensional *alignment matrix*. $\mathcal{A}$ has to satisfy the following properties (see Figure 2.1(a)):

($P1$) The strings in $\mathcal{A}$ all have the same length.

($P2$) Ignoring dashes ("$-$"), string $\bar{s}^i$ is identical to string $s^i$.

($P3$) No column of the alignment matrix consist entirely of dashes.

We say two characters $\bar{s}^i_l$ and $\bar{s}^j_m$ are *aligned* by $\mathcal{A}$ if $l = m$, i.e. they are placed in the same column of the alignment matrix. For a given pairwise scoring matrix $w$ on letters from alphabet $\bar{\Sigma}$, the *sum of pairs (SP)* score for a multiple alignment $\mathcal{A}$ is the sum of the scores of all pairwise projections [CL88]. If the score of a pairwise alignment is obtained by summing the score contributed by each pair of aligned characters, the overall score of the multiple alignment is given by $c(\mathcal{A}) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{h=1}^{l} w(\bar{s}^i_h, \bar{s}^j_h)$, where $l$ denotes the (equal) length of strings in $\mathcal{A}$ and $w(-,-) := 0$.

Depending on the definition of the scoring matrix $w$, the score of an alignment can be naturally interpreted as distances between strings or as a measure of similarity of strings. In the description of our approach, we will formally cast the multiple alignment problem as a maximization ("of weights") problem.

To create biologically meaningful alignments, our objective function includes a term that reflects the notion of a *gap*. For a given alignment $\mathcal{A}$, a gap in $\bar{s}^i$ with respect to $\bar{s}^j$ is a maximal, consecutive run of dashes in $\bar{s}^i$ in the projection of $\mathcal{A}$ to strings $\bar{s}^i$ and $\bar{s}^j$. Associated with each of these gaps is a cost. In the *affine gap cost* model the cost of a single gap of length $q$ is given by the affine function $c_{\text{open}} + qc_{\text{ext}}$, i.e. such a gap contributes a weight of $-c_{\text{open}} - qc_{\text{ext}} = w_{\text{open}} + qw_{\text{ext}}$ to the total weight of the alignment, while $w(x,-) = w(-,x) := 0$ for any $x \in \Sigma$.

Given a multiple alignment $\mathcal{A}$, the *induced pairwise alignment* of two strings $s^i$ and $s^j$ is obtained from $\mathcal{A}$ by extracting the two rows for $s^i$ and $s^j$ from the alignment matrix, from which any two opposing dashes are removed subsequently. Let $\#gaps_{ij}$ and $\#dashes_{ij}$ denote the number of gaps, respectively the total number of dashes, in

the induced alignment of strings $s^i$ and $s^j$. Then we can state the multiple sequence alignment problem with SP score and (truly) affine gap costs formally as follows.

**Problem 2.1** (MULTIPLE SEQUENCE ALIGNMENT (MSA))**.** *Given a set of $k$ strings over an alphabet $\Sigma$, a pairwise scoring matrix $w$ and parameters $w_{\text{open}}$, $w_{\text{ext}}$ of an affine gap weight function, the problem calls for an alignment $\mathcal{A}$ with maximal SP score*

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \left( \sum_{h=1}^{l} w(\bar{s}_h^i, \bar{s}_h^j) + w_{\text{open}}(\#gaps_{ij}) + w_{\text{ext}}(\#dashes_{ij}) \right).$$

We refer to this variant of the problem simply as *multiple sequence alignment* (MSA) throughout the rest of the chapter.

## 2.2 A Polyhedral Description of MSA

In [ACLR06] Althaus *et al.* investigate the polyhedral structure of the convex hull of the set of feasible solutions to the integer linear programming (ILP) formulation of the multiple sequence alignment problem as given by Reinert in [Rei99].

For ease of notation, they define the *gapped alignment graph*, a mixed graph whose node set corresponds to the characters of the strings and whose edge set is partitioned into undirected alignment edges and directed positioning arcs, as follows: $G = (V, E_A \cup A_P)$ with $V = V^i \cup \cdots \cup V^k$ and $V^i = \{u^i_j \mid 1 \leq j \leq \|s^i\|\}$, $E_A = \{uv \mid u \in V^i, v \in V^j, i \neq j\}$ and $A_P = \{(u^i_l, u^i_{l+1}) \mid 1 \leq i \leq k \text{ and } 1 \leq l < \|s^i\|\}$ (see Figure 2.1(b)). Furthermore, $\mathcal{G} = \{(u, v, j) \mid u, v \in V^i, j \neq i\}$ denotes the set of all possible gaps.

An edge in $E_A$ is *realized* by an alignment if its endpoints are placed into the same column of the alignment matrix, i.e the corresponding characters are *aligned*. Accordingly, a gap $(u^i_l, u^i_m, j)$ is *realized* by an alignment if the substring of $s^i$ from position $l$ to position $m$ is aligned to gap characters "–" in string $s^j$, whereas both $s^i_{l-1}$ and $s^i_{m+1}$ are aligned to characters in $s^j$. Arcs in $A_p$ represent the consecutivity of characters within the same string and are independent of the alignment.

In [Rei99], Reinert calls pairs $(E', \mathcal{G}')$, for which there exists an alignment $\mathcal{A}$ such that $E'$ and $\mathcal{G}'$ are the set of edges in $E_A$, respectively gaps in $\mathcal{G}$, that are realized by $\mathcal{A}$, *gapped traces*. Each edge $u^i_l u^j_m \in E_A$ is assigned a weight $w_{u^i_l u^j_m} := w(s^i_l, s^j_m)$ and each gap $(u^i_l, u^i_m, j)$ is assigned the weight $w_{(u^i_l, u^i_m, j)} := w_{\text{open}} + (m - l + 1) \cdot w_{\text{ext}}$, which represents the benefit of realizing that edge or gap. Notice that different alignments might correspond to the same gapped trace $(E', \mathcal{G}')$, but all such alignments have the same score $\sum_{e \in E'} w_e + \sum_{g \in \mathcal{G}'} w_g$.

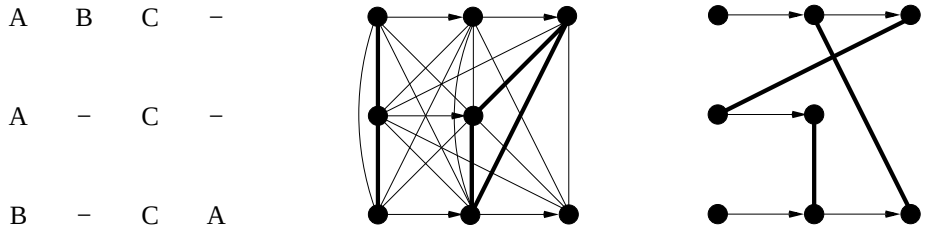
The ILP formulation uses a binary variable $x_e$ for every alignment edge $e \in E_A$, and a binary variable $y_g$ for every possible gap $g \in \mathcal{G}$, indicating whether edge $e$, respectively gap $g$, is realized. Reinert [Rei99] showed that the incidence vectors of the gapped traces are exactly the $\{0, 1\}$-assignments to the variables such that

**(PwA)** we have pairwise alignments between every pair of strings,

**(MixCyc)** there are no *mixed cycles* $M$, i.e. in the subgraph of the gapped alignment graph consisting of the positioning arcs $A_P$ and the realized edges $\{e \in E_A \mid x_e = 1\}$ there is no cycle that respects the direction of the arcs of $A_P$ (and uses the edges of $E_A$ in either direction) and contains at least one arc of $A_P$ (see Figure 2.1(c)):

$$\sum_{e \in M \cap E_A} x_e \leq |M \cap E| - 1,$$

**(Trans)** transitivity is preserved, i.e. if $u$ is aligned with $v$ and $v$ with $w$, then $u$ is aligned with $w$, for $u, v, w \in V$.




(a) Alignment of input sequences

(b) Gapped alignment graph

(c) Ordering conflict

Figure 2.1: (a) A possible alignment $\mathcal{A}$ of the input strings $\mathcal{S} = \{ABC, AC, BCA\}$. $\mathcal{A}$ is represented by $3 \times 4$ alignment matrix. (b) The gapped alignment graph for the strings in $\mathcal{S}$. The thick edges specify alignment $\mathcal{A}$. (c) The alignment edges can not be realized at the same time in an alignment. Together with appropriate arcs of $A_P$, they form a mixed cycle.

These three conditions are easily formulated as linear constraints (see [ACLR06]). Intuitively, mixed cycles can be considered as a generalization of crossing edges that have to be avoided in a bipartite matching representing a pairwise alignment. Given weights $w_e$ associated with variables $x_e$, $e \in E_A$, and gap weights $w_g$ associated with variables $y_g$, $g \in \mathcal{G}$, we denote the problem of finding a solution satisfying conditions (PwA), (MixCyc), and (Trans) (a gapped trace) of maximum weight as $(P)$ and its optimal value as $v(P)$.

As the number of inequalities is exponential, Althaus *et al.* use a cutting plane framework to solve the LP relaxation (all inequalities have a polynomial-time separation algorithm). In the experiments, they observed that the number of iterations in the cutting plane approach can be reduced if they use additional variables $z_{(u,v)}$ for $u \in V^i, v \in V^j, i \neq j$, with the property that $z_{(u,v)} = 1$ iff at least one character of the string of $u$ lying (not strictly) right of $u$ is aligned to a character of the string of $v$ lying (not strictly) left of $v$, i.e. $z_{(u_l^i, u_m^j)} = 1$, iff there is $l' \geq l$ and $m' \leq m$ with $x_{u_{l'}^i u_{m'}^j} = 1$. This condition is captured by the inequalities

$$
\begin{aligned}
0 \leq z \leq 1, \qquad & z_{(u^i_{\|s^i\|}, u^j_1)} = x_{u^i_{\|s^i\|} u^j_1}, \\
& z_{(u^i_l, u^j_m)} \geq z_{(u^i_{l+1}, u^j_m)} + x_{u^i_l u^j_m} \quad \text{and} \\
& z_{(u^i_l, u^j_m)} \geq z_{(u^i_l, u^j_{m-1})} + x_{u^i_l u^j_m}.
\end{aligned} \tag{2.1}
$$

Notice that indicator variables $x_e$ are associated with undirected edges $e = uv$, whereas variables $z_a$ are associated with directed arcs $a = (u, v)$. In the following, we describe the inequalities used in [ACLR06] to enforce (MixCyc). We refrain from explicitly specifying the inequalities enforcing (PwA) and (Trans), as they are not crucial for the understanding of our approach.

Using the additional $z$-variables, we can define facets that guarantee (MixCyc) as follows.

We model the mixed cycles as introduced above by letting $A_A = \{(u,v) \mid u \in V^i, v \in V^j, i \neq j\}$, i.e. for each undirected edge $uv \in E_A$, we have the two directed arcs $(u,v)$ and $(v,u)$ in $A_A$. Then a directed cycle $M \subseteq A_A \cup A_P$ in $(V, A_A \cup A_P)$ that contains at least one arc of $A_P$ uniquely defines a mixed cycle. With a slight abuse of terminology, we drop the distinction between the term *mixed cycle* in its original meaning, namely cycles as defined in condition (MixCyc) having both undirected and directed edges, and their corresponding cycles in $(V, A_A \cup A_P)$, consisting exclusively of directed arcs. The set of all mixed cycles is denoted by $\mathcal{M}$. We show, that for a mixed cycle $M \in \mathcal{M}$ the inequality

$$\sum_{e \in M \cap A_A} z_e \leq |M \cap A_A| - 1 \tag{2.2}$$

is valid. Assume $\sum_{e \in M \cap A_A} z_e = |M \cap A_A|$. Consider an arbitrary arc $(u_l^i, u_m^j) \in M \cap A_A$. Since positioning arcs are directed "from left to right", the arc from $A_A$ preceding $(u_l^i, u_m^j)$ on cycle $M$ must terminate in a node $u_{l'}^i$ with $l' \leq l$. Similarly, the arc from $A_A$ succeeding $(u_l^i, u_m^j)$ on cycle $M$ must originate in a node $u_{m'}^j$ with $m' \geq m$. According to the definition of variable $z_{(u_l^i, u_m^j)}$, there is $l'' \geq l$ and $m'' \leq m$ with $x_{u_{l''}^i u_{m''}^j} = 1$. Thus by replacing every arc $(u_l^i, u_m^j) \in M \cap A_A$ by a (possibly empty) sequence of positioning arcs $(u_l^i, u_{l+1}^i), \ldots, (u_{l''-1}^i, u_{l''}^i)$, followed by edge $u_{l''}^i u_{m''}^j$, followed by a (possibly empty) sequence of positioning arcs $(u_{m''}^j, u_{m''+1}^j), \ldots, (u_{m-1}^j, u_m^j)$, we can construct a mixed cycle, all of which (undirected) alignment edges can not be realized at the same time and which thus represents an ordering conflict (see Figure 2.1(c)). Therefore, (2.2) must hold. These inequalities imply (MixCyc) as $z_{(u,v)} \geq x_{uv}$, and they are used in [ACLR06] to model a *lifted* variant of constraints (MixCyc). In the following discussion, with a *mixed cycle inequality* we refer to inequalities of the form (2.2), unless explicitly mentioned otherwise. They define facets under appropriate technical conditions.

In particular, a mixed cycle inequality can only define a facet if $M$ contains exactly one positioning arc of $A_P$. Assume a mixed cycle $M$ contains at least two arcs of $A_P$ and let $(u_l^i, u_{l+1}^i)$ be one such arc that is succeeded by an alignment edge. Let $M'$ be the cycle obtained from $M$ by replacing arcs $(v, u_l^i), (u_l^i, u_{l+1}^i), (u_{l+1}^i, w)$ by arcs $(v, u_l^i)$ and $(u_l^i, w)$. Then the mixed cycle inequality for $M'$ implies the mixed cycle inequality for $M$ since $z_{(u_l^i, w)} \geq z_{(u_{l+1}^i, w)}$. The constraints (2.2) can be formulated similarly without using the additional $z$-variables.

Based on the new $z$-variables, we derive from problem $(P)$ problem $(P_z)$ by adding constraints (2.1) and replacing (MixCyc) by constraints (2.2) and denote its optimal value by $v(P_z)$.

## 2.3 The Extended Pairwise Alignment Problem (EPA)

Our Lagrangian relaxation approach is based on the integer linear program $(P_z)$ outlined in previous section. Since a single variable $x_{uv}$, $y_{(u,v,j)}$, or $z_{(u,v)}$ involves exactly two strings, we can partition the three classes of variables into sets $X^{i,j}$, $Y^{i,j}$, and $Z^{i,j}$, respectively, that contain variables involving sequences $s^i$ and $s^j$. We observe, that for an arbitrary but fixed pair of strings $s^i$, $s^j$, variables $x_e \in X^{i,j}$ and $y_g \in Y^{i,j}$ in a solution to the ILP yield a description of a pairwise alignment of strings $s^i$ and $s^j$, along with appropriate values for the variables in $Z^{i,j}$ that satisfy (2.1). The constraints (MixCyc) and (Trans) guarantee that all pairwise alignments together form a multiple sequence alignment. We call an assignment of $\{0,1\}$-values to variables in $(X^{i,j}, Y^{i,j}, Z^{i,j})$ such that $(X^{i,j}, Y^{i,j})$ imposes a pairwise alignment, and $Z^{i,j}$ satisfies inequalities (2.1), an *extended pairwise alignment* (EPA) of strings $s^i$ and $s^j$.

**Problem 2.2** (EXTENDED PAIRWISE ALIGNMENT ($\mathrm{EPA}_{\max}$))**.** *Given weights for the variables in $X^{i,j}$, $Y^{i,j}$ and $Z^{i,j}$, problem* $\mathrm{EPA}_{\max}$ *calls for an extended pairwise alignment of strings $s^i$ and $s^j$ of maximum total weight.*

We denote by $(P'_z)$ the problem resulting from relaxing condition (Trans) in $(P_z)$ (during the experiments, it turned out that relaxing condition (Trans) is more efficient in practice than dualizing them). Then the Lagrangian relaxation of $(P'_z)$ relative to the constraints for condition (MixCyc), i.e. inequalities (2.2), with nonnegative Lagrangian multipliers $\lambda$, is an extended pairwise alignment problem. More precisely, for Lagrangian multipliers $\lambda_M \geq 0$ associated with each mixed cycle inequality of $M \in \mathcal{M}$, we have to solve the Lagrangian relaxation problem

$$\sum_{M\in\mathcal{M}} \lambda_M(|M \cap A_A| - 1) \quad + \qquad (LR_\lambda)$$

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{e\in E_A} w_e x_e + \sum_{g\in\mathcal{G}} w_g y_g - \sum_{M\in\mathcal{M}} \lambda_M \sum_{e\in M\cap A_A} z_e \\ \text{such that} & (X^{i,j}, Y^{i,j}, Z^{i,j}) \text{ forms an EPA} \quad \forall\, 1 \leq i < j \leq k. \end{array}$$

Its optimal value, denoted by $v(LR_\lambda)$, is an upper bound on $v(P'_z)$ and therefore also on $v(P_z)$ and $v(P)$.

## 2.4 Pairwise Alignment Preliminaries

Recall how a pairwise alignment with arbitrary gap weights is computed for two strings $s$ and $t$ of length $n_s$ and $n_t$, respectively (without loss of generality we assume $n_t \leq n_s$) [Gus97]. By a simple dynamic programming algorithm, we compute for every $1 \leq l \leq n_s$ and every $1 \leq m \leq n_t$ the optimal alignment of prefixes $s_1 \dots s_l$ and $t_1 \dots t_m$ that aligns $s_l$ and $t_m$, and whose score is denoted by $D(l, m)$. This can be done by comparing all optimal alignments of prefixes $s_1 \dots s_{l'}$ and $t_1 \dots t_{m'}$ for $l' < l$ and $m' < m$, adding the appropriate gap weight to the score $w(s_l, t_m)$ obtained for aligning $s_l$ and $t_m$. If the weight of a gap is an arbitrary function $w(q)$ of its length $q$, the determination of the optimal alignment value $\max_{x \leq n_s, y \leq n_t} [D(x, y) + w(n_s - x) + w(n_t - y)]$, takes time $\mathcal{O}\left(n_s^2 n_t^2\right)$[1].

In the affine gap weight model (a single gap of length $q$ contributes weight $w_{\text{open}} + q w_{\text{ext}}$), we can restrict the dependence of each cell in the dynamic programming matrix to adjacent entries in the matrix by associating more than one variable with each entry as follows. Besides computing $D(l, m)$, we compute the score of the optimal partial alignment of prefixes $s_1 \dots s_l$ and $t_1 \dots t_m$ that aligns character $s_l$ to a character $t_k$ with $k < m$, denoted by $V(l, m)$, and the one that aligns $t_m$ to a character $s_k$ with $k < l$, denoted by $H(l, m)$. Intuitively, a value $V(l, m)$ already includes the opening cost for the gap in string $t$, and therefore the recurrence that relates $V(l, m)$ to $V(l, m+1)$ only adds $w_{\text{ext}}$, but not $w_{\text{open}}$. Each of the terms $D(l, m)$, $V(l, m)$ and $H(l, m)$ can be evaluated by a constant number of references to previously determined values, and thus the running time reduces to $\mathcal{O}(n_s n_t)$.

The pairwise alignment problem can be interpreted as a longest path problem in an acyclic graph, having three nodes $D(l, m)$, $V(l, m)$, and $H(l, m)$ for every pair of prefixes $s_1 \dots s_l$ and $t_1 \dots t_m$. To reflect the structure of the dynamic programming matrix, three nodes $D(l, m)$, $V(l, m)$ and $H(l, m)$ form a *cell* $(l, m)$ and are drawn at coordinates $(l, m)$ in the plane. Each recurrence relation is represented by a weighted (directed) arc in the graph, which we call the *dynamic programming graph*. Figure 2.2 shows three cells of the dynamic programming graph along with arcs between the respective nodes. In the following discussion, the term $\mathscr{S}(l, m)$, with $\mathscr{S} \in \{D, V, H\}$, is used interchangeably to refer both to a node in the dynamic programming graph and to the score of the specific type of alignment it represents.

Each pairwise alignment corresponds to a path through the dynamic programming graph from node $D(0, 0)$ to a node of cell $(n_s, n_t)$. Every arc of the path represents a certain kind of column in the alignment matrix, determined by the type of its target node (Figure 2.2): an *alignment arc* from an arbitrary node in cell $(l-1, m-1)$ to node $D(l, m)$ corresponds to an alignment of characters $s_l$ and $t_m$. Accordingly, a *gap arc* has a target node $V(l, m)$ or $H(l, m)$ and represents a gap opening (denoted by a dashed line in Figure 2.2) or a gap extension. We refer to gap arcs from a node of a cell $(i, j)$ to a node of cell $(i, j+1)$ as *horizontal* (gap) arcs, to gap arcs from a node of a cell $(i, j)$ to a node of cell $(i+1, j)$ as *vertical* (gap) arcs, and we call alignment arcs *diagonal* arcs.

[1] The running time can be reduced to $\mathcal{O}\left(n_s^2 n_t\right)$ by distinguishing three different types of alignments.

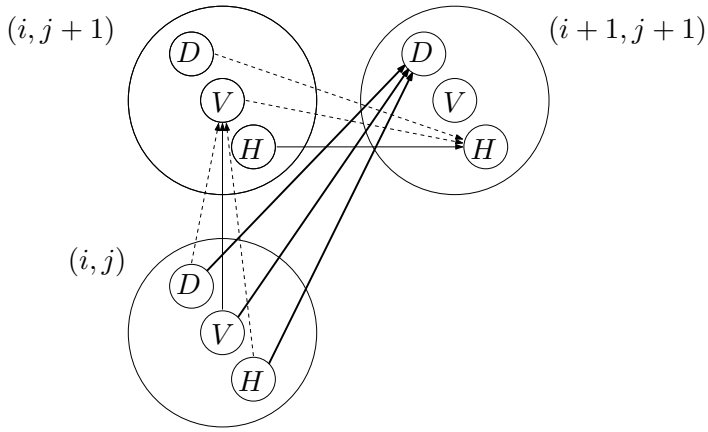



Figure 2.2: Three cells of the dynamic programming graph, each containing three nodes. The type of a node ($D$, $H$, or $V$) represents a certain kind of partial alignment. Note that dependencies (arcs) are only between specific nodes: alignment arcs always run "diagonally", denoted by a thick solid line, while gap arcs only run "vertically" or "horizontally". Arcs representing the opening of a gap are illustrated by a dashed line, while gap extension arcs are drawn with a thin solid line.

## 2.5 EPA$_{\max}$ via Dynamic Programming

In the further discussion, we will always assume that the extended pairwise alignment problem involves strings $s := s^i$ and $t := s^j$, for arbitrary but fixed $1 \leq i < j \leq k$.

Compared to the general pairwise alignment problem discussed in previous section, in the EPA$_{\max}$ problem we are confronted with the difficulty of additional $z$-variables in the objective function. Therefore, assume some variable $z_{(u,v)}$ is multiplied by a non-zero value in the objective function, as the arc $(u, v) \in A_A$ is contained in at least one mixed cycle $M$, to which a non-zero Lagrangian multiplier $\lambda_M$ is associated. Recall that the coefficient of variable $z_{(u,v)}$ in the objective function is $-\sum_{M \in \mathcal{M} | (u,v) \in M} \lambda_M$ (see $(LR_\lambda)$). Then this coefficient is added to the value of the extended pairwise alignment $\mathcal{A}$, or equivalently, to the length of the corresponding path $p$ in the dynamic programming graph, if $\mathcal{A}$ realizes at least one edge, respectively $p$ traverses at least one alignment arc, that enforces $z_{(u,v)} = 1$. Let $u = u_l^i$ and $v = u_m^j$. Then $z_{(u,v)} = 1$ iff there is $l' \geq l$ and $m' \leq m$ such that $x_{u_{l'}^i u_{m'}^j} = 1$ (see definition of variables $z_{(u,v)}$ in (2.1)). In the dynamic program graph, this corresponds to alignment arcs whose target node lie in the lower right rectangle from cell $(l, m)$, to which we will refer as *blue obstacle*.

**Definition 2.1** (Blue Obstacle)**.** *For a cell $(l, m)$ in the dynamic programming graph, a* blues obstacle *$\mathcal{O}_b(l, m)$ contains all cells $(l', m')$, in notation $(l', m') \in \mathcal{O}_b(l, m)$, with $l' \geq l$ and $m' \leq m$.*

Analogously, if $u$ lies in string $s^j$ and $v$ in string $s^i$, this corresponds to alignment arcs whose target node lie in an upper left rectangle, which we will call *red obstacles*.

**Definition 2.2** (Red Obstacle)**.** *For a cell $(l, m)$ in the dynamic programming graph, a* red obstacle *$\mathscr{O}_r(l, m)$ contains all cells $(l', m')$, in notation $(l', m') \in \mathscr{O}_r(l, m)$, with $l' \leq l$ and $m' \geq m$.*

**Definition 2.3** (Origin)**.** *For a blue obstacle $\mathscr{O}_b(l, m)$ or a red obstacle $\mathscr{O}_r(l, m)$, cell $(l, m)$ is called its* origin*.*

To simplify notation, we use variable $o$ to refer to an obstacle of arbitrary "color" and with arbitrary origin. Let the set of all blue and red obstacles that arise from $z$-variables with non-zero coefficients be denoted by $\mathscr{O}_b$ and $\mathscr{O}_r$, respectively, and let $\mathscr{O} = \mathscr{O}_b \cup \mathscr{O}_r$. Then the extended pairwise alignment problem is solvable by a dynamic program in $\mathcal{O}\left(n_s^2 n_t^2 \, |\mathscr{O}|\right)$ time, following the same approach as above: we compute the best extended alignment of all pairs of prefixes $s_1 \dots s_l$ and $t_1 \dots t_m$ that aligns $s_l$ and $t_m$, based on all best extended alignments of strings $s_1 \dots s_{l'}$ and $t_1 \dots t_{m'}$, for $l' < l$ and $m' < m$. We add the appropriate gap weight to $w(s_l, t_m)$ and add the coefficients of all $z$-variables that are enforced to be 1 by setting $x_{u_l^i u_m^j} = 1$ (i.e., by aligning characters $s_l$ and $t_m$) but that are 0-valued if $x_{u_l^i u_m^j} = 0$ and $x_{u_{l'}^i u_{m'}^j} = 1$ (see (2.1) for the definition of $z$-variables). In the dynamic programming graph, these $z$-variables correspond to obstacles $o \in \mathscr{O}$ with $(l, m) \in o$ but $(l', m') \notin o$.

Notice that the information that $s_{l'}$ and $t_{m'}$ are the last two characters aligned in $\mathcal{A}$, that is, $\mathcal{A}$ restricted to $s_{l'+1} \dots s_{l-1}$ and $t_{m'+1} \dots t_{m-1}$ is a maximal, consecutive run of alternate gaps in either string, suffices to determine the set of $z$-variables whose coefficients have to be added to the objective value when aligning $s_l$ and $t_m$.

As introduced above, $z$-variables with non-zero coefficients in the integer linear programming formulation $(LR_\lambda)$ map to obstacles in the dynamic programming graph. Since non-zero coefficients of $z$-variables are negative, we will refer to these coefficients as penalties and associate their *absolute value* with the corresponding obstacles $o$, denoted by $\Lambda(o)$. We say we *charge* (the penalty of) or *pay* for an obstacle, when we really mean that we add the negative coefficient of the corresponding $z$-variable to the objective value. With a slight abuse of notation, we say a node $u$ of the dynamic programming graph is contained in an obstacle $o$, denoted by $u \in o$, if $u$ lies in a cell $(l, m) \in o$.

### 2.5.1 Reducing the Dynamic Programming Graph

Similar as in the affine gap weight model of the general pairwise alignment problem, we reduce the complexity of the dynamic program by decreasing the alignment's history, necessary to determine the benefit of any possible continuation in a partial alignment. The determination of the set of obstacles, whose associated penalty we have to pay when traversing an alignment arc $(u, v)$ in the dynamic programming graph, poses the major problem. A subset of obstacles that node $v$ lies in might have been charged already before, namely those obstacles, that contain the target node of the preceding alignment arc on the current path. However, this arc can not be precomputed in a straightforward way, since the longest path in this context does not have optimal substructure. The

dynamic programming approach from previous section eludes this difficulty by comparing all possible choices of preceding alignment arcs.

Here we will follow a different approach and try to exploit the relation between computed scores in adjacent cells to reduce the dependency on previous nodes of the dynamic programming graph.

Informally, we say that we *enter* an obstacle $o$ with an arc $(u, v)$ if node $v$ lies within obstacle $o$, but source node $u$ is not contained in $o$, i.e. $v \in o$ but $u \notin o$. Then the key idea is to charge the associated penalty $\Lambda(o)$ as soon as we enter an obstacle $o$ along an arc $a$, independent of the type of $a$. In case of an alignment arc, we indeed have to penalize the score by $\Lambda(o)$, but if $a$ is a gap arc, the path might leave this obstacle without traversing any alignment arc in between. We therefore have to add new nodes and edges to the dynamic programming graph that allow the optimal path to bypass obstacles in which it does not traverse any alignment arc. On the positive side, the weight of an alignment arc $a$ does no longer depend on the structure of the previous path, as this strategy allows us to assume that all obstacles that contain the source node of $a$ have already been charged. We only have to pay for obstacles that we enter along $a$.

To be more precise, when traversing an alignment arc with target node $D(x, y)$, we pay for red obstacles with origin $(x', y)$ for $x' \geq x$ and blue obstacles with origin $(x, y')$ for $y' \geq y$. Concerning the traversal of a gap arc $(u, v)$ we observe, that the remainder of the path will not traverse an alignment arc whose target node is contained in an obstacle $o$ if $v$ is a node in the origin of $o$. In this case, the length of the path must not be reduced by $\Lambda(o)$. Thus, for using the gap arc from a node in cell $(x - 1, y)$ to node $H(x, y)$, we only charge the penalties of blue obstacles having origin $(x, y')$ with $y' > y$. Similarly, for using the gap arc from a node in cell $(x, y - 1)$ to node $V(x, y)$ the penalties of red obstacles with origin $(x', y)$ with $x' > x$ are charged. This motivates the following definition.

**Definition 2.4** (Enclosing Obstacles)**.** *The set of enclosing blue obstacles $\mathcal{Q}_b(p)$ of a cell $p = (x, y)$ contains all blue obstacles $\mathscr{O}_b(l, m)$ with $l \leq x, m > y$. Accordingly, $\mathcal{Q}_r(p) = \{\mathscr{O}_r(l', m') \mid l' > x, m' \leq y\}$. Furthermore, we define the overall set of enclosing obstacles by $\mathcal{Q}(p) = \mathcal{Q}_b(p) \cup \mathcal{Q}_r(p)$.*

We say that an obstacle $o$ encloses a node $u$ if $o$ encloses the cell that contains $u$. With a slight abuse of notation we denote the set of enclosing obstacles of a node $u$ by $\mathcal{Q}(u)$, where it is clear from the context whether $u$ refers to a cell or to a node. For a cell $(x, y)$, we will use the more convenient notation $\mathcal{Q}(x, y)$ instead of $\mathcal{Q}((x, y))$.

Using the notion of enclosing obstacles, the set of obstacles that have to be charged when traversing a gap arc $(u, v)$ is given by $\mathcal{Q}(v) \setminus \mathcal{Q}(u)$, i.e. the set of obstacles that enclose node $v$ but not source node $u$.

To summarize, we need to refine our definition of *entering* an obstacle: We *enter* an obstacle $o$ along an alignment arc $(u, v)$ if $v \in o$ but $u \notin o$ (as defined above). But we *enter* an obstacle $o$ along a gap arc $(u', v')$ if $o \in \big(\mathcal{Q}(v') \setminus \mathcal{Q}(u')\big)$. Where it is not crucial

for the understanding of our approach, we will not explicitly point out whether we enter an obstacle along an alignment arc or along a gap arc.

Notice that the set of obstacles enclosing a cell $(x, y)$ contains exactly those obstacles whose associated penalties have to be charged due to the traversal of an alignment arc from a node in cell $(x, y)$ to node $D(x + 1, y + 1)$, but which already have been taken into account by the reduced weight of previous arcs (that entered the obstacles).

Clearly, while traversing a path through the dynamic programming graph, we might leave obstacles that we have entered before:

**Observation 2.1.** *For arbitrary cells $p, q, r$ in the dynamic programming graph we have $\big(\mathcal{Q}(q) \setminus \mathcal{Q}(p)\big) \cup \big(\mathcal{Q}(r) \setminus \mathcal{Q}(q)\big) \supseteq \mathcal{Q}(r) \setminus \mathcal{Q}(p)$.*

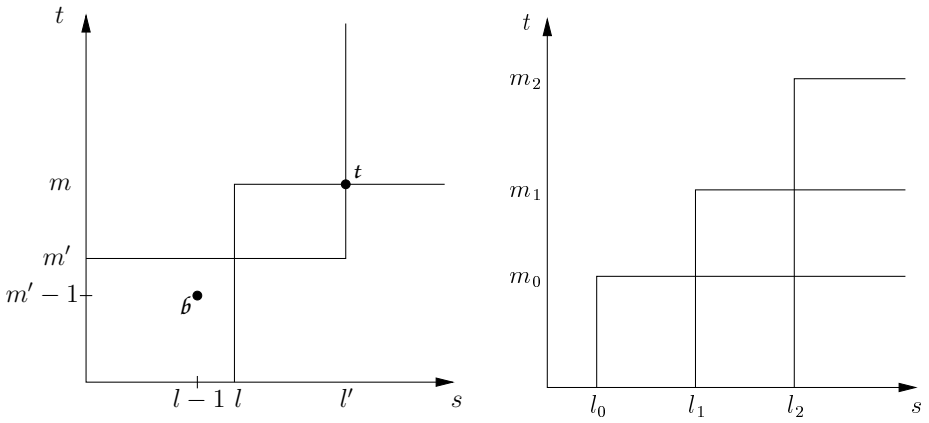
If the preceding alignment arc on the path was always known when traversing an alignment arc $a$ (as is the case with the dynamic programming approach from previous section), the set of obstacles that we additionally have to charge when using arc $a$ could be defined in terms of enclosing obstacles as follows:

**Observation 2.2.** *For $l' < l$ and $m' < m$ let $\mathcal{A}$ be an alignment that aligns character $s_{l'}$ with character $t_{m'}$ and character $s_l$ with character $t_m$, with gaps in between. The set of obstacles whose penalties have to be charged additionally for the alignment of $s_l$ and $t_m$ is equal to the union of $\mathcal{Q}(l - 1, m - 1) \setminus \mathcal{Q}(l', m')$ and the set of obstacles entered with the alignment arc having target node $D(l, m)$.*

Clearly, we charge the penalty of an obstacle at most once, namely when we enter it. Furthermore, from Observations 2.1 and 2.2 it follows that the set of obstacles we pay for contains all obstacles whose penalties indeed have to be subtracted from the score (as the corresponding $z$-variable takes the value 1). However, we even charge $\Lambda(o)$ for an obstacle $o$ if the path enters $o$ but traverses exclusively gap arc in $o$. In that case, the corresponding $z$-variable takes value 0, and we therefore underestimate the length of the path. Hence, we have to ensure that the optimal path is able to "bypass" obstacles in which it does not traverse any alignment arc and which thereby do not have to be paid for.

We accomplish this by adding new nodes and arcs to the dynamic programming graph. Additionally, we compute, for every pair of prefixes $s_1 \dots s_l$ and $t_1 \dots t_m$, a fourth value $B(l, m)$ denoting the value of the optimal extended alignment that aligns either character $s_l$ to "-" strictly left from $t_m$ or character $t_m$ to "-" strictly left from $s_l$. This new value $B(l, m)$ is represented by a new $B$-node in cell $(l, m)$ in the dynamic programming graph. In other words, when proceeding from a $B$-node along a gap arc $(u, v)$, we assume that the gap opening weight for either string was already added, and therefore the weight of such an edge will only account for the extension weight of that gap (plus the penalties of possible obstacles entered along $(u, v)$). We extend the definition of *gap arcs* to arcs having a target node of type $B$.

Before we introduce the new edges formally, we need some basic definitions.




(a) Conflicting obstacles (b) Dominating obstacles

Figure 2.3: (a) A pair of conflicting obstacles, together with its base $\mathfrak{b}$ and its tail $\mathfrak{t}$. (b) Obstacle $\mathscr{O}_b(l_0, m_0)$ dominates obstacles $\mathscr{O}_b(l_1, m_1)$ and $\mathscr{O}_b(l_2, m_2)$. Obstacle $\mathscr{O}_b(l_1, m_1)$ is minimal in $\mathcal{D}(\mathscr{O}_b(l_0, m_0))$.

**Definition 2.5** (Conflicting Obstacles)**.** *We call a pair of a blue obstacle $\mathscr{O}_b(l, m)$ and a red obstacle $\mathscr{O}_r(l', m')$* conflicting *if $l' \geq l$ and $m' \leq m$.*

**Definition 2.6** (Base, Tail)**.** *The* base $\mathfrak{b}(\mathscr{O}_b(l, m), \mathscr{O}_r(l', m'))$ *of a pair of conflicting obstacles is defined as cell $(l - 1, m' - 1)$, the* tail $\mathfrak{t}(\mathscr{O}_b(l, m), \mathscr{O}_r(l', m'))$ *as cell $(l', m)$.*

**Definition 2.7** (Dominating Obstacle)**.** *We say a cell $(l, m)$* dominates *a cell $(l', m')$, denoted by $(l, m) < (l', m')$, if $l < l'$ and $m < m'$ $\big((l, m) \leq (l', m')$, if $l \leq l'$ and $m \leq m'\big)$. Accordingly, a blue (red) obstacle $\mathscr{O}_{b(r)}(l, m)$* dominates *an obstacle $\mathscr{O}_{b(r)}(l', m')$ iff origin $(l, m)$ dominates origin $(l', m')$.*

**Definition 2.8** (Minimal Obstacle)**.** *A blue (red) obstacle is* minimal *in set $\hat{\mathscr{O}}_b \subseteq \mathscr{O}_b$ ($\hat{\mathscr{O}}_r \subseteq \mathscr{O}_r$) if it is not dominated by any other obstacle in $\hat{\mathscr{O}}_b$ ($\hat{\mathscr{O}}_r$).*

We denote the set of obstacles that are dominated by a given obstacle $o$ by $\mathcal{D}(o)$. Figures 2.3(a) and 2.3(b) illustrate the newly introduced notion.

For a path $p$ corresponding to a given alignment $\mathcal{A}$ we formally define the set of obstacles which $p$ is not allowed to enter.

**Definition 2.9** (Forbidden Obstacle)**.** *Given an extended alignment $\mathcal{A}$, we call the set of obstacles whose corresponding $z$-variable is 0 under $\mathcal{A}$ the* forbidden obstacles *with respect to $\mathcal{A}$.*

Note that foribidden obstacles are exactly those obstacles that we do not have to pay, provided the path traverses exclusively alignment arcs with target nodes $D(l, m)$ such that alignment edge $u_l^i u_m^j$ is realized by $\mathcal{A}$.

Our modification of the underlying dynamic programming graph is based on the following observation. For an optimal extended alignment $\mathcal{A}$ of strings $s$ and $t$ let $l' < l$ and $m' < m$ be such that $\mathcal{A}$ restricted to $s_{l'+1} \dots s_{l-1}$ and $t_{m'+1} \dots t_{m-1}$ is a maximal, consecutive run of alternate gaps in either string. We have to ensure the existence of a path from node $D(l', m')$ to the appropriate node of cell $(l-1, m-1)$ that does not enter any forbidden obstacle. The type of node $v$ in cell $(l-1, m-1)$ the optimal path will hit is determined as follows:

$$v = \begin{cases} B(l-1, m-1) & \text{if } l' < l, m' < m \\ H(l-1, m-1) & \text{if } l' < l, m' = m \\ V(l-1, m-1) & \text{if } l' = l, m' < m \end{cases}$$

Starting from node $D(l', m')$, we traverse gap arcs in the original dynamic programming graph (extended by the B-nodes indicating that we have added the opening weight of gaps in both strings) until we would enter with any of the outgoing gap arcs a forbidden obstacle. This motivates the addition of a $B$-node to each cell, as it allows us to use gap arcs in either string alternatively without adding the gap opening weight multiple times (see Figure 2.4).

Notice that we cannot enter and leave an obstacle using exclusively gap arcs of one type, e.g. with horizontal gap arcs, we can enter blue obstacles, but not leave them. In particular, if $l' < l$ and $m' = m$ or if $l' = l$ and $m' < m$, the path can reach node $H(l-1, m-1)$, respectively node $V(l-1, m-1)$, without entering any forbidden obstacle.

Therefore, consider the case where $l' < l$ and $m' < m$, that is, we want to reach node $B(l-1, m-1)$. Let us further assume that we cannot proceed from a node in cell $(l_\mathfrak{b}, m_\mathfrak{b})$, $l_\mathfrak{b} < l$, $m_\mathfrak{b} < m$, without entering a forbidden obstacle (see Figure 2.4). Clearly, $(l_\mathfrak{b}, m_\mathfrak{b})$ is the base of a pair of conflicting forbidden obstacles. More precisely, $l_\mathfrak{b}$ is the smallest value with $l_\mathfrak{b} \geq l'$ such that there is a pair of conflicting forbidden obstacles with base $(l_\mathfrak{b}, m'')$. Similarly, $m_\mathfrak{b}$ is the smallest value with $m_\mathfrak{b} \geq m'$ such that there exists a pair of conflicting forbidden obstacles with base $(l'', m_\mathfrak{b})$. In other words, for the base $(l'_\mathfrak{b}, m'_\mathfrak{b})$ of every pair of conflicting forbidden obstacles with $(l', m') \leq (l'_\mathfrak{b}, m'_\mathfrak{b})$, it holds $(l_\mathfrak{b}, m_\mathfrak{b}) \leq (l'_\mathfrak{b}, m'_\mathfrak{b})$.

Analogously, we can argue that there is a tail $(l_\mathfrak{t}, m_\mathfrak{t})$ with $(l', m') < (l_\mathfrak{t}, m_\mathfrak{t}) \leq (l-1, m-1)$ of a pair of conflicting obstacles from which we can reach node $B(l-1, m-1)$ without entering a forbidden obstacle (see Figure 2.4). From the transitivity of "$\leq$" and the fact that the base of a pair of conflicting obstacles always dominates the tail of that pair it follows $(l_\mathfrak{b}, m_\mathfrak{b}) < (l_\mathfrak{t}, m_\mathfrak{t})$.

Therefore, the insertion of arcs from the four nodes of the base of every pair of conflicting obstacles $(o_b, o_r)$ to the $B$-node of the target of every pair $(o'_b, o'_r)$ such that $\mathfrak{b}(o_b, o_r) \leq \mathfrak{b}(o'_b, o'_r)$ and $\mathfrak{t}(o_b, o_r) \leq \mathfrak{t}(o'_b, o'_r)$, would enable us to "jump over" obstacles that we do not have to pay. The weights for these arcs are determined by the cost of the gaps leading from $\mathfrak{b}(o_b, o_r)$ to $\mathfrak{t}(o'_b, o'_r)$ plus the penalties implied by obstacles enclosing $\mathfrak{t}(o'_b, o'_r)$, but not $\mathfrak{b}(o_b, o_r)$.

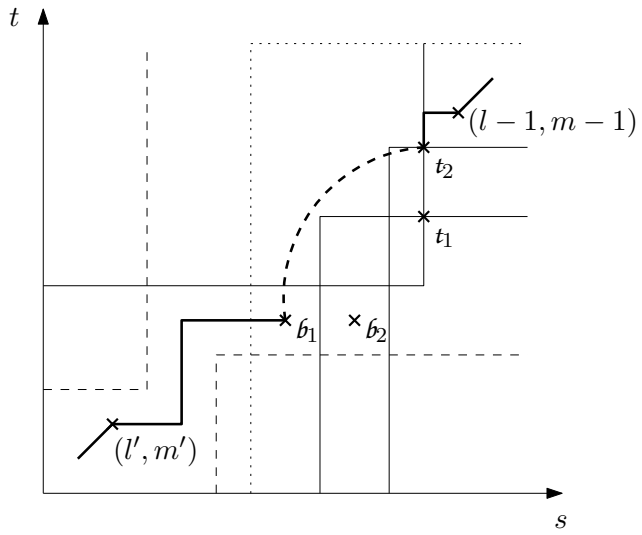



Figure 2.4: In this illustration of the dynamic programming graph, cells are contracted to single points, such that arcs appear between cells instead of nodes. The path (solid thick line) can bypass the dashed obstacles within the graph, as they are not in conflict with any other obstacle. The dotted obstacle is not a forbidden obstacle since it contains the diagonal alignment arc with target node $D(l, m)$. From base $b_1 = (l_b, m_b)$ the path cannot proceed along a gap arc without entering any forbidden obstacle. A new arc from B-node in cell $b_1$ to B-node in cell $t_2 = (l_t, m_t)$ allows the path to avoid obstacles whose penalty it does not have to pay.

The overall structure of the resulting graph, whose longest path from node $D(0, 0)$ to a node in cell $(n_s, n_t)$ determines the score $C(n_s, n_t)$ of the optimal extended pairwise alignment (see recurrence 2.8), can be described in terms of the following recurrences. The base case is:

$$D(0,0) = 0 \tag{2.3}$$

$$V(l,0) = l \cdot w_{\text{ext}} + w_{\text{open}} - \sum_{o \in \mathcal{Q}(l,0)} \Lambda(o) \tag{2.4}$$

$$H(0,m) = m \cdot w_{\text{ext}} + w_{\text{open}} - \sum_{o \in \mathcal{Q}(0,m)} \Lambda(o) \tag{2.5}$$

$$B(l,1) = (l+1) \cdot w_{\text{ext}} + 2 \cdot w_{\text{open}} - \sum_{o \in \mathcal{Q}(l,1)} \Lambda(o) \tag{2.6}$$

$$B(1,m) = (m+1) \cdot w_{\text{ext}} + 2 \cdot w_{\text{open}} - \sum_{o \in \mathcal{Q}(1,m)} \Lambda(o) \tag{2.7}$$

$D(l, m)$ is undefined when exactly one of $l$ and $m$ is 0; or in other words, there is no alignment arc with a target node $D(l, 0)$, $l > 0$, or with a target node $D(0, m)$, $m > 0$. The same holds for gap arcs with target node $V(l, 1)$, $H(1, m)$, $B(l, 0)$, or $B(0, m)$. In the following general recurrences we set an undefined value to $-\infty$, which translates to an empty set of outgoing arcs from a node in the dynamic programming graph that corresponds to an undefined value.

$$C(l,m) = \max\{D(l,m), V(l,m), H(l,m), B(l,m)\} \tag{2.8}$$

with

$$D(l,m) = C(l-1,m-1) + w(s_l,t_m) - \sum_{o=\mathscr{O}_r(i,m),i\geq l} \Lambda(o) - \sum_{o=\mathscr{O}_b(l,j),j\geq m} \Lambda(o) \tag{2.9}$$

$$V(l,m) = \max\left\{ \begin{array}{l} D(l,m-1) + w_{\text{ext}} + w_{\text{open}} \\ V(l,m-1) + w_{\text{ext}} \end{array} \right\} - \sum_{o=\mathscr{O}_r(i,m),i>l} \Lambda(o) \tag{2.10}$$

$$H(l,m) = \max\left\{ \begin{array}{l} D(l-1,m) + w_{\text{ext}} + w_{\text{open}} \\ H(l-1,m) + w_{\text{ext}} \end{array} \right\} - \sum_{o=\mathscr{O}_b(l,j),j>m} \Lambda(o) \tag{2.11}$$

$$B(l,m) = \max\left\{ \begin{array}{l} \mathfrak{J}(l,m) \\ B(l-1,m) + w_{\text{ext}} - \sum_{o=\mathscr{O}_b(l,j),j>m} \Lambda(o) \\ B(l,m-1) + w_{\text{ext}} - \sum_{o=\mathscr{O}_r(i,m),i>l} \Lambda(o) \\ V(l-1,m) + w_{\text{ext}} + w_{\text{open}} - \sum_{o=\mathscr{O}_b(l,j),j>m} \Lambda(o) \\ H(l,m-1) + w_{\text{ext}} + w_{\text{open}} - \sum_{o=\mathscr{O}_r(i,m),i>l} \Lambda(o) \end{array} \right. \tag{2.12}$$

where

$$\mathfrak{J}(l,m) = \max_{(l',m')\in\mathcal{B}_{l,m}} \left\{ \begin{array}{l} D(l',m') + qw_{\text{ext}} + 2w_{\text{open}} \\ V(l',m') + qw_{\text{ext}} + w_{\text{open}} \\ H(l',m') + qw_{\text{ext}} + w_{\text{open}} \\ B(l',m') + qw_{\text{ext}} \end{array} \right\} - \sum_{o\in\mathcal{Q}(l,m)\setminus\mathcal{Q}(l'm')} \Lambda(o), \tag{2.13}$$

if cell $(l,m)$ is the target of a pair of conflicting obstacles, and $\mathfrak{J}(l,m) := -\infty$ otherwise.

We denote by $q$ the Minkowski's $\mathscr{L}_1$ distance between the cells containing the source node, respectively target node, of a newly inserted arc, i.e. $q = \|(l,m) - (l',m')\|_1$. Provided $(l,m) = \mathfrak{t}(o_b,o_r)$ for some pair of conflicting obstacles $(o_b,o_r)$, set

$$\mathcal{B}_{l,m} = \{\mathfrak{b}(o_b',o_r') \mid \mathfrak{b}(o_b',o_r') \leq \mathfrak{b}(o_b,o_r) \text{ and } \mathfrak{t}(o_b',o_r') \leq (l,m)\}.$$

As the number of conflicting obstacles is at most $|\mathscr{O}|^2$, the number of additional arcs is at most $\mathcal{O}\left(|\mathscr{O}|^4\right)$, and hence the running time of the dynamic programming algorithm is $\mathcal{O}\left(n_s n_t + |\mathscr{O}|^4\right)$.

### 2.5.2 The Bypass Graph

To further reduce the number of additional arcs (dependencies) in our dynamic programming graph, we augment the original dynamic programming graph with a *bypass graph* (BPG), which is correlated to the transitive reduction of the induced subgraph on the set of arcs added in last section.

The nodes of the bypass graph (formal definition below) represent pairs of conflicting obstacles. In the following, the *base* $\mathfrak{b}(v)$ and the *target* $\mathfrak{t}(v)$ of a node $v$ in the BPG will denote the base or target, respectively, of the pair of conflicting obstacles node $v$ represents (see Definition 2.6). Intuitively, reaching a node $v$ in the bypass graph along a path $p$, with $\mathfrak{b}(v) = (l, m)$ and $\mathfrak{t}(v) = (l', m')$, can be interpreted as having a consecutive run of alternate gaps $s_{g+1}, \ldots, s_{l'}$ and $t_{h+1}, \ldots, t_{m'}$, for some $g \leq l$ and $h \leq m$. In particular, there is an arc of weight 0 from each node $v$ in the BPG to the $B$-node of cell $\mathfrak{t}(v)$. Note that in this case, the last alignment arc on path $p$ has target node $D(g, h)$.

The major difficulty lies in the assignment of weights to the arcs in the bypass graph. As the overall goal is to minimize the number of additional arcs, the weight of an arc must not be based on any "global" assumption concerning the preceding alignment arc on the current path. Instead, the weight of an arc can only incorporate "local" information of the form $(g, h) \leq (l, m)$ with $D(g, h)$ being the target node of the preceding alignment arc and $(l, m)$ being the base of the corresponding pair of conflicting obstacles (see above).

Recall that for cells $(i, j)$ and $(l, m)$, $\|(i, j) - (l, m)\|_1$ denotes Minkowski's $\mathscr{L}_1$ distance between points $(i, j)$ and $(l, m)$ in the plane. An arc $(v, w)$ between BPG nodes $v$ and $w$ models the extension of a gap in exactly one of the two strings by $\|\mathfrak{t}(w) - \mathfrak{t}(v)\|_1$ characters, i.e. by traversing an arc $(v, w)$ in the BPG tail $\mathfrak{t}(v)$ moves vertically upwards or horizontally to the right towards $\mathfrak{t}(w)$. Contrariwise, the corresponding base moves simultaneously to the right, respectively upwards. Therefore, the weight of a BPG arc $(v, w)$ must take into account

(a) the cost of extending a gap by $\|\mathfrak{t}(w) - \mathfrak{t}(v)\|_1$ characters,

(b) the penalties associated with obstacles in $\mathcal{Q}(\mathfrak{t}(w)) \setminus \mathcal{Q}(\mathfrak{t}(v))$,

(c) a compensation for penalties associated with forbidden obstacles that we are leaving when proceeding from $\mathfrak{t}(v)$ to $\mathfrak{t}(w)$.

The third part is required due to the fact that no assumption can be made when traversing a BPG arc $(v', w')$ concerning the continuation of the path from $\mathfrak{t}(w')$, and therefore the penalty of a forbidden obstacle might have been payed in (b) by a previous arc $(v', w')$. At the same time, the inexact information that the last alignment arc is "to the lower left" of $\mathfrak{b}(v)$ does not allow us to decide for every obstacles we are leaving in (c) whether it is a forbidden obstacle. Only those among the obstacles we are leaving in (c) can be identified as forbidden obstacles with certainty that are not enclosing $\mathfrak{b}(v)$. Even more, when continuing from node $w$, we must not make any assumptions about the incoming arc along which the path hit node $w$ and thus we will have to further relax

our constraint concerning the position of the last alignment arc to that it is to the lower left of $\mathfrak{b}(w)$. As a consequence, we have to ensure, that the length of the path leading to $w$ is not penalized by any forbidden obstacle that encloses both $\mathfrak{b}(w)$ and $\mathfrak{t}(w)$. Such an obstacle $o$ would not be compensated in (c) for edge $(v, w)$, and for any subsequent arc it could not be decided whether $o$ is a forbidden obstacle. This is illustrated in Figure 2.5. As we will see below, the definition of the arc set ensures the there always exists of a path through the BPG along which penalized and compensated obstacles cancel out each other in such a way, that exactly the non-forbidden obstacles remain paid for.

**Definition 2.10** (Bypass Graph)**.** *We define the* Bypass Graph (BPG) $G = (\mathcal{V}, \mathcal{E}, l)$ *with edge set* $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ *and length function* $l\colon \mathcal{E} \to \mathbb{R}$ *as follows. The vertex set* $\mathcal{V}$ *contains a node* $v$ *for all pairs of conflicting obstacles. Let* $v^b$ *and* $v^r$ *denote the blue, respectively red obstacle of the pair of conflicting obstacles represented by* $v$*.* $\mathcal{E} = \mathcal{E}_b \cup \mathcal{E}_r$*, where* $\mathcal{E}_b = \{(v, w) \mid v^r = w^r$ *and* $w^b$ *is minimal in* $\mathcal{D}(v^b)\}$ *and* $\mathcal{E}_r = \{(v, w) \mid v^b = w^b$ *and* $w^r$ *is minimal in* $\mathcal{D}(v^r)\}$*. Every edge* $(v, w) \in \mathcal{E}_\kappa$*,* $\kappa \in \{b, r\}$*, is assigned a length*

$$l((v,w)) = w_{ext} \cdot \|\mathfrak{t}(w) - \mathfrak{t}(v)\|_1 - \sum_{o \in \mathcal{Q}^-(v,w)} \Lambda(o) + \sum_{o \in \mathcal{Q}^+(v,w)} \Lambda(o),$$

*where*

$$\mathcal{Q}^-(v,w) = \mathcal{Q}(\mathfrak{t}(w)) \setminus \mathcal{Q}(\mathfrak{t}(v)) \quad \text{and}$$
$$\mathcal{Q}^+(v,w) = \{\mathscr{O}_\kappa(i,j) \in \mathcal{Q}_\kappa(\mathfrak{t}(v)) \setminus \mathcal{Q}_\kappa(\mathfrak{t}(w)) \mid w^\kappa = \mathscr{O}_\kappa(l,m), i \geq l, j \leq m\}.$$

The length of an arc $(v, w) \in \mathcal{E}_b$ in the BPG is motivated in Figure 2.6. We connect the bypass graph to the original dynamic programming graph (including $B$-nodes) by arcs as follows: If $(i, j)$ is the base of a pair of conflicting obstacles with corresponding node $v \in \mathcal{V}$ in the BPG, we add arcs from all nodes in cell $(i, j)$ to $v$ (recursion formula (2.15)) and by arcs from all $v \in \mathcal{V}$ to the $B$-node of tail $\mathfrak{t}(v)$ (formula (2.14)).

The overall structure of the resulting graph can be described by the same recurrence relations as in previous section, except that nodes of type $B$ might be the target of arcs originating in a node of the bypass graph:

$$B(l,m) = \max \begin{cases} \max_{v \in \mathcal{V}: \mathfrak{t}(v) = (l,m)} \{\delta(v)\} \\ B(l-1,m) + w_{\text{ext}} - \sum_{o = \mathscr{O}_b(l,j), j > m} \Lambda(o) \\ B(l,m-1) + w_{\text{ext}} - \sum_{o = \mathscr{O}_r(i,m), i > l} \Lambda(o) \\ V(l-1,m) + w_{\text{ext}} + w_{\text{open}} - \sum_{o = \mathscr{O}_b(l,j), j > m} \Lambda(o) \\ H(l,m-1) + w_{\text{ext}} + w_{\text{open}} - \sum_{o = \mathscr{O}_r(i,m), i > l} \Lambda(o) \end{cases} \tag{2.14}$$

where

$$\delta(v) = \max \left\{ \begin{array}{l} \max_{u:(u,v) \in \mathcal{E}} \{\delta(u) + l((u,v))\} \\ \left\{ \begin{array}{l} D(\mathfrak{b}(v)) + q w_{\text{ext}} + 2 w_{\text{open}} \\ V(\mathfrak{b}(v)) + q w_{\text{ext}} + w_{\text{open}} \\ H(\mathfrak{b}(v)) + q w_{\text{ext}} + w_{\text{open}} \\ B(\mathfrak{b}(v)) + q w_{\text{ext}} \end{array} \right\} - \sum_{o \in \mathcal{Q}(\mathfrak{t}(v)) \setminus \mathcal{Q}(\mathfrak{b}(v))} \Lambda(o), \end{array} \right. \tag{2.15}$$

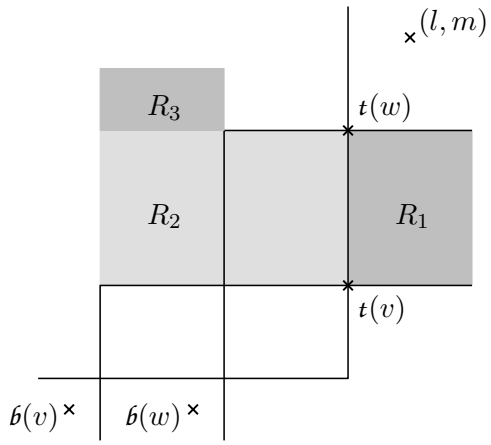



Figure 2.5: Assume a path traverses arc $(v, w)$ in the BPG and returns to the original DP graph at node $B(l, m)$. Only cells are shown, and the contained nodes are omitted. When proceeding from $t(v)$ to $t(w)$ red obstacles whose origin lie in $R_1$ are entered and hence their penalties have to be charged. At the same time, blue obstacles that originate in rectangle $R_2$ are left. Since the last alignment arc is "to the lower left" of $\mathfrak{b}(v)$, those obstacles are forbidden obstacles, and we thus compensate (add) their associated penalties. After reaching node $w$, we have to relax the constraint concerning the location of the last alignment arc to be "to the lower left" of $\mathfrak{b}(w)$. As a consequence, we would not be able to decide whether we have to compensate the penalty of obstacles originating in $R_3$. Thus, there have to exist arcs in the BPG that enable us to reach node $B(l, m)$ on a path, on which these regions do not contain the origins of any obstacles.

with $q = \|t(v) - \mathfrak{b}(v)\|_1$. Values $D$, $V$, and $H$ are defined by recurrences (2.9)-(2.11), the base cases by relations (2.3)-(2.7). The longest path from node $D(0, 0)$ to a node in cell $(n_s, n_t)$ determines the score $C(n_s, n_t)$ of the optimal extended pairwise alignment.

### Complexity

Since there are at most $|\mathscr{O}|^2$ conflicting pairs of obstacles, the number of additional nodes $|\mathcal{V}|$ is at most $|\mathscr{O}|^2$. From Definition 2.10 it follows immediately that the number of additional arcs $|\mathcal{E}|$ is at most $\mathcal{O}\left(|\mathscr{O}|^3\right)$, as an edge of the BPG is defined by three obstacles. Therefore, the running time to compute an optimal solution to problem EPA$_{\max}$ involving strings $s$ and $t$ with $\|s\| = n$ and $\|t\| = m$ is $\mathcal{O}\left(nm + |\mathscr{O}|^3\right)$.

We improve the practical performance of our algorithm for solving the extended pairwise alignment problem by applying an $A^*$-approach: Notice that the scores $D(l, m)$, $V(l, m)$, $H(l, m)$, and $B(l, m)$ for an arbitrary Lagrangian multiplier vector $(\lambda_M)_{M \in \mathcal{M}} \geq \mathbf{0}$ can be at most the scores when all multipliers $\lambda_M$ are set to 0. Therefore, the length of a longest path from a node in any cell $(l, m)$ to a node in cell $(n_s, n_t)$ determined for $\lambda = \mathbf{0}$ provides a heuristic estimate for all instances with $\lambda \geq \mathbf{0}$, which is monotonic, and thus the first path found from $(0, 0)$ to $(n_s, n_t)$ is optimal.

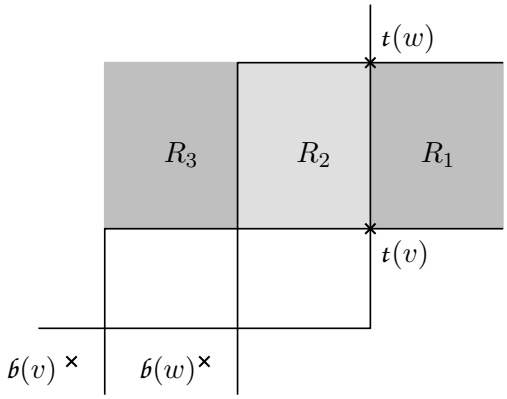



Figure 2.6: Nodes $v$ and $w$ in the BPG corresponding to pairs of conflicting obstacles. If no blue obstacle with origin in $R_3$ exists, $w^b$ is minimal in $\mathcal{D}(v^b)$ and thus $(v, w) \in \mathcal{E}_b$. In this example, $\mathcal{Q}^-(v, w)$ contains exclusively red obstacles with origin in $R_1$ (obstacles we are entering), and $\mathcal{Q}^+(v, w)$ contains blue obstacles with origin in $R_2$ (obstacles we are leaving).

### 2.5.3 Proof of Correctness

The overall dynamic programming graph is obtained by augmenting the original dynamic programming graph (extended by the $B$-nodes) with the bypass graph introduced in the previous section. In the following, we will refer to the original dynamic programming graph extended by the $B$-nodes simply as the dynamic programming graph (DP graph). A reference to the overall construction will be explicitly characterized as such.

Let $p$ be a path starting at $D(0, 0)$ through the DP graph and the bypass graph ending at a node $D(i, j)$. Furthermore, let $D(l, m)$ be the last node of type $D$ on $p$ preceding $D(i, j)$, and let $\hat{p}$ be the prefix of $p$ up to node $D(l, m)$. If $d$ denotes the score of the alignment of prefixes $s_1 \dots s_l$ and $t_1 \dots t_m$ induced by $\hat{p}$, the score of the alignment induced by $p$ is

$$d + w(s_i, t_j) + w_{\text{ext}} \cdot \|(i-1, j-1) - (l, m)\|_1 + r \cdot w_{\text{open}} - \sum_{o \in \mathcal{R}} \Lambda(o) - \sum_{o \in \mathcal{B}} \Lambda(o),$$

where

$$r = \begin{cases} 0 & \text{if } (i-1, j-1) = (l, m), \\ 2 & \text{if } i-1 > l \text{ and } j-1 > m, \\ 1 & \text{otherwise,} \end{cases}$$

and

$$\mathcal{R} = \{\mathscr{O}_r(i', j') \mid i \le i', m < j' \le j\},$$
$$\mathcal{B} = \{\mathscr{O}_b(i', j') \mid j \le j', l < i' \le i\}.$$

The following theorem shows that the optimal extended pairwise alignment of $s$ and $t$ can be determined by iterating over all $D(x, y)$, $1 \leq x \leq n_s$ and $1 \leq y \leq n_t$, adding the appropriate weight for the remaining gaps. Alternatively, the score of the optimal extended pairwise alignment of $s$ and $t$ corresponds to the maximum of $D(n_s, n_t)$, $V(n_s, n_t)$, $H(n_s, n_t)$, and $B(n_s, n_t)$.

**Theorem 2.1.** *Given strings $s$ and $t$ of length $n_s$ and $n_t$, respectively, $D(x, y)$, for $1 \leq x \leq n_s$ and $1 \leq y \leq n_t$, is equal to the score of an optimal extended pairwise alignment of prefixes $s_1 \dots s_x$ and $t_1 \dots t_y$ that aligns $s_x$ with $t_y$.*

*Proof.* Consider arbitrary but fixed indices $1 \leq l < l' < n_s$ and $1 \leq m < m' < n_t$ and assume that $D(l, m)$ and $D(l'+1, m'+1)$ are the targets of two realized alignment edges with gaps in between. We will show in Lemma 2.2 and Lemma 2.3 that

(a) there is a path of length $w_{\text{ext}} \cdot \|(l', m') - (l, m)\|_1 + 2 \cdot w_{\text{open}} - \sum_{o \in \mathcal{Q}(l', m') \setminus \mathcal{Q}(l, m)} \Lambda(o)$ between $D(l, m)$ and $B(l', m')$,

(b) any path between $D(l, m)$ and $B(l', m')$ that does not traverse any alignment arc has length at most $w_{\text{ext}} \cdot \|(l', m') - (l, m)\|_1 + 2 \cdot w_{\text{open}} - \sum_{o \in \mathcal{Q}(l', m') \setminus \mathcal{Q}(l, m)} \Lambda(o)$.

Similar assumptions can be made if $l = l'$ or $m = m'$, in which case we add the gap opening weight at most once and the path ends at a node of type $H$, $V$ or $D$.

Using these two facts, we can prove the statement of the theorem by induction over $x$ and $y$. For $x = 1$ and $y = 1$ there is nothing to show. Consider $x, y \geq 1$ with $x + y > 2$.

Assume in the optimal extended pairwise alignment that aligns $s_x$ and $t_y$ the last alignment arc preceding the one with target $D(x, y)$ has target $D(l, m)$. If such an alignment arc does not exist, we set $l = 0$ and $m = 0$. Using fact (a) and the induction hypothesis, we obtain, by setting $q := \|(x-1, y-1) - (l, m)\|_1$ and using $r$ as defined above,

$$\begin{aligned} D(x, y) &\geq D(l, m) + q \cdot w_{\text{ext}} + r \cdot w_{\text{open}} - \sum_{o \in \mathcal{Q}(x-1, y-1) \setminus \mathcal{Q}(l, m)} \Lambda(o) + \\ &\quad + w(s_x, t_y) - \sum_{o = \mathscr{O}_r(i, y), i \geq x} \Lambda(o) - \sum_{o = \mathscr{O}_b(x, j), j \geq y} \Lambda(o) \qquad (2.16) \\ &= D(l, m) + q \cdot w_{\text{ext}} + r \cdot w_{\text{open}} - \sum_{o \in \hat{\mathscr{O}}_r \cup \hat{\mathscr{O}}_b} \Lambda(o), \end{aligned}$$

where $\hat{\mathscr{O}}_r = \{\mathscr{O}_r(i, j) \mid x \leq i, m < j \leq y\}$ and $\hat{\mathscr{O}}_b = \{\mathscr{O}_b(i, j) \mid y \leq j, l < i \leq x\}$. This value is equal to the score of the optimal extended pairwise alignment of prefixes $s_1 \dots s_x$ and $t_1 \dots t_y$ that aligns $s_x$ and $t_y$.

Now let $p$ be the longest path ending in $D(x, y)$. Notice that the last arc on path $p$ is an alignment arc. Let $D(l, m)$ be the target of the last alignment arc on $p$ preceding

$D(x, y)$. Using fact (b) and the induction hypothesis, we can simply replace "$\geq$" in equation (2.16) by "$\leq$" to obtain analogously

$$D(x, y) \leq D(l, m) + q \cdot w_{\text{ext}} + r \cdot w_{\text{open}} - \sum_{o \in \hat{\mathscr{O}}_r \cup \hat{\mathscr{O}}_b} \Lambda(o),$$

where $q$, $r$, $\hat{\mathscr{O}}_r$ and $\hat{\mathscr{O}}_r$ are as defined above. This value corresponds to the score of the extended pairwise alignment of prefixes $s_1 \dots s_x$ and $t_1 \dots t_y$ that aligns $s_x$ with $t_y$ and $s_l$ with $t_m$. Furthermore, it is based on the optimal extended pairwise alignment of prefixes $s_1 \dots s_l$ and $t_1 \dots t_m$ that aligns $s_l$ with $t_m$. Clearly, the score of this specific alignment is bounded from above by the score of the optimal extended alignment of prefixes $s_1 \dots s_x$ and $t_1 \dots t_y$ that aligns $s_x$ with $t_y$.

□

It remains to show the correctness of assumptions (a) and (b) and their modifications for the case $l' = l$ or $m' = m$ used in the proof. If $l' = l$ and $m' = m$, there is nothing to show. If $l' = l$ or $m' = m$ and the other inequality is strict, we use exclusively horizontal, respectively, exclusively vertical gap arcs, and we therefore do not leave obstacles that we enter. Hence, we do not need to enter the bypass graph and can proceed simply in the DP graph. Correctness of assumptions (a) and (b) still need to be shown for the case where $l' < l$ and $m' < m$. Assumption (a) mainly relies on the existence of a path through the bypass graph that represents a consecutive run of alternate gaps in either string and that is penalized only by newly entered obstacles:

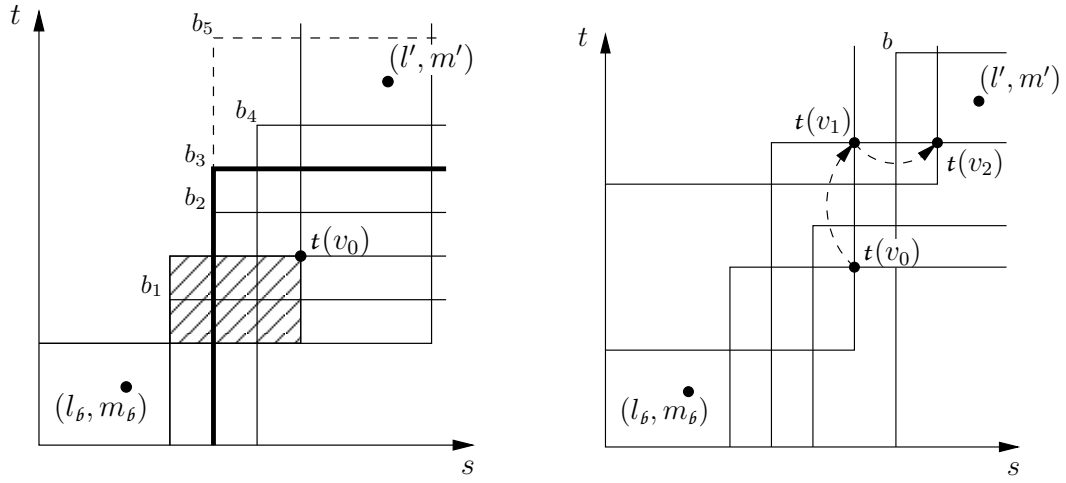
**Lemma 2.1.** *Given a node $v \in \mathcal{V}$ of the BPG, $\mathfrak{b}(v) = (l_\mathfrak{b}, m_\mathfrak{b})$, and a cell $(l', m')$ with $\mathfrak{t}(v) < (l', m')$, there exists a node $v_n \in \mathcal{V}$ and a path $p$ through the BPG from every source node $\mathscr{S}(l_\mathfrak{b}, m_\mathfrak{b})$, $\mathscr{S} \in \{D, V, H, B\}$, to the node $B(\mathfrak{t}(v_n))$ of length*

$$w_{\text{ext}} \cdot ||\mathfrak{t}(v_n) - \mathfrak{b}(v)||_1 + r \cdot w_{\text{open}} - \sum_{o \in \mathcal{Q}(\mathfrak{t}(v_n)) \setminus \mathcal{Q}(\mathfrak{b}(v))} \Lambda(o),$$

*such that $\mathcal{Q}(\mathfrak{t}(v_n)) \setminus \mathcal{Q}(\mathfrak{b}(v)) \subseteq \mathcal{Q}(l', m')$, i.e. obstacles enclosing $\mathfrak{t}(v_n)$ but not $\mathfrak{b}(v)$ also enclose $(l', m')$. Thereby $r = 2$ if $\mathscr{S} = D$, $r = 1$ if $\mathscr{S} \in \{H, V\}$ and $r = 0$ if $\mathscr{S} = B$.*

In the following proof of Lemma 2.1 we use functions $\xi$ and $\psi$ that are defined for an obstacle $o = \mathscr{O}_\kappa(l, m)$ as $\xi(o) = l$ and $\psi(o) = m$. Furthermore, we denote by $A \uplus B$ the union of disjoint sets $A$ and $B$.

*Proof.* We construct a sequence $\langle v_i \rangle_0^n$ of pairs of conflicting obstacles as follows (see Figure 2.7(a)): We select $v_0 = (v_0^b, v_0^r)$ with maximal $\xi(v_0^r)$ and $\psi(v_0^b)$, such that $\mathfrak{b}(v_0) =$




(a) Sequence construction (b) Example sequence

Figure 2.7: Given base $(l_\mathfrak{b}, m_\mathfrak{b})$ and a cell $(l', m')$ as in Lemma 2.1. (a) The initial node $v_0$ in the BPG is determined by the shaded rectangle. Note that $\psi(v_0^b) > \psi(b_1)$. In the example, set $\mathcal{Q}_b^0$ contains blue obstacles $b_2, b_3, b_4$, but not $b_5$, since $b_5$ encloses $(l', m')$. $b_3$ is a leftmost obstacle in $\mathcal{Q}_b^0$ (min. property) and $\psi(b_3) > \psi(b_2)$ (max. property) and therefore $v_1 = (b_3, v_0^r)$. (b) An example sequence $\langle v_i \rangle_0^2$ depicted by the sequence of its tail cells, $(v_0, v_1) \in \mathcal{E}_b$, $(v_1, v_2) \in \mathcal{E}_r$. The sequence terminates at $v_2$ as $b$ encloses $(l', m')$.

$(l_\mathfrak{b}, m_\mathfrak{b})$ and $v_0^b, v_0^r \notin \mathcal{Q}(l', m')$. Let $\mathcal{Q}_b^i$ be the set of blue obstacles that enclose the tail of $v_i$ but neither cell $(l', m')$ nor $(l_\mathfrak{b}, m_\mathfrak{b})$, i.e. $\mathcal{Q}_b^i = \mathcal{Q}_b(\mathfrak{t}(v_i)) \setminus \big(\mathcal{Q}_b(\mathfrak{b}(v_0)) \cup \mathcal{Q}_b(l', m')\big)$. Accordingly, $\mathcal{Q}_r^i = \mathcal{Q}_r(\mathfrak{t}(v_i)) \setminus \big(\mathcal{Q}_r(\mathfrak{b}(v_0)) \cup \mathcal{Q}_r(l', m')\big)$. Then for $i \geq 1$, if $\mathcal{Q}_b^{i-1} \neq \emptyset$, $v_i$ is obtained from $v_{i-1}$ by picking the uppermost among the leftmost blue obstacles in $\mathcal{Q}_b^{i-1}$ while keeping the red obstacle unchanged, i.e. $v_i = (\mathscr{O}_b(g, h), v_{i-1}^r)$, with $\mathscr{O}_b(g, h) \in \mathcal{Q}_b^{i-1}$ such that $\forall g', h'$, $\mathscr{O}_b(g', h') \in \mathcal{Q}_b^{i-1} : g \leq g'$ *(min. property)* and $\forall h'$, $\mathscr{O}_b(g, h') \in \mathcal{Q}_b^{i-1} : h > h'$ *(max. property)*. Similarly, if $\mathcal{Q}_b^{i-1} = \emptyset$ but $\mathcal{Q}_r^{i-1} \neq \emptyset$, we retain the blue obstacle and choose the rightmost among the lowermost red obstacles in $\mathcal{Q}_r^{i-1}$, i.e. we set $v_i = (v_{i-1}^b, \mathscr{O}_r(g, h))$, with $\mathscr{O}_r(g, h) \in \mathcal{Q}_r^{i-1}$ such that $\forall g', h'$, $\mathscr{O}_r(g', h') \in \mathcal{Q}_r^{i-1} : h' \geq h$ and $\forall g'$, $\mathscr{O}_r(g', h) \in \mathcal{Q}_r^{i-1} : g' < g$. The sequence terminates at $v_n$ if $\mathcal{Q}_b^n = \mathcal{Q}_r^n = \emptyset$ (Figure 2.7(b)).

In the following, we show that nodes in the bypass graph representing pairs of conflicting obstacles in $\langle v_i \rangle_0^n$ lie on a path $\hat{p}$ that can be easily extended to a path $p$ having the required properties. It can be easily verified that there exists an arc between nodes corresponding to two consecutive pairs of obstacles in $\langle v_i \rangle_0^n$: the min. and max. properties of our construction of sequence $\langle v_i \rangle_0^n$ ensure $v_i^b \in \mathcal{D}(v_{i-1}^b)$ if $\mathcal{Q}_b^{i-1} \neq \emptyset$, and $v_i^r \in \mathcal{D}(v_{i-1}^r)$ otherwise. Furthermore, the existence of an obstacle $\hat{v}^\kappa \in \mathcal{D}(v_{i-1}^\kappa)$ with $\hat{v}^\kappa < v_i^\kappa$ would be in contradiction to the min. property of $v_i^\kappa$, meaning $v_i^\kappa$ is minimal in $\mathcal{D}(v_{i-1}^\kappa)$ and thus $(v_{i-1}, v_i) \in \mathcal{E}_\kappa$, for all $1 \leq i \leq n$, $\kappa \in \{b, r\}$.

We will argue by induction on the number of arcs $k$, $1 \leq k \leq n$, on a prefix of the path

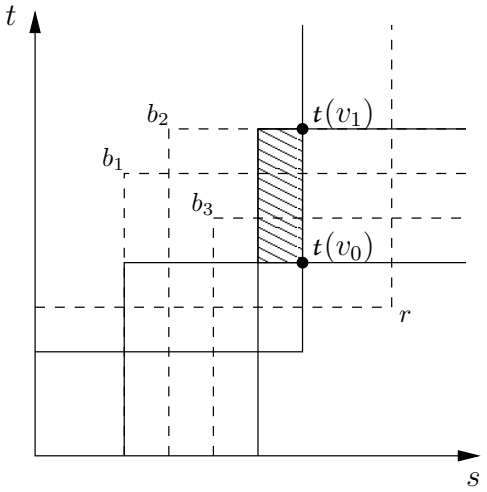


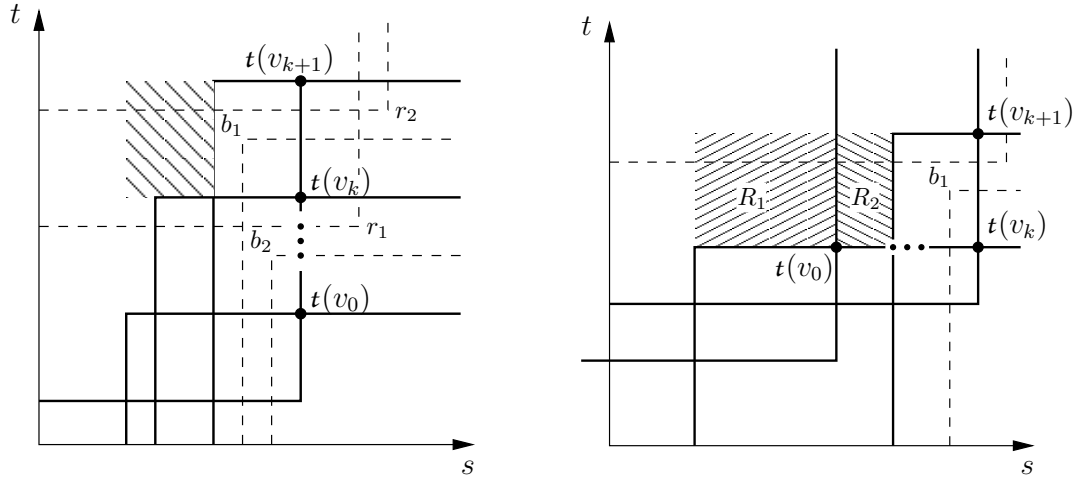

Figure 2.8: Consider $(v_0, v_1) \in \mathcal{E}_b$ lying on a path $\hat{p}$ through the BPG as described in Lemma 2.1. Nodes $v_0$, $v_1$ are represented by their corresponding tails and by obstacles drawn by solid lines. Obstacles in $\mathcal{Q}^+(v_0, v_1)$ originate in the shaded rectangle. The existence of obstacle $b_1$ is in contradiction to the maximality of $\psi(v_0^b)$, $b_2$ is in conflict with the min. property of $\xi(v_1^b)$. According to the definition of an arc in the BPG, $v_1^b$ is minimal in $\mathcal{D}(v_0^b)$, and therefore obstacle $b_3$ cannot exist. It follows that $\mathcal{Q}^+(v_0, v_1) = Q^+(1)$.

induced by sequence $\langle v_i \rangle_0^n$, that

$$\sum_{i=1}^{k} l((v_{i-1}, v_i)) = w_{\text{ext}} \cdot ||t(v_k) - t(v_0)||_1 - \sum_{o \in \mathcal{Q}^-(k)} \Lambda(o) + \sum_{o \in \mathcal{Q}^+(k)} \Lambda(o), \tag{2.17}$$

with $\mathcal{Q}^-(k) = \mathcal{Q}(t(v_k)) \setminus \mathcal{Q}(t(v_0))$ and $\mathcal{Q}^+(k) = \mathcal{Q}(t(v_0)) \setminus (\mathcal{Q}(b(v_0)) \cup \mathcal{Q}(t(v_k)))$. In other words, the length of path $\hat{p}$ going from $v_0$ to $v_k$ accounts for the extension cost of gaps between cells $t(v_0)$ and $t(v_k)$ and is penalized by obstacles enclosing $t(v_k)$ but not $t(v_0)$. Additionally, penalties of obstacles that $\hat{p}$ leaves are compensated if they enclose $t(v_0)$ but not $b(v_0)$. Note that these obstacles are being paid for when traversing an arc connecting a node in cell $b(v_0)$ of the dynamic programming graph with BPG node $v_0$. Also, the weight of this arc incorporates any necessary gap opening costs, depending on the type of its source node. A crucial observation in this context is, that penalties assigned to obstacles that $\hat{p}$ enters along one arc and leaves along a subsequent arc cancel out each other.

For the base case ($k = 1$) it suffices to show that $\mathcal{Q}^+(1) = \mathcal{Q}^+(v_0, v_1)$ (compare equation (2.17) for $k = 1$ with the length of an arc in the BPG, Definition 2.10). Without loss of generality, assume that $(v_0, v_1) \in \mathcal{E}_b$ (see Figure 2.8). Note that for general $(v_{i-1}, v_i) \in \mathcal{E}_b$ every red obstacle enclosing $t(v_{i-1})$ also encloses $t(v_i)$ (e.g. red obstacle $r$ in Figure 2.8) and thus $\mathcal{Q}(t(v_{i-1})) \setminus \mathcal{Q}(t(v_i)) \subseteq \mathscr{O}_b$. For every element $o \in \mathcal{Q}^+(v_0, v_1)$ it holds $o \notin \mathcal{Q}(b(v_0))$ and $o \notin \mathcal{Q}(t(v_1))$ by definition, and thus $\mathcal{Q}^+(v_0, v_1) \subseteq \mathcal{Q}^+(1)$. In order to show that $\mathcal{Q}^+(v_0, v_1) \supseteq \mathcal{Q}^+(1)$, consider an arbitrary element $\mathscr{O}_b(g, h) \in \mathcal{Q}^+(1)$.




(a) $(v_{i-1}, v_i) \in \mathcal{E}_b$, $\forall 1 \leq i \leq k$ (b) $\exists 1 \leq i \leq k$ $(v_{i-1}, v_i) \in \mathcal{E}_r$

Figure 2.9: (a) Obstacles in $\mathcal{Q}^-(k)$ are from $\mathscr{O}_r$ and enclose $t(v_{k+1})$, see obstacle $r_1$. Therefore, $\mathcal{Q}^-(k+1)$ is obtained by simply adding obstacles that enclose $t(v_{k+1})$, but not $t(v_k)$, like obstacle $r_2$. Obstacles in $\mathcal{Q}^+(k+1)$ can be divided into two subsets, depending on whether they enclose $t(v_k)$ (obstacle $b_1$) or not (obstacle $b_2$). The latter one coincides with set $\mathcal{Q}^+(k)$. The first subset is equal to set $\mathcal{Q}^+(v_k, v_{k+1})$, as the min. and max. properties of elements of sequence $\langle v_i \rangle$ imply the shaded rectangle to be empty. (b) Obstacles in $\mathcal{Q}^-(k)$ must not enclose $t(v_{k+1})$ (e.g. obstacle $b_1$) and thus have to be removed from $\mathcal{Q}^-(k) \uplus \mathcal{Q}^-(v_k, v_{k+1})$ to obtain $\mathcal{Q}^+(k+1)$. Note that no blue obstacle originates in rectangles $R_1$ (an arc in $\mathcal{E}_r$ is traversed only if there is no outgoing arc in $\mathcal{E}_b$) or $R_2$ (min. and max. properties of elements of $\langle v_i \rangle$). Therefore, obstacles enclosing $t(v_k)$ but not $t(v_{k+1})$ do not enclose $t(v_0)$, and $\mathcal{Q}^+(k+1) = \mathcal{Q}^+(k)$ follows (obstacles enclosing $t(v_0)$ but not $t(v_k)$ do not enclose $t(v_{k+1})$).

From $\mathscr{O}_b(g,h) \notin \mathcal{Q}(b(v_0))$ and $\mathscr{O}_b(g,h) \in \mathcal{Q}(t(v_0))$ it follows that $g \geq \xi(v_0^b)$. Furthermore, $g = \xi(v_0^b)$ and $h = \psi(v_1^b)$ are contradictory to the max. property of $v_0^b$ and the min. property of $v_1^b$, respectively (see obstacle $b_1$ and $b_2$ in Figure 2.8). At the same time, $\xi(v_0^b) < g < \xi(v_1^b)$ and $\psi(v_0^b) < h < \psi(v_1^b)$ are in contradiction to the minimality of $v_1^b$ in $\mathcal{D}(v_0^b)$ (obstacle $b_3$ in Figure 2.8), from which we conclude that $g \geq \xi(v_1^b)$ and $h \leq \psi(v_1^b)$, and thus $\mathcal{Q}^+(1) \subseteq \mathcal{Q}^+(v_0, v_1)$.

Now assume that equation (2.17) is true for some $k$ with $1 \leq k < n$. Then the path obtained by appending arc $(v_k, v_{k+1})$ has length

$$\begin{aligned} & q_k w_{\text{ext}} - \sum_{o \in \mathcal{Q}^-(k)} \Lambda(o) + \sum_{o \in \mathcal{Q}^+(k)} \Lambda(o) + l(v_k, v_{k+1}) \\ = \; & q_{k+1} w_{\text{ext}} - \sum_{o \in \mathcal{Q}^-(k) \uplus \mathcal{Q}^-(v_k, v_{k+1})} \Lambda(o) + \sum_{o \in \mathcal{Q}^+(k) \uplus \mathcal{Q}^+(v_k, v_{k+1})} \Lambda(o), \end{aligned}$$

where $q_i = ||t(v_i) - t(v_0)||_1$.

Now assume $(v_k, v_{k+1}) \in \mathcal{E}_b$ (for $(v_k, v_{k+1}) \in \mathcal{E}_r$ a symmetric argument applies). Then it

is easy to see, that

$$\mathcal{Q}^-(k+1) = \mathcal{Q}^-(k) \uplus \mathcal{Q}^-(v_k, v_{k+1}) \quad \text{and} \tag{2.18}$$
$$\mathcal{Q}^+(k+1) = \mathcal{Q}^+(k) \uplus \mathcal{Q}^+(v_k, v_{k+1}) \tag{2.19}$$

if $(v_{i-1}, v_i) \in \mathcal{E}_b$, $\forall 1 \leq i \leq k$ (see Figure 2.9(a)), and

$$\mathcal{Q}^-(k+1) = (\mathcal{Q}^-(k) \uplus \mathcal{Q}^-(v_k, v_{k+1})) \setminus \mathcal{Q}^+(v_k, v_{k+1}) \quad \text{and} \tag{2.20}$$
$$\mathcal{Q}^+(k+1) = \mathcal{Q}^+(k) \tag{2.21}$$

otherwise (see Figure 2.9(b)). In both cases, (2.17) follows by induction.

Now let again $q_n = ||\mathfrak{t}(v_n) - \mathfrak{t}(v_0)||_1$ and $\hat{q}_n = ||\mathfrak{t}(v_n) - \mathfrak{b}(v_0)||_1$. Then by extending path $\hat{p}$ by an arc from an appropriate base node $\mathscr{S}(\mathfrak{b}(v_0))$, $\mathscr{S} \in \{D, H, V, B\}$, to BPG node $v_0$, we obtain a path $p$ of desired length

$$\hat{q}_n w_{\text{ext}} + r \cdot w_{\text{open}} - \sum_{o \in \mathcal{Q}(\mathfrak{t}(v_n)) \setminus \mathcal{Q}(\mathfrak{b}(v_0))} \Lambda(o),$$

where the number $r$ of gaps we are opening in cell $\mathfrak{b}(v_0)$ is determined by the type $\mathscr{S}$ of the base node in which path $p$ originates. More precisely, $r = 2$ if $\mathscr{S} = D$, $r = 1$ if $\mathscr{S} \in \{H, V\}$ and $r = 0$ if $\mathscr{S} = B$.

Note that the termination condition of sequence $\langle v_i \rangle_0^n$ implies $\forall o \in \mathcal{Q}(\mathfrak{t}(v_n)) \setminus \mathcal{Q}(\mathfrak{b}(v_0)) : o \in \mathcal{Q}(l', m')$, and therefore the claim of the lemma follows. □

**Lemma 2.2.** *Given strings $s$ and $t$ of length $n_s$ and $n_t$, respectively, consider arbitrary but fixed indices $1 \leq l < l' < n_s$ and $1 \leq m < m' < n_t$. There is a path from $D(l, m)$ to $B(l', m')$ of length*

$$w_{\text{ext}} \cdot ||(l', m') - (l, m)||_1 + 2 \cdot w_{\text{open}} - \sum_{o \in \mathcal{Q}(l', m') \setminus \mathcal{Q}(l, m)} \Lambda(o).$$

*Proof.* Starting from node $D(l, m)$, traversing exclusively gap arcs, we enter the BPG from a node in cell $(l_\mathfrak{b}, m_\mathfrak{b})$, from which we cannot proceed without entering a forbidden obstacle. Cell $(l_\mathfrak{b}, m_\mathfrak{b})$ must be the base of a pair of conflicting obstacles (see Figure 2.3(a)). We thus construct a sequence $\langle v_i \rangle_0^n$ of pairs of conflicting obstacles as described in the proof of Lemma 2.1 to determine the path through the BPG. If we now can find a path from the $B$-node in cell $\mathfrak{t}(v_n)$ to node $B(l', m')$ using exclusively gap arcs that are not entering any forbidden obstacles, the overall path from $D(l, m)$ to $B(l', m')$ has desired length $w_{\text{ext}} \cdot ||(l', m') - (l, m)||_1 + 2 \cdot w_{\text{open}} - \sum_{o \in \mathcal{Q}(l', m') \setminus \mathcal{Q}(l, m)} \Lambda(o)$. Otherwise, we again reach the base of a pair of conflicting forbidden obstacles, and we apply Lemma 2.1 again to bypass forbidden obstacles. □

Finally, we show that we do not overestimate the optimal path length.

**Lemma 2.3.** *Given strings $s$ and $t$ of length $n_s$ and $n_t$, respectively, consider arbitrary but fixed indices $1 \leq l < l' < n_s$ and $1 \leq m < m' < n_t$. Any path from $D(l,m)$ to $B(l',m')$ that uses only gap arcs has length at most*

$$w_{\text{ext}} \cdot ||(l',m') - (l,m)||_1 + 2 \cdot w_{\text{open}} - \sum_{o \in \mathcal{Q}(l',m') \setminus \mathcal{Q}(l,m)} \Lambda(o).$$

*Proof.* For the sake of simplicity, consider an arbitrary path that enters the BPG only once from a node $\mathscr{S}(l_\natural, m_\natural)$ and returns to the original dynamic programming graph at a node $B(t_n)$. Then obstacles in $\mathcal{Q}(l',m') \setminus \mathcal{Q}(l,m)$ can be subdivided into three disjoint groups. Obstacles that enclose $(l_\natural, m_\natural)$, obstacles in $\mathcal{Q}(t_n) \setminus \mathcal{Q}(l_\natural, m_\natural)$, and obstacles that do not enclose $t_n$. Obstacles from first and third group must be entered and thus paid by the sequence of gap arcs leading from $D(l,m)$ to $\mathscr{S}(l_\natural, m_\natural)$ (formulas (2.10)-(2.11) and (2.14)), and from $B(t_n)$ to $B(l',m')$ (formula (2.14)), respectively. The length of an arbitrary path $p'$ from $\mathscr{S}(l_\natural, m_\natural)$ to $B(t_n)$ through the BPG differs from path $p$ induced by sequence $\langle v_i \rangle_0^n$ and constructed in Lemma 2.1 only in two aspects. First, for an arc $(v_k, v_{k+1})$ on path $\hat{p}$ we have to relax (2.19) to $\mathcal{Q}^+(k+1) \supseteq \mathcal{Q}^+(k) \uplus \mathcal{Q}^+(v_k, v_{k+1})$ and equation (2.20) to $\mathcal{Q}^-(k+1) \subseteq \mathcal{Q}^-(k) \setminus \mathcal{Q}^+(v_k, v_{k+1})$. Intuitively, when traversing arc $(v_k, v_{k+1}) \in \mathcal{E}_b$ in Figure 2.9(a), the shaded rectangle may still contain obstacles. And second, $(\mathcal{Q}(t_n) \setminus \mathcal{Q}(l_\natural, m_\natural)) \subseteq \mathcal{Q}(l',m')$ (see termination condition of sequence $\langle v_i \rangle_0^n$) does not necessarily hold. As a consequence, obstacles that contribute to the penalty of path $p$ also contribute to the penalty of $\hat{p}$, and the claim follows. □

## 2.6 Improving the Lagrangian Relaxation Bound

Recall that $(LR_\lambda)$ is the Lagrangian relaxation of $(P'_z)$ relative to the mixed cycle constraints, with nonnegative Lagrangian multipliers $\lambda$. The original ILP formulation for MSA is $(P)$. Furthermore, $v(LR_\lambda)$ is the score of the optimal extended pairwise alignment for given $\lambda \geq \mathbf{0}$.

Since the optimal score $v(LR_\lambda)$ is an upper bound on the optimal score of $(P)$ (the optimal alignment score) for all multiplier vectors $\lambda \in \mathbb{R}^m_+$, $m = |\mathcal{M}|$, we are interested in solving the *Lagrangian dual* of $(P'_z)$ relative to the mixed cycle constraints, namely

$$\min_{\lambda \geq 0} v(LR_\lambda), \tag{LR}$$

to obtain tighter bounds for our branch-and-bound algorithm. If $\lambda^*$ is an optimal solution to $(LR)$, then $v(LR)$ denotes its optimal value $v(LR_{\lambda^*})$.

### 2.6.1 Subgradient Optimization

It is well known [Fis04] that the *Lagrangian function* $f(\lambda) = v(LR_\lambda)$ (in our case, where MSA is a maximization problem) is a convex function of $\lambda$, but it is not differentiable at points, where the optimal solution to $(LR_\lambda)$ is not unique. A commonly used approach to determine near-optimal Lagrangian multipliers efficiently is based on the vector of *subgradients* $g(\lambda) \in \mathbb{R}^m$, associated with a given $\lambda$. The set $\partial f(\lambda^0)$ of all subgradients of $f(\lambda)$ at a point $\lambda^0$ is always nonempty, and it is apparent that the vector $g = (g_M)_{M \in \mathcal{M}}$ of slacks of the mixed cycle constraints

$$g_M(\lambda^0) = |M \cap A_A| - 1 - \sum_{e \in M \cap A_A} \bar{z}_e, \quad M \in \mathcal{M},$$

is contained in $\partial f(\lambda^0)$, where $\bar{z}$ is an optimal solution to $(LR_{\lambda^0})$. The iterative approach proposed by Held and Karp [HK71] generates a sequence $\lambda^0, \lambda^1, \ldots$ of Lagrangian multipliers by taking at iteration $k+1$ a step in the direction opposite to a subgradient of $f(\lambda^k)$, projecting the resulting point onto the nonnegative orthant ($\lambda$ must be nonnegative):

$$\lambda_M^{k+1} = \max\left\{0,\ \lambda_M^k - \theta \cdot \frac{v(LR_{\lambda^k}) - LB}{\|g(\lambda^k)\|^2}\, g_M(\lambda^k)\right\}, \quad M \in \mathcal{M}, \tag{2.22}$$

where $LB$ is a previously computed lower bound on $v(LR)$, and $\theta$ is a step size parameter assuming values in $\{x \in \mathbb{R} \mid 0 < x \leq 2\}$. We obtain $LB$ by applying a heuristic to $(P)$.

More precisely, given an optimal solution $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ to $(LR_\lambda)$ for arbitrary $\lambda \geq \mathbf{0}$, we select in a greedy fashion edges from the set $\{e \in E_A \mid \bar{x}_e = 1\}$ that satisfy conditions *(PwA), (MixCyc),* and *(Trans).*

Concerning the adaption of scalar step size $\theta$, our approach differs from the classical Held-Karp method, which reduces parameter $\theta$ by a factor of 2 whenever there is no upper bound improvement for a certain number of consecutive iterations. If the best and worst upper bounds computed in the last $p$ iterations differ by more than 1%, we suspect that we are "overshooting", and thus we halve the current value of $\theta$. If, in contrast, the two values are within 0.1% from each other, we overestimate $v(LR)$ and therefore increase $\theta$ by a factor of 1.5. Similarly to [CFT99], we experienced a faster convergence to near optimal multipliers using this strategy, compared to the classical approach.

As (2.2) involves exponentially many mixed cycle inequalities that would have to be dualized, formula (2.22) can not be applied in a straightforward way, but we use the relax-and-cut framework outlined below.

### 2.6.2 Relax-and-Cut

In the traditional case of the subgradient method (SM), when the number of dualized constraints is not too large, Beasley [Bea93] reported good practical convergence to $v(LR)$, when arbitrarily setting subgradient components $g_i = 0$ whenever $g_i \geq 0$ and $\lambda_i = 0$. In other words, subgradient entries were arbitrarily set to 0 if the corresponding dualized inequality is not violated by the current solution and has Lagrangian multiplier 0 associated.

Following [Luc92, Luc93], we extend this idea to our context, by setting $g_M = 0$ for all $M \in \mathcal{M}$ with $\lambda_M = 0$ whose corresponding mixed cycle inequalities are not violated by the current Lagrangian solution. These multipliers would remain zero valued at the end of the current iteration and thus would not directly contribute to $v(LR_\lambda)$. We call the corresponding constraints *inactive inequalities.* Conversely, we call inequalities, whose associated multiplier may directly contribute to the Lagrangian objective function, *active inequalities.* These are the constraints (2.2) that are violated by the Lagrangian solution and those inequalities that have nonzero Lagrangian multipliers associated with them.

As a consequence, only active inequalities will be used in (2.22) to determine the step size. The typically huge number of mixed cycle inequalities with strictly positive slack would result in an extremely small step size. Lagrangian multipliers would remain nearly unchanged throughout the iterations and thus imply a high numerical risk for the convergence of SM.

This dynamic scheme, where the pool of active inequalities may continuously change, heavily relies on the ability to identify, at every iteration of SM, inequalities that are violated by the Lagrangian solution. In our case, the separation problem reduces to a shortest path problem with nonnegative edges weights and thus can be solved efficiently using Dijkstra's algorithm [CLRS01].

In the first iteration, we set all Lagrangian multipliers to 0 and initialize a *constraint pool* to be empty. At any given SM iteration, we identify mixed cycle inequalities that are most violated by the average of the last $h$ solutions, and add these constraints to the pool. More precisely, the most violated mixed cycle that contains a positioning arc $(u, v)$ is obtained by a shortest path computation from node $v$ to node $u$, where the weight of an alignment edge $u_l^i v_m^j$ is defined as $1 - \frac{x}{h}$ if $s_l^i$ and $s_m^j$ were aligned in $x$ out of the last $h$ extended pairwise alignments between strings $s^i$ and $s^j$. Positioning arcs are assigned weight 0. This conservative strategy prevents the pool of active constraints from growing too rapidly. Notice that active inequalities may become inactive again and thus have to leave the constraint pool.

A general flow chart of the overall branch and bound algorithm based on the proposed Lagrangian relaxation approach is given in Figure 2.10.

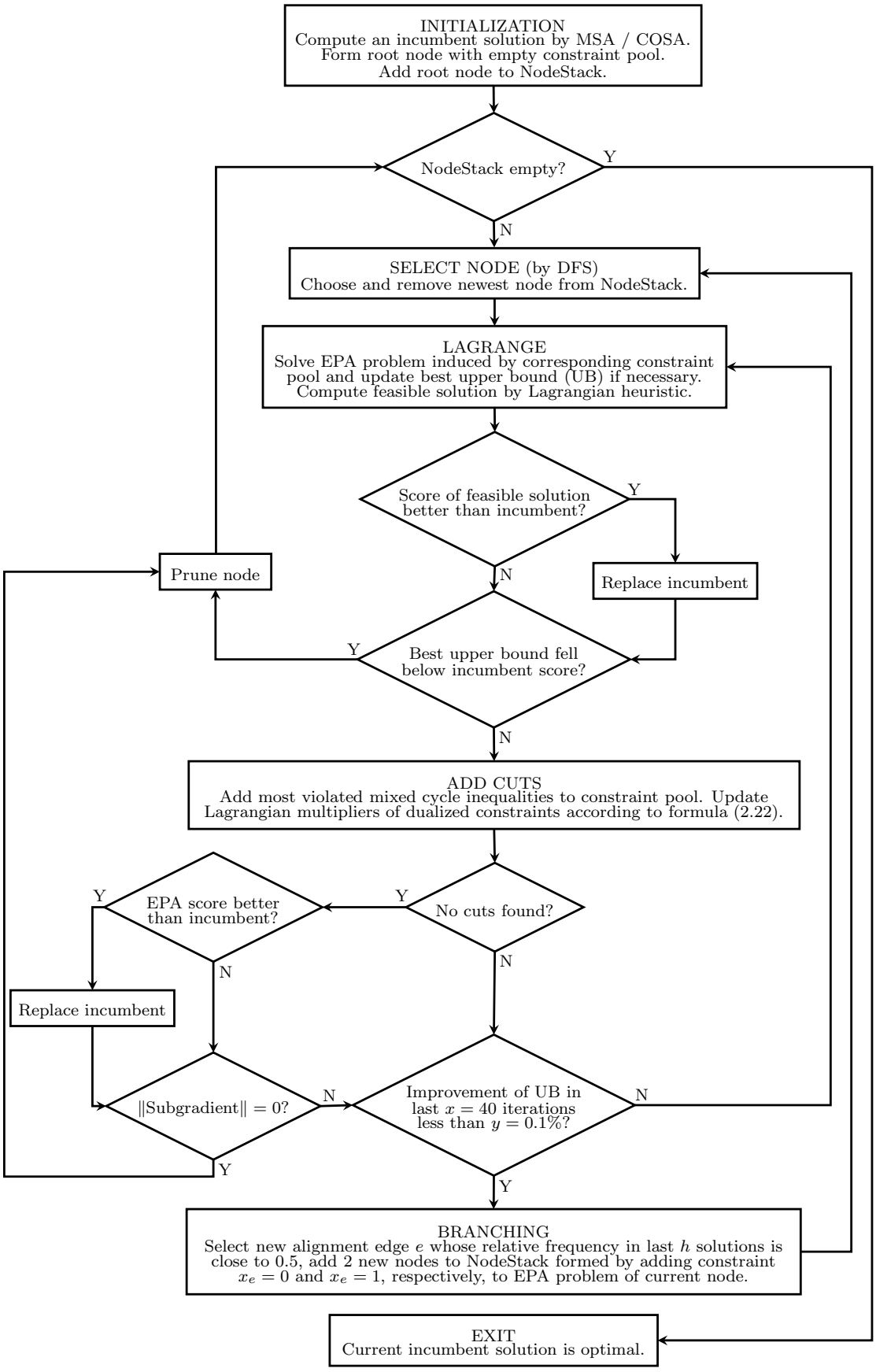



Figure 2.10: Flow chart of the proposed B&B algorithm

## 2.7 Experiments

We have implemented our Lagrangian approach in C++ using the LEDA library [MN95]. To solve the extended pairwise alignment problem, an $A^*$ search algorithm determines the longest path through the dynamic programming graph (augmented with the bypass graph). The upper bounds were improved by a relax-and-cut modification to the subgradient method and used in a branch-and-bound (B&B) framework to prune the search space in order to find an optimal multiple alignment. In each node of the branch-and-bound tree, a lower bound is computed by selecting, in a greedy fashion, alignment edges realized in the current Lagrangian solution that satisfy conditions *(PwA), (MixCyc),* and *(Trans)*. We set $w_{\text{ext}} = -4$ and $w_{\text{open}} = -6$, i.e. the gap arcs were assigned a weight that was computed as $-4q-6$, where $q$ is the number of characters in the corresponding gap. The weights for the alignment edges in $E_A$ were obtained by the BLOSUM62 amino acid substitution matrix.

We tested our implementation , which we call LASA (LAgrangian Sequence Alignment), on a set of instances of the BAliBASE library [TPP99], including reference alignments from last inception, version 3. The benchmark alignments from reference 1 (R1) contain 4 to 6 sequences and are subdivided into three groups of different length (*short, medium, long*). They are further categorized into three subgroups by the degree of similarity between the sequences (group *V1*: identity $< 25\%$, group *V2*: identity $20-40\%$, group *V3*: identity $> 35\%$). In BAliBASE 3 two datasets are prepared: The *homologous region* set, which is similar to BAliBASE 2, and the *full length* set, where sequences contain non-homologous residues.

### 2.7.1 Algorithmic Issues

The purpose of the experiments presented in this subsection is to evaluate various decisions made in the design of our algorithm.

First, we give computational evidence for the effectiveness of our novel approach to select violated inequalities to be added to our constraint pool. Considering the average of the last $h$ solutions to the extended pairwise alignment problem instead of taking into account only the current solution ($h = 1$) dramatically reduces the number of iterations (see Table 2.1). Only short sequences of high identity (*short, V3*) could be solved for $h = 1$. Moreover, this table shows that the extended pairwise alignment problems are solved at least twice as fast when applying the $A^*$ approach, instead of a standard dynamic programming algorithm.

The columns in Table 2.1 have the following meaning:

**Instance:** Name of the instance in BAliBASE, along with an indication $(k, n)$ of the number of sequences $k$ and the overall number of characters $n$.

**h = $*$ :** The number of solutions that were considered in an average Lagrangian solution.

Table 2.1: We give the number of iterations needed by our approach for different numbers $h$ of solutions that were considered in an average Lagrangian solution. The default is $h = 10$. The last column gives the time spent in the root node of the B&B tree if the $A^*$ search is replaced by a simple dynamic programming algorithm.

| | $h = 1$ | $h = 2$ | $h = 20$ | $h = 30$ | LASA (A*, $h = 10$) | | DynProg, $h = 10$ |
|---|---|---|---|---|---|---|---|
| Instance | #Iter | #Iter | #Iter | #Iter | #Iter | Time | Time |
| 1aho (5/320) | 748,470 | 2,496 | 1,194 | 1,283 | 1,089 | 10 | 22 |
| 1csp (5/339) | 17 | 14 | 19 | 19 | 17 | <1 | <1 |
| 1dox (4/374) | 80,001 | 271 | 211 | 207 | 253 | 1 | 5 |
| 1fkj (5/517) | 316,072 | 849 | 707 | 676 | 348 | 9 | 25 |
| 1fmb (4/400) | 1,372 | 14 | 14 | 14 | 13 | <1 | <1 |
| 1krn (5/390) | 191,281 | 634 | 148 | 155 | 104 | 1 | 8 |
| 1plc (5/470) | 232,591 | 489 | 642 | 513 | 218 | 6 | 14 |
| 2fxb (5/287) | 16,425 | 15 | 11 | 11 | 11 | <1 | <1 |
| 2mhr (5/572) | 60,005 | 93 | 116 | 177 | 65 | 3 | 8 |
| 9rnt (5/499) | 54 | 49 | 40 | 40 | 39 | 1 | 3 |

**LASA:** Default version of LASA, i.e. $h = 10$ and using the $A^*$ approach.

**DynProg:** LASA based on a simple dynamic programming scheme, instead of A* search.

**#Iter:** Number of SM iterations needed by a specific variant of LASA.

**Time:** Total running time in seconds needed by a specific variant of LASA.

We have developed an efficient algorithm for the extended pairwise alignment problem in three steps. First, we presented in Section 2.5 a direct dynamic programming approach, to which we refer as *simple algorithm* in the following. Then, we reduced the complexity of the dynamic programming (DP) graph by inserting arcs only between dominating bases and dominated targets. We call the algorithm that is based on the resulting DP graph *improved algorithm* (see Section 2.5.1). Finally, we augmented the original DP graph with a bypass graph, which is correlated to the transitive reduction of the induced subgraph on the set of newly added arcs (see Section 2.5.2). We distinguish the latter algorithm from the previous by explicitly mentioning whether the DP graph is based on a "transitive reduction".

In doing so, we were able to reduce the running time for finding an optimal extended alignment between strings $s$ and $t$ with $\|s\| = n$ and $\|t\| = m$ from $\mathcal{O}\left(n^2m^2\left|\mathscr{O}\right|\right)$ for the simple algorithm, over $\mathcal{O}\big(nm + \left|\mathscr{O}\right|^4\big)$ for the improved algorithm without "transitive reduction", to $\mathcal{O}\big(nm + \left|\mathscr{O}\right|^3\big)$ for the algorithm based on the bypass graph. Notice

however, that the implicit bound $|\mathscr{O}| \in \mathcal{O}(nm)$ foils this theoretic improvement. Thus, we tried to assess the practical performance of the simple algorithm and both versions of the improved algorithm (with and without "transitive reduction") by considering the size of the underlying graph structure after the last iteration in the root node of the branch-and-bound tree. Table 2.2 indicates that for a BPG $G = (\mathcal{V}, \mathcal{E})$, $\mathcal{O}\big(|\mathscr{O}|^2\big)$ and $\mathcal{O}\big(|\mathscr{O}|^3\big)$ are rather pessimistic estimates for $|\mathcal{V}|$, respectively $|\mathcal{E}|$, and therefore we expect the running time of the simple algorithm to be significantly larger than the running time of the improved algorithm using the bypass graph. Moreover, the "transitive reduction" obtained by introducing the bypass graph reduces the number of additional arcs considerably.

In the experiments that follow, we compare our implementation (LASA) with various existing methods for the multiple alignment problem. The purpose of these experiments is twofold. On the one hand, we want to evaluate the complexity of instances our current implementation is able to solve in reasonable time. On the other hand, we want to assess the quality of alignments that are *optimal* in our model of the multiple sequence alignment problem. For the latter we additionally used reference alignments from three different sets: SABmark [WLW05], PREFAB [Edg04] and artificially created alignments using Rose [SEM98]. We used the original benchmarking measures proposed by its respective database. A score between 0 and 1 indicates the degree of accordance with the reference alignment.

Table 2.2: For the first four benchmark alignments of each subgroup of short and medium sized instances we compare the size of the underlying graphs of the simple respectively improved algorithm (with and without "transitive reduction"). The figures show a snapshot after the last iteration in the root node of the branch-and-bound tree. We give the average number of obstacles between each pair of strings (#Obst), the average number of nodes in the bypass graph (#BPG-Nodes), the average number of arcs of the BPG (#BPG-Arcs), including arcs connecting the BPG to the original DP graph, and the average number of additional arcs needed when no BPG is used (#Arcs).

| Instance | #Obst | #BPG-Nodes | #BPG-Arcs | #Arcs |
|---|---|---|---|---|
| *Reference 1 Short, V3* | | | | |
| 1aho (5/320) | 32 | 7 | 42 | 205 |
| 1csp (5/339) | 6 | 0 | 0 | 0 |
| 1dox (4/374) | 18 | 3 | 18 | 56 |
| 1fkj (5/517) | 29 | 6 | 35 | 137 |
| *Reference 1 Short, V2* | | | | |
| 1aab (4/291) | 18 | 3 | 18 | 51 |
| 1csy (5/510) | 57 | 11 | 58 | 333 |
| 1fjlA (6/398) | 13 | 2 | 13 | 33 |
| 1hfh (5/606) | 61 | 17 | 91 | 760 |
| *Reference 1 Short, V1* | | | | |
| 1aboA (5/297) | 63 | 40 | 236 | 4047 |
| 1tvxA (4/242) | 82 | 61 | 373 | 8979 |
| 1idy (5/269) | 51 | 21 | 120 | 1314 |
| 1r69 (4/277) | 74 | 29 | 165 | 1934 |
| *Reference 1 Medium, V3* | | | | |
| 1amk (5/1241) | 17 | 1 | 7 | 9 |
| 1ar5A (4/794) | 39 | 10 | 56 | 258 |
| 1ezm (5/1515) | 24 | 5 | 26 | 83 |
| 1led (4/947) | 65 | 13 | 72 | 468 |
| *Reference 1 Medium, V2* | | | | |
| 1ad2 (4/828) | 51 | 11 | 61 | 360 |
| 1aym3 (4/932) | 54 | 21 | 122 | 1286 |
| 1gdoA (4/988) | 104 | 22 | 121 | 1168 |
| 1ldg (4/1240) | 67 | 14 | 76 | 491 |

### 2.7.2 Quantitative Evaluation

We compared the performance of our implementation with the non-heuristic methods MSA [LAK89] (its improved version [GKS95a]) and COSA [ACLR06]. The multiple sequence alignment program MSA is based on dynamic programming and uses the so called *quasi-affine gap cost* model, a simplification of the (truly) affine gap cost model (used by our approach). The branch-and-cut algorithm COSA is based on the same ILP formulation as introduced in Section 2.2 and uses CPLEX [ILO06] as LP-solver.

We ran the experiments on a system with a 2,39 GHz AMD Opteron Processor with 8 GB of RAM. Any run that exceeded a CPU time limit of 13 hours was considered unsuccessful, indicated by "−" in Table 2.3.

Tables 2.3-2.5 report our results on short, medium sized and long instances from the BAliBASE library, reference 1. The columns have the following meaning:

**Instance:** Name of the instance in BAliBASE, along with an indication $(k, n)$ of the number of sequences $k$ and the overall number of characters $n$.

**Heur:** Value of the initial feasible solution found by COSA or MSA.

**PUB:** An upper bound on the optimal score obtained by summing over all optimal pairwise scores.

**Root:** Value of the Lagrangian upper bound at the root node of the branch-and-bound tree.

**Opt:** The score of an optimal solution.

**#Nodes:** Number of branch-and-bound subproblems solved.

**#Iter:** Total number of iterations needed by the subgradient optimization method.

**Time:** Total running time needed to find an optimal solution.

Although MSA reduces the complexity of the problem by incorporating quasi-affine gap costs into the multiple alignment, it could hardly solve instances with a moderate degree of similarity. In contrast, our implementation LASA outperforms the CPLEX based approach COSA, which is, to the best of our knowledge, the only existing method besides LASA that is able to optimize the sum of pairs score with truly affine gap costs of a multiple alignment. COSA was not able to solve any of the medium sized or long benchmark alignments, while LASA found the optimal solution within minutes. This is mainly because the LPs are quite complicated to solve. In their experiments, Althaus *et al.* [ACLR06] noticed that it is more efficient to solve the LPs using the barrier algorithm "from scratch" than reoptimizing using the *dual simplex* after adding violated inequalities to the LP. This indicates the degree of difficulty of the involved LPs. Moreover, one instance crashed as CPLEX was not able to solve an LP (see Table 2.3).

The running time of LASA and COSA strongly depends on tight initial lower bounds. For example (see Table 2.5), it took LASA about 13 hours to find an optimal solution for the long instance 3pmg when the weaker bound obtained by the heuristic method MSA was used. In contrast, an optimal multiple alignment could be found in roughly one hour, if the score of an optimal solution was known in advance.

Table 2.3: Results on short instances from BAliBASE reference 1, subdivided according to their degree of similarity intro groups *V1*, *V2* and *V3*. The running time is given in the format hh:mm:ss, mm:ss or otherwise simply in seconds. A "−" indicates that no solution was found within 13 hours of CPU time.

| | | | | | LASA | | | COSA | MSA |
|---|---|---|---|---|---|---|---|---|---|
| Instance | Heur | PUB | Root | Opt | #Nodes | #Iter | Time | Time | Time |
| *Reference 1 Short, V3* | | | | | | | | | |
| 1aho (5/320) | 877 | 987 | 884 | 881 | 7 | 1,089 | <1 | 01:29 | - |
| 1csp (5/339) | 1,457 | 1,473 | 1,457 | 1,457 | 1 | 17 | <1 | 1 | <1 |
| 1dox (4/374) | 749 | 782 | 751 | 750 | 3 | 253 | 3 | 30 | <1 |
| 1fkj (5/517) | 1,578 | 1,675 | 1,585 | 1,578 | 3 | 348 | 13 | 6:04 | - |
| 1fmb (4/400) | 1,333 | 1,353 | 1,333 | 1,333 | 1 | 13 | <1 | 2 | <1 |
| 1krn (5/390) | 1,523 | 1,558 | 1,523 | 1,523 | 1 | 104 | 1 | 6 | 6 |
| 1plc (5/470) | 1,736 | 1,824 | 1,736 | 1,736 | 1 | 218 | 6 | 04:24 | 20:14 |
| 2fxb (5/287) | 1,341 | 1,352 | 1,341 | 1,341 | 1 | 11 | < | < 1 | <1 |
| 2mhr (5/572) | 2,364 | 2,406 | 2,364 | 2,364 | 1 | 65 | 3 | 2 | 17 |
| 9rnt (5/499) | 2,550 | 2,573 | 2,550 | 2,550 | 1 | 39 | 1 | 4 | <1 |
| *Reference 1 Short, V2* | | | | | | | | | |
| 1aab (4/291) | 231 | 257 | 231 | 231 | 1 | 100 | <1 | 4 | < 1 |
| 1csy (5/510) | 649 | 769 | 649 | 649 | 1 | 393 | 17 | 03:01 | - |
| 1fjlA (6/398) | 674 | 731 | 676 | 674 | 5 | 561 | 12 | 34 | - |
| 1hfh (5/606) | 903 | 1,067 | 911 | 903 | 3 | 411 | 33 | - | - |
| 1hpi (4/293) | 386 | 439 | 386 | 386 | 1 | 298 | 4 | 53 | 7 |
| 1pfc (5/560) | 994 | 1,139 | 1,004 | 994 | 11 | 1,387 | 01:48 | 37:46 | - |
| 1tgxA (4/239) | 247 | 317 | 247 | 247 | 1 | 566 | 9 | 53 | - |
| 1ycc (4/426) | 117 | 309 | 202 | 200 | 7 | 1,865 | 02:19 | - | - |
| 3cyr (4/414) | 515 | 615 | 522 | 515 | 7 | 983 | 38 | -* | 45 |
| *Reference 1 Short, V1* | | | | | | | | | |
| 1aboA (5/297) | −685 | −476 | −604 | −676 | 3,497 | 417,260 | 11:04:02 | - | - |
| 1aboA (5/297) | −676 | −476 | −604 | −676 | 2953 | 349792 | 09:13:49 | - | - |
| 1tvxA (4/242) | −409 | −260 | −358 | −405 | 777 | 122,785 | 01:59:44 | - | - |
| 1idy (5/269) | −420 | −273 | −356 | −414 | 4,193 | 678,592 | 12:00:48 | - | - |
| 1idy (5/269) | −414 | −273 | −352 | −414 | 3529 | 594746 | 10:27:30 | - | - |
| 1r69 (4/277) | −326 | −207 | −289 | −326 | 253 | 54,668 | 58:40 | - | - |
| 1ubi (4/327) | −372 | −246 | −330 | −372 | 215 | 43,620 | 01:12:57 | - | - |
| 1wit (5/484) | −198 | −25 | −186 | −197 | 15 | 4,221 | 07:42 | - | - |
| 2trx (4/362) | −182 | −88 | −178 | −182 | 5 | 2,186 | 03:04 | - | - |

*With the COSA-code, the instance 3cyr crashed after about one hour of computation time as the LP-solver was not able to solve the underlying LP.

Table 2.4: Results on medium sized instances from BAliBASE reference 1. The running time is given in the format hh:mm:ss, mm:ss or otherwise simply in seconds. COSA and MSA were not able to solve any of these benchmark alignments. Results on group *V1* are omitted, since LASA was not able to solve these instances in the allowed time frame.

| Instance | Heur | PUB | Root | Opt | #Nodes | #Iter | Time |
|---|---|---|---|---|---|---|---|
| *Reference 1 Medium, V3* | | | | | | | |
| 1amk (5/1241) | 5,668 | 5,728 | 5,669 | 5,669 | 1 | 60 | 8 |
| 1ar5A (4/794) | 2,303 | 2,357 | 2,304 | 2,303 | 3 | 262 | 20 |
| 1ezm (5/1515) | 8,378 | 8,466 | 8,378 | 8,378 | 1 | 105 | 23 |
| 1led (4/947) | 2,150 | 2,282 | 2,158 | 2,150 | 33 | 1,435 | 03:54 |
| 1ppn (5/1083) | 4,718 | 4,811 | 4,729 | 4,724 | 23 | 925 | 03:10 |
| 1pysA (4/1005) | 2,730 | 2,796 | 2,732 | 2,730 | 3 | 223 | 28 |
| 1thm (4/1097) | 3,466 | 3,516 | 3,468 | 3,468 | 3 | 233 | 30 |
| 1tis (5/1413) | 5,854 | 5,999 | 5,874 | 5,856 | 83 | 2,993 | 18:31 |
| 1zin (4/852) | 2,357 | 2,411 | 2,361 | 2,357 | 13 | 625 | 01:03 |
| 5ptp (5/1162) | 4,190 | 4,329 | 4,233 | 4,205 | 193 | 8,337 | 35:48 |
| *Reference 1 Medium, V2* | | | | | | | |
| 1ad2 (4/828) | 1,195 | 1,270 | 1,197 | 1,195 | 7 | 419 | 42 |
| 1aym3 (4/932) | 1,544 | 1,664 | 1,551 | 1,544 | 17 | 1,060 | 02:37 |
| 1gdoA (4/988) | 980 | 1,201 | 1,003 | 984 | 459 | 31,291 | 02:38:36 |
| 1ldg (4/1240) | 1,526 | 1,640 | 1,539 | 1,526 | 41 | 2,160 | 08:32 |
| 1mrj (4/1025) | 1,461 | 1,608 | 1,473 | 1,464 | 27 | 1,681 | 05:29 |
| 1pgtA (4/828) | 683 | 808 | 691 | 690 | 9 | 926 | 02:05 |
| 1pii (4/1006) | 1,099 | 1,256 | 1,103 | 1,100 | 23 | 1,320 | 04:54 |
| 1ton (5/1173) | 1,550 | 1,898 | 1,609 | 1,554 | 807 | 44,148 | 05:32:47 |

Table 2.5: Results on long instances from BAliBASE reference 1. The running time is given in the format hh:mm:ss. Only three instances could be solved by LASA. MSA and COSA were not able to solve any of these benchmark alignments. Instance 3pmg was solved once with an initial lower bound obtained by MSA (7363) and once with the optimal value (7418) computed by LASA itself.

| Instance | Heur | PUB | Root | Opt | #Nodes | #Iter | Time |
|---|---|---|---|---|---|---|---|
| 1ad3 (4/1746) | 5355 | 5424 | | 5358 | 21 | 734 | 00:04:25 |
| actin (5/1924) | 8018 | 8178 | 8039 | 8022 | 45 | 2138 | 00:19:41 |
| 3pmg (4/2224) | 7363 | 7602 | 7460 | 7418 | 1397 | 53350 | 12:50:50 |
| 3pmg (4/2224) | 7418 | 7602 | 7448 | 7418 | 119 | 4789 | 01:08:37 |

### 2.7.3 Qualitative Evaluation

In terms of alignment quality, we compared our implementation LASA with the heuristic methods T-COFFEE [NHH00], CLUSTALW [THG94a], MAFFT [KiKTM05], MUSCLE [Edg04], DIALIGN [Mor04] and POA [LGS02].

Our primary goal was to develop a new non-heuristic algorithm that allows to solve instances to optimality, that were too complex for previous methods to obtain optimal solutions. Nevertheless, we evaluated the quality of the alignments produced by LASA. Our approach ranks among the best programs implemented so far (see Table 2.6 and Table 2.7 ). The quality could be probably improved by a more careful choice of the objective function. In our current implementation we use a fixed objective for all instances, independent of their level of identity.

Table 2.6: Average score of the alignments computed by different programs. Only instances that have been solved by LASA in less than 2 hours were considered. In BAliBASE 3.0, full length sequences (full) and instances from the homologous region set (hom) are distinguished.

| Group | LASA | T-COFFEE | CLUSTALW | MAFFT | MUSCLE |
|---|---|---|---|---|---|
| *BAliBASE 2.0* | | | | | |
| Short V1 | 0.969 | 0.968 | 0.984 | 0.988 | 0.970 |
| Short V2 | 0.865 | 0.819 | 0.936 | 0.948 | 0.811 |
| Short V3 | 0.512 | 0.340 | 0.562 | 0.882 | 0.537 |
| Medium V1 | 0.944 | 0.952 | 0.943 | 0.969 | 0.969 |
| Medium V2 | 0.933 | 0.886 | 0.911 | 0.901 | 0.895 |
| Long V1 | 0.960 | 0.941 | 0.933 | 0.976 | 0.982 |
| *BAliBASE 3.0* | | | | | |
| Ref1 V1 full | 0.942 | 0.966 | 0.935 | 0.939 | 0.952 |
| Ref1 V1 hom | 0.795 | 0.672 | 0.764 | 0.819 | 0.780 |
| Ref1 V2 full | 0.918 | 0.900 | 0.918 | 0.919 | 0.905 |
| Ref1 V2 hom | 0.894 | 0.876 | 0.895 | 0.882 | 0.888 |

Table 2.7: Rows show the average developer ($f_D$) score and modeler ($f_M$) score for the "Superfamily"($\leq$ 50% identity) and "Twilight Zone" ($\leq$ 25% identity) sets in the SABmark database, respectively the quality ($Q$) score and total column ($TC$) score for PREFAB instances and for reference alignments created by Rose, achieved by each aligner. The latter are grouped according to their average evolutionary distance. The number of sequences in each set is given in parenthesis.

| Group/Score | LASA | T-COFFEE | CLUSTALW | MAFFT | MUSCLE | DIALIGN | POA |
|---|---|---|---|---|---|---|---|
| *SABmark (405)* | | | | | | | |
| Superfam. $f_D$ | 79.56 | 80.50 | 81.04 | 82.32 | 80.80 | 75.50 | 69.41 |
| Superfam. $f_M$ | 61.11 | 62.50 | 62.39 | 62.88 | 62.28 | 60.57 | 63.69 |
| Twilight $f_D$ | 47.82 | 46.84 | 50.3 | 48.72 | 47.91 | 40.64 | 29.74 |
| Twilight $f_M$ | 33.43 | 33.13 | 34.25 | 34.26 | 33.43 | 31.23 | 34.66 |
| *PREFAB (161)* | | | | | | | |
| $Q$ | 0.71 | 0.69 | 0.69 | 0.69 | 0.71 | 0.63 | 0.57 |
| $TC$ | 0.71 | 0.69 | 0.69 | 0.69 | 0.71 | 0.63 | 0.57 |
| *Rose (264)* | | | | | | | |
| Dist100 $Q$ | 0.91 | 0.90 | 0.88 | 0.91 | 0.92 | 0.87 | 0.79 |
| Dist100 $TC$ | 0.86 | 0.86 | 0.82 | 0.88 | 0.88 | 0.81 | 0.67 |
| Dist150 $Q$ | 0.78 | 0.76 | 0.80 | 0.81 | 0.83 | 0.72 | 0.55 |
| Dist150 $TC$ | 0.70 | 0.67 | 0.72 | 0.71 | 0.75 | 0.59 | 0.38 |
| Dist200 $Q$ | 0.67 | 0.60 | 0.62 | 0.63 | 0.67 | 0.51 | 0.34 |
| Dist200 $TC$ | 0.50 | 0.44 | 0.46 | 0.50 | 0.54 | 0.32 | 0.18 |
| Dist250 $Q$ | 0.58 | 0.46 | 0.55 | 0.49 | 0.53 | 0.39 | 0.29 |
| Dist250 $TC$ | 0.37 | 0.23 | 0.36 | 0.28 | 0.35 | 0.18 | 0.11 |

## 2.8 Conclusion

We have presented a Lagrangian relaxation approach for the multiple sequence alignment problem with the sum of pairs scoring scheme, that allows us to obtain strong bounds on the optimal solution by solving a generalization of the pairwise alignment problem. We are able to deal with truly affine gap costs, which is, to the best of our knowledge, possible so far only with COSA [ACLR06]. Due to a huge expense in computational resources, quasi-affine gap costs are usually computed instead, as it is the case for the alignment tool MSA. We strengthen the obtained Lagrangian relaxation bound by a slightly modified relax-and-cut variant of the subgradient optimization method. Experiments have shown, that the strategy to dualize constraints that are most violated by the convex combination of the current Lagrangian solution and a small number of preceding solutions increases the convergence rate dramatically. By utilizing the resulting bounds on the optimal ILP value in a branch-and-bound manner we achieve running times that significantly outperform state-of-the-art exact or almost exact methods like COSA and MSA. Furthermore, we are able to solve instances with a degree of difficulty that none of the previous exact methods was able to solve.

Compared to the branch-and-cut approach COSA, we obviously profit from the efficiency, with which we are able to compute a longest path in an acyclic graph that avoids obstacles. In contrast, the running time of COSA is dominated by solving a general linear program with an exponential number of variables and constraints. Even for moderately difficult instances the LPs become too large to be solvable in reasonable time.

Concerning the quality of the alignments, our results rank among the best alignments computed so far. Nevertheless, we believe that a more careful abstraction of the underlying biological problem, in particular concerning the objective function, can improve the quality of the alignments considerably.

An important issue concerning a further improvement of the proposed algorithm is to evaluate the applicability of recent developments (see [Lem01]) in the field of nondifferentiable optimization methods to the solution of the Lagrangian dual problem. A more sophisticated Lagrangian heuristic for computing lower bounds in the nodes of the branch-and-bound tree will be necessary to be able to solve instances of larger size and a higher degree of difficulty.

Chapter 3

# Interval Constraint Coloring

## 3.1 Introduction

A challenging and important problem in biochemistry is to determine protein-protein and protein-ligand interactions. A decisive role in this interaction plays the tertiary structure of a protein, i.e. the spatial arrangement, which is indispensable for its function. There are various laboratory-driven approaches, each with advantages and drawbacks. One method that provides information about the tertiary structure in terms of solvent accessibility is the so-called *hydrogen-deuterium exchange*, abbreviated by HDX. This is a chemical reaction where a hydrogen atom of the protein is replaced by a deuterium atom, or vice versa. To this end, the protein solution is diluted with $D_2O$. Intuitively, the exchange process happens at a higher rate at amino acids, or residues, that are more exposed to the solvent. Put differently, the exchange rates for residues at the outside of the complex are higher than inside, with the exception of *proline*, which does not possess an amide hydrogen atom. Note that though deuterium is heavier than hydrogen, they are almost identical from a chemical point of view. Hence, the exchange rate may be monitored by mass spectroscopy while the tertiary structure remains unaffected by the process. Comparing the solvent accessibility of the isolated protein and the protein-protein (or protein-ligand) complex reveals the interaction interface, that can be feeded into computer docking tools subsequently.

However, this method does not deliver that fine grained information that would allow the exchange rate for each residue to be determined directly. Rather, we get aggregate information for several overlapping fragments covering the whole protein. For example, the experimental data only tell us how many residues of such a fragment react at low, medium, and high exchange rate, respectively. Moreover, we know the exact location and size of each fragment in the protein. It remains to find a valid assignment of exchange rates to all residues that matches the experimentally found bulk information. If the solution is not unique, we want to enumerate all valid assignments or a representative subset thereof as a basis for further chemical considerations.

This chapter is organized as follows. After giving some biochemical background to the problem and introducing the problem formally in the next subsections, we present an ILP approach in Section 3.2. A combinatorial approach for the 2-color case is described in Section 3.3, which is used as a subroutine to approximate the optimal solution to the general case of an arbitrary but fixed number of colors in Section 3.4. The Lagrangian relaxation approach to enumerate all optimal colorings is introduced in Section 3.5. We compare the performance of the different methods in experiments on real-world instances in Section 3.6. An algorithm that rounds a fractional solution to obtain a coloring that satisfies all the requirements within an error of $\pm 1$ is presented in Section 3.7. After introducing a Quasi-PTAS for the maximization variant of the problem in Section 3.8, we give hardness results in Section 3.9. Finally, in Section 3.10, we conclude our work with a evaluation of our results together with a short outlook on possible future work.

### 3.1.1 Biochemical Motivation

Determination of protein-protein interactions is best accomplished by X-ray crystal diffraction and NMR [Zui02] since both methods provide the highest resolution of the sites of interaction. On the downside, these methods require large (milligram) quantities of protein. Other techniques to determine protein-protein interactions rely on chemical or photo-induced reactions with MS analysis [LC02, KHJ$^+$06] and reveal functional groups that are exposed to the solvent. These methods also suffer from physical experimental limitations.

Another group of methods utilizes hydroxyl radical reactions with alkyl C-H bonds. The OH group tends to react mainly with surface-exposed residues providing a good footprint of the solvent exposed surface of the protein(s) [GCA00, SBH04]. The modification is covalent and thus irreversible, but each modification can potentially change the conformation of the protein, thus skewing results.

Exchange of labile hydrogens for deuteriums (HDX) as a probe of protein surface accessibility does not change the conformation of the protein. Advantages over NMR and X-ray crystallography structural determination are the ability to work at low concentration and high molecular weight.

Figure 3.1 shows the schematic of an HDX experiments. The experiment is initiated by dilution of the protein solution into a biological buffer made with $D_2O$. Solvent accessible hydrogens are exchanged with deuterium. The exchange is quenched (greatly slowed) by dropping the pH to between pH 2.3 and pH 2.5 and lowering the temperature to approximately 0° C. The protein complex is digested with a protease that is active under quench conditions (such as pepsin) and on-line liquid chromatography is performed directly to the FT-ICR MS (Fourier transform ion cyclotron resonance mass spectrometry). Deuterium incorporation is monitored by the increase in mass of each peptic fragment as the deuteron is added.

The data sets produced are large and each spectrum has hundreds of overlapping peptic fragments. From these data, the exchange rate is easily determined for the same

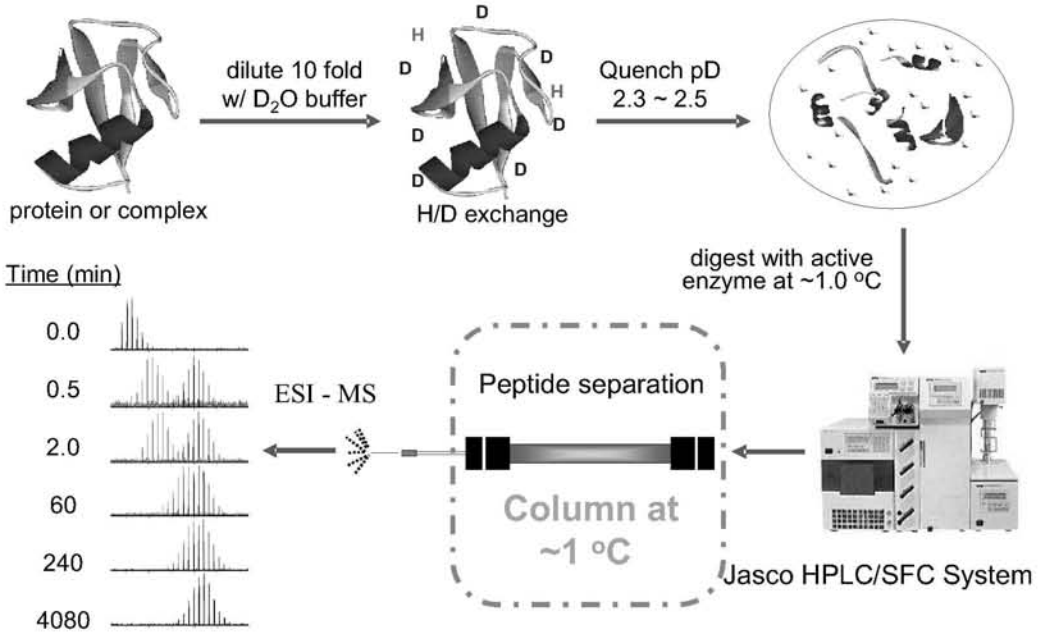



Figure 3.1: Schematic diagram of HDX experiment. The experiment starts by diluting the protein solution 10 fold into deuterated buffer, after which exchange occurs for predetermined time points, and is then quenched by lowering the temperature and dropping the pH. The protein is then digested and injected onto the HPLC (High-performance liquid chromatography) for separation and mass analysis.

peptic fragments from the protein and the protein-protein complex [LLE+02]. When peptic fragments are not directly comparable, but are overlapping (see Table 3.1 for an example), manual interpretation must be performed to assign exchange rates to individual residues. HDX data analysis is the real bottle neck in these experiments, thus an automated computational method for HDX rates is necessary.

Furthermore, we are interested in all possible exchange rate assignments, as we want to determine protein conformation, protein-protein interaction, and protein-ligand interactions. These data will be useful in the design and synthesis of small molecules to be used as therapeutic agents.

### 3.1.2 Formal Problem Definition

The problem of assigning exchange rates to residues as arising in the HDX experiments, can be rephrased in mathematical terms as follows. We are given a protein of $n$ residues and a set of peptic fragments, which correspond to intervals of $[n]$. The fragments cover the whole protein and may overlap. Furthermore, there are $k$ possible exchange rates to which we refer as colors in the following. The goal is to assign a color to the elements in set $[n]$ using $k$ colors such that a given set of requirements is satisfied. Each requirement

Table 3.1: Overlap of peptic fragments as seen in an HDX of the model protein myoglobin. It shows the total number of amide hydrogens exchanged in each peptide (All) and the number of amide hydrogens predicted to be either Slow, Medium or Fast by maximum entropy method (MEM) evaluation of the H/D exchange rate distribution [ZGM97].

| No | Amino acid sequence | All | Slow | Medium | Fast |
|---|---|---|---|---|---|
| 1 | *GLSDGEWQQVLNVWGKVEADIAGHGQEVL* | 28 | 15 | 8 | 5 |
| 2 | *GLSDGEWQQVL* | 10 | 7 | 2 | 1 |
| 3 | *NVWGKVEAD* | 8 | 5 | 2 | 1 |
| 4 | *VLNVWGKVEADIAGHGQE* | 17 | 12 | 1 | 4 |
| 5 | *NVWGKVEA* | 7 | 5 | 1 | 1 |
| 6 | *WQQVLNVWGKVEADIAGHGQEVL* | 15 | 11 | 1 | 3 |
| 7 | *GLSDGEW* | 6 | 4 | 1 | 1 |
| 8 | *WQQVL* | 4 | 3 | 1 | 0 |
| 9 | *IAGHGQEVL* | 8 | 7 | 1 | 0 |

is made up of a closed interval $I \subseteq [n]$ and a complete specification of how many elements in $I$ should be colored with each color.

**Interval Constraint Coloring** More formally, in an idealized setting we consider the following problem. We are given a set of intervals $\mathcal{I} \subseteq \{[i,j] \mid 1 \leq i \leq j \leq n\}$ with $|\mathcal{I}| = m$ defined on the set $V = [n]$, a set of color classes $[k]$, and we are given a requirement function $r : \mathcal{I} \times [k] \to \mathbb{Z}_+$ such that $\sum_{c\in[k]} r(I,c) = |I|$ for all $I \in \mathcal{I}$. We represent elements in $V$ by vertices numbered from 1 to $n$. A coloring $\chi : V \to [k]$ of the vertices is said to be *feasible* if for every $I \in \mathcal{I}$ we have

$$|\{i \in I \mid \chi(i) = c\}| = r(I,c) \text{ for all } c \in [k]. \tag{3.1}$$

**Problem 3.1** (INTERVALCOLORING)**.** *Given an instance $(V, I, k, r)$, compute a feasible coloring $\chi : V \to [k]$, if one exists.*

Where the meaning is clear from the context, we will simply use the term *vertex* to refer to the integer in $V$ it represents, such that intervals in $\mathcal{I}$ *contain vertices*. We use $N_\chi(I,c)$ to denote the number of vertices in interval $I$ colored $c$ by $\chi$, that is,

$$N_\chi(I,c) := |\{i \in I \mid \chi(i) = c\}|. \tag{3.2}$$

**Error-Minimal Interval Constraint Coloring** However, data collected in real experiments usually contain some noise, such that a feasible coloring does not necessarily exist. In this case, we would like to compute a (all) coloring(s) that minimize(s) the

total sum of errors. We define the error of a coloring $\chi$ in interval $I \in \mathcal{I}$ with respect to color $c$ by

$$e_{I,c} := \big|r(I,c) - N_\chi(I,c)\big|. \tag{3.3}$$

**Problem 3.2** (INTERVALCOLORING$_{\min}$)**.** *Given an instance $(V, I, k, r)$, an optimal coloring minimizes*

$$\sum_{I \in \mathcal{I}} \sum_{c \in [k]} e_{I,c}. \tag{3.4}$$

We denote by $\mathbf{e}_c = (e_{I,c})_{I \in \mathcal{I}_P}$ the vector of errors with respect to color $c$. Since in our application domain optimal assignments of exchange rates (colors) are evaluated in a second step by biochemists, we also describe algorithms, that are able to enumerate *all* optimal solutions to problem INTERVALCOLORING$_{\min}$.

**Maximizing Interval Constraint Coloring** Another way of dealing with instances that do not admit a feasible coloring is to compute a coloring that maximizes the number of intervals satisfying (3.1). More generally, we assume a nonnegative weight $w(I)$ associated with intervals in $\mathcal{I}$, and seek a maximum weight subset of intervals for which a feasible coloring exists.

**Problem 3.3** (INTERVALCOLORING$_{\max}$)**.** *Given an instance $(V, \mathcal{I}, k, r)$ and nonnegative weights $w(I)$ associated with each interval $I \in \mathcal{I}$, find a subset $\mathcal{I}' \subseteq \mathcal{I}$, maximizing $w(\mathcal{I}') := \sum_{I \in \mathcal{I}'} w(I)$, such that there exists a coloring of $V$ satisfying (3.1) for each $I \in \mathcal{I}'$.*

Due to the one-to-one correspondence between residues and vertices in $V$, between peptic fragments and intervals in $\mathcal{I}$, and between exchange rates and colors, we use the respective terms interchangeably, whenever it is not crucial for the understanding to focus either on the biological data or on the mathematical abstraction.

## 3.2 A Polyhedral Approach

In this section, we present a polyhedral description of the interval constraint coloring problem, which will serve as a basis for an enumerative algorithm similar to branch-and-bound.

In an initial binary (0-1) integer programming (BIP) formulation of the problem, we encode the choice of coloring a vertex with a certain color by a binary variable. In our experiments it turns out, however, that by using this formulation, a single integer solution (coloring) can be found efficiently, whereas finding all colorings can become a very time-consuming task. This is mainly due to the large number of feasible colorings, which we combine to equivalence classes in an improved integer linear programming (ILP) formulation. Since integer variables in the new ILP formulation introduced in Section 3.2.2 provide aggregate information about the distribution of colors within certain intervals, the number of solutions to the improved ILP formulation is just a fraction of the number of solutions to the original BIP formulation. Furthermore, colorings contained in an equivalence class can be easily derived from the corresponding solution to the new ILP.

### 3.2.1 A Binary IP Formulation

The interval constraint coloring problem (IntervalColoring) is captured by the binary integer program (BIP) given below, i.e. the feasible solutions of this program correspond to feasible colorings.

Let $\chi : V \to [k]$ be an assignment of colors to residues in $V = [n]$ and let $x_{i,c}$ be an indicator variable denoting whether $i$ is colored $c$ or not. More precisely, for $i \in V$ and $c \in [k]$, $x_{i,c} = 1$ if $\chi(i) = c$, and $x_{i,c} = 0$ otherwise. We let $\mathbf{x}_c = (x_{1,c}, x_{2,c}, \ldots, x_{n,c})$ be the vector of binary variables modeling the assignment of color $c$ and let $\mathbf{x} = (\mathbf{x}_c)_{c\in[k]}$, $\mathbf{x} \in \{0,1\}^{kn}$.

The first constraint in integer program (3.5) ensures that exactly one color is assigned to each residue. Feasible colorings are in one-to-one correspondence with assignments of 0-1 values to components of $\mathbf{x}$, if they furthermore satisfy the second constraint, i.e. if every requirement imposed by the intervals is satisfied.

$$\begin{aligned} \textstyle\sum_{c\in[k]} x_{i,c} &= 1, && \forall\, i \in [n], \\ \textstyle\sum_{i\in I} x_{i,c} &= r(I,c), && \forall\, I \in \mathcal{I}, c \in [k], \\ x_{i,c} &\in \{0,1\}, && \forall\, i \in [n], c \in [k]. \end{aligned} \tag{3.5}$$

In problem $\text{IntervalColoring}_{\min}$, the error minimization variant of the interval constraint coloring problem, we aim to minimize a sum of absolute values (3.4). If we translate the definition of the error that we make when coloring residues in interval $I$

with color $c$ (see (3.3)) into the context of 0-1 assignments to variables $\mathbf{x}_c$, we are dealing with the problem of minimizing

$$\sum_{I\in\mathcal{I}}\sum_{c\in[k]}\Big|r(I,c)-\sum_{i\in I}x_{i,c}\Big|.$$

We observe that $|r(I,c)-\sum_{i\in I}x_{i,c}|$ is the smallest number $e_{I,c}$ that satisfies $r(I,c)-\sum_{i\in I}x_{i,c}\leq e_{I,c}$ and $-r(I,c)+\sum_{i\in I}x_{i,c}\leq e_{I,c}$, and we obtain the integer linear programming formulation

$$\begin{aligned}
\text{minimize}\quad & \textstyle\sum_{I\in\mathcal{I}}\sum_{c\in[k]}e_{I,c} && && (3.6)\\
\text{subject to}\quad & \textstyle\sum_{c\in[k]}x_{i,c}=1, && \forall\, i\in[n], && (3.7)\\
& \textstyle\sum_{i\in I}x_{i,c}-r(I,c)\leq e_{I,c}, && \forall\, I\in\mathcal{I}, c\in[k], && (3.8)\\
& \textstyle-\sum_{i\in I}x_{i,c}+r(I,c)\leq e_{I,c}, && \forall\, I\in\mathcal{I}, c\in[k], && (3.9)\\
& x_{i,c}\in\{0,1\}, && \forall\, i\in[n], c\in[k]. && (3.10)
\end{aligned}$$

We refer to this integer linear program as *basic-BIP*. The number of solutions to the basic-BIP is typically very large (see Table 3.2 in Section 3.6). Intuitively, there are relatively long intervals in which no fragment starts or ends. Clearly, feasibility and the respective objective value are invariant under the permutation of colors within these intervals. Solutions derived by such permutations from each other are put into equivalence classes in the next section.

### 3.2.2 An Improved ILP Formulation

From a feasible solution to the binary integer program formulation introduced above one can easily obtain a new feasible solution that is equivalent in the following sense.

**Definition 3.1** (Equivalent Coloring)**.** *Let $\mathcal{P}_\mathcal{I}$ be the partition of $[n]$ into a minimal number of intervals, such that for each interval $J\in\mathcal{P}_\mathcal{I}$ and each interval $I\in\mathcal{I}$ either $J\subseteq I$ or $J\cap I=\emptyset$. We call two feasible colorings $\chi$ and $\chi'$* equivalent*, if they differ only by a permutation of colors within intervals of $\mathcal{P}_\mathcal{I}$.*

See Figure 3.2 for an example partition $\mathcal{P}_\mathcal{I}$ of 9 vertices into 4 intervals.

Thus, from a coloring $\chi$ any equivalent coloring $\chi'$ can be produced by a sequence of transpositions restricted to intervals in $\mathcal{P}_\mathcal{I}$. Thereby a coloring $\chi_2$ is obtained from a coloring $\chi_1$ by a transposition $\tau=(i,j)$, $i,j\in J$, $J\in\mathcal{P}_\mathcal{I}$, if $\chi_2(i)=\chi_1(j)$, $\chi_2(j)=\chi_1(i)$ and $\chi_2(l)=\chi_1(l)$ for $l\in J$, $l\neq i,j$. Note that equivalent colorings exhibit the same total error.

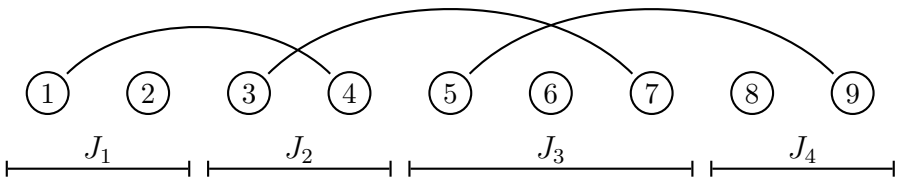



Figure 3.2: In this example, partition $\mathcal{P}_\mathcal{I}$ of vertex set $[9]$ implied by intervals $[1,4],[3,7],[5,9] \in \mathcal{I}$, as introduced in Definition 3.1, contains intervals $J_1$, $J_2$, $J_3$, and $J_4$.

To prevent our branching approach (see Section 3.2.3) from enumerating all equivalent solutions, we modify the proposed basic-BIP as follows. For $c \in [k]$ and $I \in \mathcal{P}_\mathcal{I}$, we replace the binary variables $x_{i,c}$ with $i \in I$ by a single integer variable $y_{I,c}$ with $y_{I,c} := \sum_{i \in I} x_{i,c}$. From now on, we assume that intervals $I_i \in \mathcal{I}$ are numbered from $i = 1$ to $i = m$ and intervals $J_j \in \mathcal{P}_\mathcal{I}$ are numbered from $j = 1$ to $j = |\mathcal{P}_\mathcal{I}|$. Moreover, we let $\mathbf{A}$ be the $(0,1)$-matrix of inclusions for intervals in partition $\mathcal{P}_\mathcal{I}$ versus intervals in $\mathcal{I}$, i.e. $\mathbf{A}$ is a $|\mathcal{I}| \times |\mathcal{P}_\mathcal{I}|$ matrix where for every $I_i \in \mathcal{I}$ and $J_j \in \mathcal{P}_\mathcal{I}$ the corresponding entry is given by

$$a_{i,j} = \begin{cases} 1 & \text{if } J_j \subseteq I_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3.11}$$

In the following, we let vector $\mathbf{y}_c = (y_{I,c})_{I \in \mathcal{P}_\mathcal{I}}$ and vector $\mathbf{r}_c = \big(r(I,c)\big)_{I \in \mathcal{P}_\mathcal{I}}$ denote the number of residues colored $c$, respectively the number of residues required to be colored $c$ by the instance. In matrix notation, constraints (3.8) and (3.9) then become

$$\begin{aligned} -\mathbf{A}\mathbf{y}_c + \mathbf{e}_c &\geq -\mathbf{r}_c \\ \mathbf{A}\mathbf{y}_c + \mathbf{e}_c &\geq \mathbf{r}_c \end{aligned}$$

for all $c \in [k]$. Hence our integer linear program can be rewritten as

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{I \in \mathcal{I}} \sum_{c \in [k]} e_{I,c} \\ \text{such that} \quad & -\mathbf{A}\mathbf{y}_c + \mathbf{e}_c \geq -\mathbf{r}_c, && \forall\, c \in [k], && (3.12) \\ & \mathbf{A}\mathbf{y}_c + \mathbf{e}_c \geq \mathbf{r}_c, && \forall\, c \in [k], && (3.13) \\ & \textstyle\sum_{c \in [k]} \mathbf{y}_c = \mathbf{p} && && (3.14) \\ & \mathbf{y} \geq \mathbf{0} \\ & \mathbf{y} \text{ integer.} \end{aligned}$$

where $\mathbf{p}$ is the vector that contains $|J|$ for each interval $J \in \mathcal{P}_{\mathcal{I}}$ and $\mathbf{y} = (\mathbf{y}_c)_{c\in[k]}$. We refer to this integer linear program as *improved-ILP*. Note that a feasible nonnegative integer solution $\mathbf{y}$ to (3.12)-(3.14) represents an equivalence class of feasible colorings.

### 3.2.3 Enumerating all Optimal Colorings

We now discuss a *divide-and-conquer* approach to generate all optimal colorings to problem IntervalColoring$_{\min}$, i.e. all colorings that are optimal with respect to cost criterion (3.4). Merely for simplicity of notation, we base the presentation of our algorithm on the basic-BIP formulation using indicator variables $\mathbf{x}$. Essentially, however, all the ideas carry over to the improved-ILP formulation using integer variables $\mathbf{y}$ as defined above.

The enumerative relaxation method we propose is adapted from the the standard branch-and-bound algorithm, which is described in detail in [Wol98]. We first determine the minimum possible total error $e$ of any coloring by a standard branch-and-bound algorithm that uses linear programming relaxations to obtain lower bounds on the total error. If we then add the constraint

$$\sum_{I\in\mathcal{I}} \sum_{c\in[k]} e_{I,c} \leq e \tag{3.15}$$

to constraints (3.7)-(3.10), we are faced with the problem of computing all feasible solutions of an integer linear program. Here our approach differs from the standard branch-and-bound algorithm in terms of storing integer solutions, pruning criteria, and branching. Nevertheless, it solves a linear programming relaxation in the nodes of an enumeration tree.

Instead of storing an integer solution only if it improves on the cost of the best feasible solution so far, we store all integer solutions we encounter in the enumeration tree. In particular, any feasible solution satisfies (3.15) and thus is optimal in the original sense (3.6).

In standard branch-and-bound, a node of the enumeration tree is pruned whenever the subproblem it defines cannot yield an *improving* integer solutions. In contrast, we can prune a node only if the corresponding subproblem does not admit *any additional* feasible integer solution, that can be guaranteed only if the corresponding linear programming relaxation is infeasible.

While in the standard branch-and-bound algorithm only fractional variables are chosen to be branched on, we might have to branch on variables that are integral too. Suppose the optimal solution $\mathbf{x}^*$ to the linear programming relaxation is integral and $x^*_{i,c} = 0$, $i \in [n]$, $c \in [k]$. Furthermore, assume variable $x_{i,c}$ has not been branched on in any previous node. As there might exist some solution with $x_{i,c} = 1$, we need to create a child node in which we set $x_{i,c} = 1$. At the same time, there might exist integer solutions with $x_{i,c} = 0$, but which differ from $\mathbf{x}^*$ in the other variables. Therefore, we also need

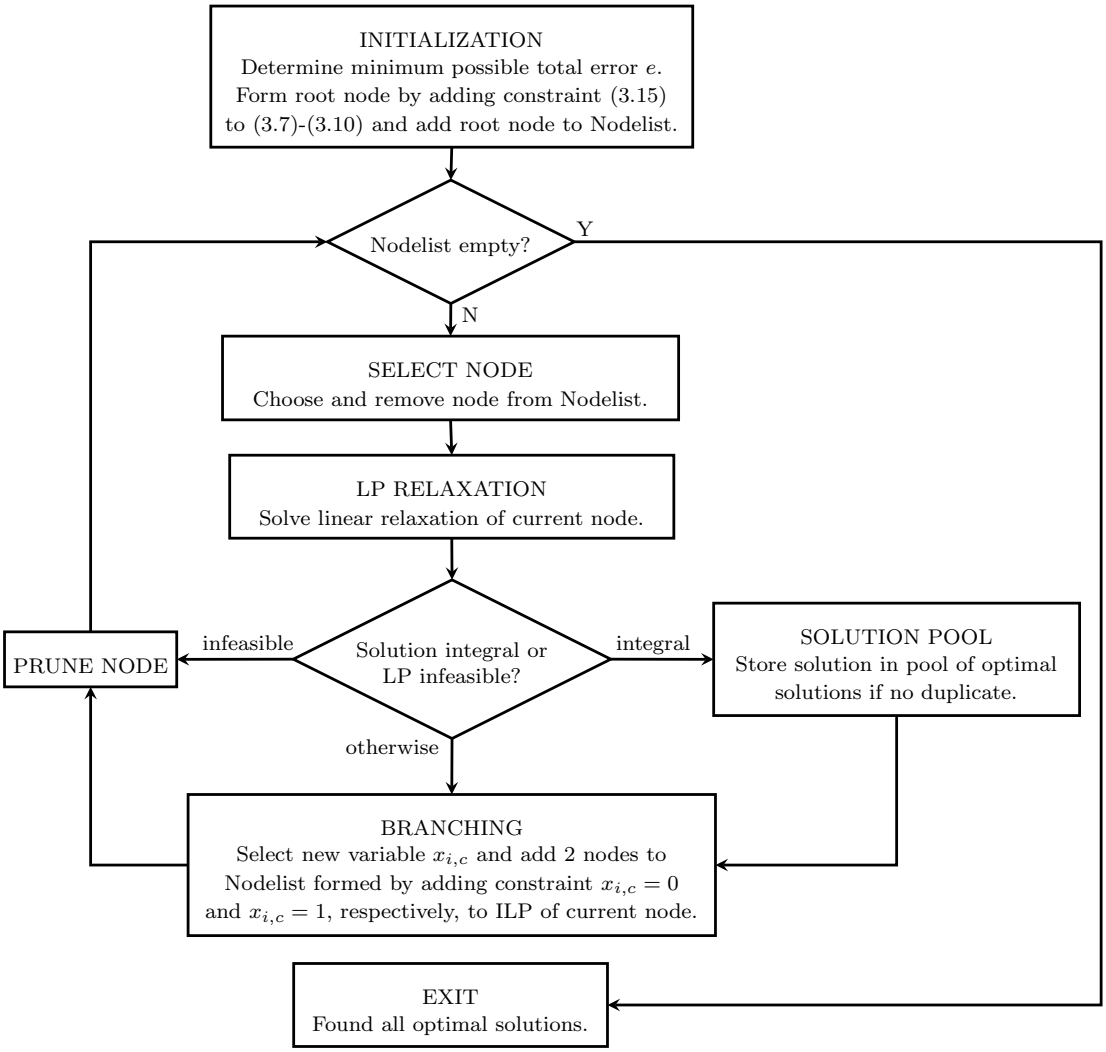



Figure 3.3: Flow chart of the B&B algorithm to enumerate all optimal solutions

to create a child node where we fix $x_{i,c}$ to 0. Notice however, that this branch contains the same integer solution as the parent and thus some care has to be taken to avoid duplicate solutions. Nevertheless, our algorithm will terminate, as there is only a finite number of variables to branch on. A simplified flow chart of the proposed algorithm is given in Figure 3.3.

On the theory side, the feasibility problem for general integer linear programs is $\mathcal{NP}$-complete. Although state-of-the-art ILP solvers perform well in practice, they exhibit exponential runtime in the worst case. Nevertheless, we were able to exploit the specific structure of the interval constraint coloring problem in the special case of two colors and give a combinatorial, strongly polynomial-time algorithm in the next section. This algorithm will serve as a building block for an approximation algorithm for the case of three and more colors in Section 3.4 and will also play a decisive role in the Lagrangian relaxation approach presented in Section 3.5.

## 3.3 A Combinatorial Approach for the 2-Color Case

In this section, we present a combinatorial algorithm that finds an optimal coloring for the special case of problem IntervalColoring$_{\min}$ where $k = 2$ in strongly polynomial time. If we distinguish only between two colors, constraint (3.14) becomes $y_{I,1}+y_{I,2} = |I|$ for all $I \in \mathcal{P}_{\mathcal{I}}$. This allows us to simplify the integer linear program considerably. Replacing $y_{I,2}$ by $|I|-y_{I,1}$ in constraints (3.12) and (3.13) and omitting the color subscript of the $y$-variables yield constraints

$$\begin{aligned} -\mathbf{A}\mathbf{y} + \mathbf{e}_1 &\geq -\mathbf{r}_1 & \qquad \mathbf{A}\mathbf{y} + \mathbf{e}_2 &\geq \mathbf{f} - \mathbf{r}_2 \\ \mathbf{A}\mathbf{y} + \mathbf{e}_1 &\geq \mathbf{r}_1 & \qquad -\mathbf{A}\mathbf{y} + \mathbf{e}_2 &\geq -\mathbf{f} + \mathbf{r}_2 \end{aligned} \tag{3.16}$$

where $\mathbf{f}$ is the vector of fragment lengths, i.e. lengths of intervals in $\mathcal{I}$. We can eliminate half of the constraints based on the following observation. Let $\mathbf{r} := \max\{\mathbf{r}_1, \mathbf{f} - \mathbf{r}_2\}$ and $\bar{\mathbf{r}} := \min\{\mathbf{r}_1, \mathbf{f}-\mathbf{r}_2\}$, where the maximum and the minimum is taken componentwise. Let $\mathbf{y} \geq \mathbf{0}$ be an arbitrary integer solution to constraints (3.16) with total error $\sum_{I\in\mathcal{I}} e_{I,1} + e_{I,2}$. We may consider the error of the given solution $\mathbf{y}$ within each interval independently. From error vectors $\mathbf{e}_1$ and $\mathbf{e}_2$, we construct new vectors $\mathbf{e} = (e_I)_{I\in\mathcal{I}}$ and $\bar{\mathbf{e}} = (\bar{e}_I)_{I\in\mathcal{I}}$, respectively, by swapping certain components between them, depending on the value of the corresponding component of $\mathbf{r}$ and $\bar{\mathbf{r}}$:

$$e_I := \begin{cases} e_{I,1} & \text{if } r_I = r(I,1), \\ e_{I,2} & \text{otherwise.} \end{cases} \qquad \bar{e}_I := \begin{cases} e_{I,1} & \text{if } \bar{r}_I = r(I,1), \\ e_{I,2} & \text{otherwise.} \end{cases}$$

Concerning the contribution of each interval $I = I_i \in \mathcal{I}$ to the total error subject to the new error vectors $\mathbf{e}$ and $\bar{\mathbf{e}}$, we distinguish between three different cases:

$$e_{I,1} + e_{I,2} = \begin{cases} r_I - \bar{r}_I & \text{if } \bar{r}_I \leq \mathbf{a}_i'\mathbf{y} \leq r_I, \\ 2e_I + r_I - \bar{r}_I & \text{if } \mathbf{a}_i'\mathbf{y} > r_I, \\ 2\bar{e}_I + r_I - \bar{r}_I & \text{if } \mathbf{a}_i'\mathbf{y} < \bar{r}_I. \end{cases}$$

Since $e_I$ and $\bar{e}_I$ are relevant only in the second, respectively third case, it is sufficient to require vector $\mathbf{e}$ to be at least $\mathbf{A}\mathbf{y}-\mathbf{r}$ and vector $\bar{\mathbf{e}}$ to be at least $\bar{\mathbf{r}}-\mathbf{A}\mathbf{y}$. Hence, a coloring with minimum total error is an optimal integer solution to the linear programming

relaxation

$$\begin{array}{lrcl} \text{minimize} & \sum_{I\in\mathcal{I}} e_I + \bar{e}_I & & \\ \text{such that} & -\mathbf{A}\mathbf{y} + \mathbf{e} & \geq & -\mathbf{r} \\ & \mathbf{A}\mathbf{y} + \bar{\mathbf{e}} & \geq & \bar{\mathbf{r}} \\ & -\mathbf{y} & \geq & -\mathbf{p} \\ & \mathbf{e}, \bar{\mathbf{e}} & \geq & \mathbf{0} \\ & \mathbf{y} & \geq & \mathbf{0}. \end{array} \tag{3.17}$$

whose constraint matrix is totally unimodular [NW99] and whose right hand side is integral. Therefore, all its basic feasible solutions are integral [NW99]. The corresponding dual linear program is given by

$$\begin{array}{lrcl} \text{maximize} & -\mathbf{r}'\mathbf{u}_e + \bar{\mathbf{r}}'\mathbf{u}_{\bar{e}} - \mathbf{p}'\mathbf{u}_p & & \\ \text{such that} & -\mathbf{A}'\mathbf{u}_e + \mathbf{A}'\mathbf{u}_{\bar{e}} - \mathbf{u}_p & \leq & \mathbf{0} \\ & \mathbf{0} \leq \mathbf{u}_e, \mathbf{u}_{\bar{e}} & \leq & \mathbf{1} \\ & \mathbf{0} \leq \mathbf{u}_p, & & \end{array} \tag{3.18}$$

which is equivalent to (multiplying the objective function by $-1$ and introducing slack variables $\mathbf{u}_s$)

$$\begin{array}{lrcl} -\text{minimize} & \mathbf{r}'\mathbf{u}_e - \bar{\mathbf{r}}'\mathbf{u}_{\bar{e}} + \mathbf{p}'\mathbf{u}_p & & \\ \text{such that} & -\mathbf{A}'\mathbf{u}_e + \mathbf{A}'\mathbf{u}_{\bar{e}} - \mathbf{u}_p + \mathbf{u}_s & = & \mathbf{0} \\ & \mathbf{0} \leq \mathbf{u}_e, \mathbf{u}_{\bar{e}} & \leq & \mathbf{1} \\ & \mathbf{0} \leq \mathbf{u}_p, \mathbf{u}_s. & & \end{array} \tag{3.19}$$

We will show next that solving dual program (3.19) reduces to finding a minimum cost circulation in a directed graph. To this end, let $\mathbf{M}$ be the matrix formed by the coefficients of the equality constraints in the linear program (3.19), i.e.

$$\mathbf{M} := \begin{pmatrix} -\mathbf{A}' & \mathbf{A}' & -\mathbf{I} & \mathbf{I} \end{pmatrix}.$$

Notice that the 1's in each column of constraint matrix $\mathbf{M}$ appear consecutively while all remaining entries are 0. Therefore, we can transform $\mathbf{M}$ by row-operations like in Gaussian elimination into a node-arc incidence matrix of a directed graph as follows: We add the dummy constraint $0 = 0$ at the end and subtract from each row its predecessor to obtain a matrix $\widetilde{\mathbf{M}}$. The columns of matrix $\widetilde{\mathbf{M}}$ have only two nonzero elements, a $+1$ in the row corresponding to the source node and a $-1$ in the row corresponding to the target node of an edge. Since the right hand side remains unchanged, we get a *minimum cost circulation problem* on a directed graph $G$ with $|\mathcal{P}_\mathcal{I}| + 1$ nodes and $\mathcal{O}(|\mathcal{I}| + |\mathcal{P}_\mathcal{I}|)$ arcs [AMO93].

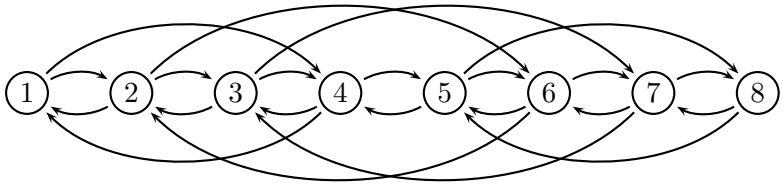



Figure 3.4: A directed graph $G$ with $|\mathcal{P}_\mathcal{I}| = 7$ as obtained from the reduction to a minimum cost circulation problem. The arcs $(1,4)$, $(2,6)$, $(3,7)$, $(5,8)$ map to intervals in $\mathcal{I}$ by the following rule: if $(i,j)$ is an arc in $G$, then there exists an interval $[i',j'] \in \mathcal{I}$ such that $i'$ is the leftmost node spanned by interval $I_i \in \mathcal{P}_\mathcal{I}$, and $j'$ is the rightmost node in interval $I_{j-1} \in \mathcal{P}_\mathcal{I}$.

More precisely, let the nodes in $G$ be identified by the number of the row in incidence matrix $\widetilde{\mathbf{M}}$ they correspond to. Furthermore, we number intervals $I_i \in \mathcal{P}_\mathcal{I}$ from $i = 1$ to $i = |\mathcal{P}_\mathcal{I}|$ such that interval $I_i$ corresponds to the $i$th row of matrix $\mathbf{M}$, or equivalently, to the $i$th column of $\mathbf{A}$ (see definition of matrix $\mathbf{A}$ in (3.11)). Notice that this numbering scheme naturally associates intervals $I_i$, and implicitly also variables $y_{I_i}$, with nodes $i$ for all $1 \leq i \leq |\mathcal{P}_\mathcal{I}|$. Node $|\mathcal{P}_\mathcal{I}| + 1$ plays a dedicated role, as it arises from the dummy constraint $0 = 0$. Then graph $G$ contains, for each variable $y_{I_i}$, $I_i \in \mathcal{P}_\mathcal{I}$, two arcs $(i, i+1)$ and $(i+1, i)$ corresponding to the constraints $0 \leq y_{I_i} \leq |I_i|$ and for each interval $[i,j] \in \mathcal{I}$ the arcs $(i', j'+1)$ and $(j'+1, i')$, where $i$ is the leftmost node in interval $I_{i'} \in \mathcal{P}_\mathcal{I}$ and $j$ the rightmost node spanned by interval $I_{j'} \in \mathcal{P}_\mathcal{I}$. Note that the number of fragments $|\mathcal{I}|$ is an implicit upper bound on the capacity of edges $(i, i+1)$ and $(i+1, i)$, $1 \leq i \leq |\mathcal{P}_\mathcal{I}|$: Every negative cost cycle contains at least on edge $(j'+1, i')$ of capacity 1 corresponding to an interval $[i,j] \in \mathcal{I}$. Figure 3.4 shows such a graph $G$ with $|\mathcal{I}_\mathcal{P}| = 7$ and $|\mathcal{I}| = 4$.

Two fundamental algorithms for solving the minimum most circulation problem are the *cycle-canceling algorithm* and the *successive shortest path algorithm* (see [AMO93] for further reference). The former maintains a feasible circulation at every step and attempts to attain optimality. The basic scheme is to start with the zero flow and repeatedly augment flow along negative cost directed cycles in the residual network until no negative cycle remains. Since the residual network with respect to an optimal circulation does not contain a negative cost (directed) cycle, we can obtain optimal node potentials $\pi = (\pi_i)_{1 \leq i \leq |\mathcal{P}_\mathcal{I}+1|}$, i.e. the corresponding dual solution, by solving a shortest path problem using the *Bellman-Ford algorithm* in time $\mathcal{O}(|\mathcal{I}| \cdot |\mathcal{P}_\mathcal{I}|)$ [CLRS01]. The difference in the potential of two neighboring nodes then yields the value of the corresponding $y$-variable, i.e. $y_i = \pi_i - \pi_{i+1}$. The values for the variables in the original improved-ILP formulation can be obtained for all $I \in \mathcal{P}_\mathcal{I}$ by $y_{I,1} = y_I$ for color 1 and by $y_{I,2} = |I| - y_I$ for color 2. Given an optimal solution $\mathbf{y}$, its total error is obtained in a straightforward way by summing up, over all intervals $I \in \mathcal{I}$, the deviation of a coloring contained in the equivalence class represented by $\mathbf{y}$ from requirements $r(I,1)$ and $r(I,2)$. If there exists a solution without an error this approach yields a solution within the running time of the Bellman-Ford algorithm.

In contrast, the node potentials $\pi$ play a decisive role in the successive shortest path algorithm. In a preprocessing step, we transform the minimum cost circulation problem into a minimum cost flow problem by saturating all negative cost arcs, which will

create excess nodes and deficit nodes. Then the algorithm maintains, at every step, a solution $\mathbf{u} = (\mathbf{u}_e, \mathbf{u}_{\bar{e}}, \mathbf{u}_p, \mathbf{u}_s)$ to (3.19) violating only the equality constraints and node potentials $\pi$ that satisfy the *reduced cost optimality condition* and attempts to achieve feasibility. Since in our case the total excess supply is upper bounded by $|\mathcal{P}_\mathcal{I}|$, the algorithm terminates in at most $|\mathcal{P}_\mathcal{I}|$ iterations. By using Dijkstra's algorithm [CLRS01] to solve a shortest path problem with nonnegative arc lengths in each iterations, the overall complexity of this algorithm is $\mathcal{O}\left(|\mathcal{P}_\mathcal{I}| \cdot |\mathcal{I}| + |\mathcal{P}_\mathcal{I}|^2 \log |\mathcal{P}_\mathcal{I}|\right)$.

In the next section we show, how we can obtain an approximate solution (without performance guarantee) to general instances with arbitrary but fixed number of colors by repetitively solving a special case involving only 2 colors.

## 3.4 An Approximation for the General Case

We present an algorithm that uses our combinatorial approach for the 2-color case from previous section as a subroutine to provide approximate solutions (without performance guarantee) for instances of problem IntervalColoring$_{\min}$ with arbitrary but fixed number of colors. The general idea is to reduce the problem to the 2-color case by merging all but one color, say color $i$, to a single color and solve the resulting problem by an algorithm for the minimum cost circulation problem, as described in Section 3.3. We remove nodes colored $i$ by the obtained solution and solve the coloring problem on the remaining nodes using $k-1$ colors recursively.

Procedure ApproxGeneral shows below the pseudocode describing our algorithm. It takes as parameter an instance of problem IntervalColoring$_{\min}$ and uses two subroutines: 2ColorsSpecial computes for the special case of 2 colors, i.e. $k = 2$, an integer solution $(\mathbf{y}_1, \mathbf{y}_2)$ to constraints (3.12)-(3.14) with minimum total error. ReduceInstance removes all nodes colored $i$ according to given solution vector $y_i$ and adapts all intervals and coloring requirements to the new set of vertices.

**Algorithm 1**: ApproxGeneral($V$, $\mathcal{I}$, $k$, $r$)

**Data**: An instance $(V, \mathcal{I}, k, r)$ of IntervalColoring$_{\min}$
**Result**: Colorings $\chi : V \mapsto [k]$ with small total error, represented by solution vector $(\mathbf{y}_c)_{c\in[k]}$

```
1 if k = 2 then
2     (y_1, y_2) ← 2ColorsSpecial(V, I, r)
3     return (y_1, y_2)
4 r̄_{I,1} ← Σ_{c=1}^{k-1} r_{I,c}, ∀I ∈ I
5 r̄_{I,2} ← r_{I,k}, ∀I ∈ I
6 (ȳ_{k-1}, y_k) ← 2ColorsSpecial(V, I, r̄)
7 (V', I', k-1, r') ← ReduceInstance(y_k)
8 (y_c)_{c∈[k-1]} ← ApproxGeneral(V', I', k-1, r')
9 return (y_c)_{c∈[k]}
```

Initially, let $|V| = n$ and $r : \mathcal{I} \times [k] \to \mathbb{Z}_+$. Provided that $k > 2$, the instance is reduced to the 2-color case with new requirement function $\bar{r} : \mathcal{I} \times [2] \to \mathbb{Z}_+$ in lines **4** and **5**. For the ease of notation, colors 1 to $k-1$ are merged to a new color 1, while color $k$ is kept unchanged as color 2 in the new instance.

Based on solution vector $\mathbf{y}_k$ for color $k$ derived from the dual solution of a minimum cost circulation problem in line **6**, we construct an instance $(V', \mathcal{I}', k-1, r')$ in line **7**, in which from every $I \in \mathcal{P}_{\mathcal{I}}$ exactly $y_{I,k}$ residues are removed. Those residues will be colored $k$ in the final solution to the original instance. Therefore, $|V'| = n - \sum_{I\in\mathcal{P}_{\mathcal{I}}} y_{I,k}$ and we remove empty intervals in $\mathcal{I}$. Furthermore, the boundaries of the remaining intervals are shifted to the left to account for deleted residues. Color 1 is split into its $k-1$ constituting colors again.

From the recursive call in line **8** we obtain an approximate solution $(\mathbf{y}_c)_{c\in[k-1]}$ to the reduced problem with $k-1$ colors which we can finally combine with $\mathbf{y}_k$ to an overall solution $(\mathbf{y}_c)_{c\in[k]}$ to the original instance.

For the sake of illustration, we applied one specific strategy to merge colors in line **4**. Namely, in each recursive call we merge colors in $\{1,\dots,k-1\}$ to a new color 1 and keep color $k$ as new color 2. Clearly, the combinatorial algorithm does not dependent on the strategy being pursued when merging $k-1$ colors. For a fixed (small) number of colors one could therefore evaluate all possible orders in which colors are fixed as new color 2 and output the minimum overall error.

In the next section we use the combinatorial approach for the 2-color case to compute, in a Lagrangian fashion, a bound on the minimum total error, which is exploited in a branch-and-bound manner to determine all optimal colorings.

## 3.5 A Lagrangian Relaxation Approach

In this section we propose a *Lagrangian relaxation* approach for finding *all* optimal solutions to problem IntervalColoring$_{\min}$. It is based on the improved-ILP formulation introduced in Section 3.2.2:

$$\begin{aligned}
\text{minimize} \quad & \textstyle\sum_{I\in\mathcal{I}} \sum_{c\in[k]} e_{I,c} && && (3.20)\\
\text{such that} \quad & -\mathbf{A}\mathbf{y}_c + \mathbf{e}_c \geq -\mathbf{r}_c, && \forall\, c \in [k], && (3.21)\\
& \mathbf{A}\mathbf{y}_c + \mathbf{e}_c \geq \mathbf{r}_c, && \forall\, c \in [k], && (3.22)\\
& \textstyle\sum_{c\in[k]} \mathbf{y}_c = \mathbf{p} && && (3.23)\\
& \mathbf{y} \geq \mathbf{0} && &&\\
& \mathbf{y} \text{ integer}, && &&
\end{aligned}$$

where $\mathbf{p}$ is the vector that contains the length of intervals in $\mathcal{P}_\mathcal{I}$ and $\mathbf{y} = (\mathbf{y}_c)_{c\in[k]}$. The problem can be considered to contain independent structures for each color $c \in [k]$, namely the set of positive integer vectors $\mathbf{y}_c$ satisfying (3.21) and (3.22) under the objective (3.20), that are linked by constraints (3.23). Therefore, dualizing the linking constraints (3.23), with Lagrangian multipliers $\lambda$, splits the problem into an independent problem for each color $c \in [k]$:

$$\begin{aligned}
\text{minimize} \quad & \textstyle\sum_{I\in\mathcal{I}} \sum_{c\in[k]} e_{I,c} + \lambda' \left(\sum_{c\in[k]} \mathbf{y}_c - \mathbf{p}\right) && && \\
\text{such that} \quad & -\mathbf{A}\mathbf{y}_c + \mathbf{e}_c \geq -\mathbf{r}_c, && \forall\, c \in [k], && \\
& \mathbf{A}\mathbf{y}_c + \mathbf{e}_c \geq \mathbf{r}_c, && \forall\, c \in [k], && (IP(\lambda))\\
& \mathbf{0} \leq \mathbf{y}_c \leq \mathbf{p}, && \forall\, c \in [k], && \\
& \mathbf{y} \text{ integer}. && && 
\end{aligned}$$

Neglecting the constant term $-\lambda'\mathbf{p}$ in the objective function and replacing error variable $\mathbf{e}$ by $\mathbf{e} + \bar{\mathbf{e}}$ we have to determine, for every color $c \in [k]$, an optimal integral solution to the following linear program:

$$\begin{aligned}
\text{minimize} \quad & \textstyle\sum_{I\in\mathcal{I}} (e_{I,c} + \bar{e}_{I,c}) + \lambda'\mathbf{y}_c && && \\
\text{such that} \quad & -\mathbf{A}\mathbf{y}_c + \mathbf{e}_c \geq -\mathbf{r}_c && && (3.24)\\
& \mathbf{A}\mathbf{y}_c + \bar{\mathbf{e}}_c \geq \mathbf{r}_c && && (3.25)\\
& \mathbf{e}_c, \bar{\mathbf{e}}_c \geq \mathbf{0} && && (3.26)\\
& \mathbf{0} \leq \mathbf{y}_c \leq \mathbf{p}. && &&
\end{aligned}$$

Note that we added constraint (3.26) to enforce $e_{I,c}$ or $\bar{e}_{I,c}$ to be zero if $\bar{e}_{I,c}$, respectively $e_{I,c}$, corresponds to the absolute value of the error, i.e. if the constraint (3.25), respec-

tively the constraint (3.24), for interval $I$ is tight. Similar as for linear program (3.17), its dual is given by (omitting the color subscript $c$):

$$\begin{array}{rrcl} -\text{minimize} & \mathbf{r}'\mathbf{u}_e - \mathbf{r}'\mathbf{u}_{\bar{e}} + \mathbf{p}'\mathbf{u}_p & & \\ \text{such that} & -\mathbf{A}'\mathbf{u}_e + \mathbf{A}'\mathbf{u}_{\bar{e}} - \mathbf{u}_p + \mathbf{u}_s & = & \lambda \\ & \mathbf{0} \leq \mathbf{u}_e, \mathbf{u}_{\bar{e}} & \leq & \mathbf{1} \\ & \mathbf{0} \leq \mathbf{u}_p, \mathbf{u}_s, & & \end{array} \tag{3.27}$$

This linear program differs from LP (3.19) in Section 3.3 only in the right-hand side of the equality constraints. Instead of a minimum cost circulation problem (right-hand side is 0), we have to solve the more general *minimum cost flow* problem [AMO93] where the supplies and demands $\bar{\lambda}$ of the nodes are determined by the difference of Lagrangian multipliers, i.e. $\bar{\lambda}$ is of dimension $|\mathcal{P}_\mathcal{I}| + 1$ and $\bar{\lambda}_i = \lambda_i - \lambda_{i-1}$ for $2 \leq i \leq |\mathcal{P}_\mathcal{I}|$, $\bar{\lambda}_1 = \lambda_1$ and $\bar{\lambda}_{|\mathcal{P}_\mathcal{I}|+1} = -\lambda_{|\mathcal{P}_\mathcal{I}|}$. A feasible flow of minimum cost can be computed efficiently by, e.g., the *cycle-canceling algorithm* and the *successive shortest path algorithm*, as well as variants of them, like the *capacity scaling algorithm* [AMO93].

### 3.5.1 Solving the Lagrangian Dual

Let $v(IP(\lambda))$ denote the optimal value of $IP(\lambda)$. Then for any vector $\lambda$ of Lagrangian multipliers, the (nondifferentiable) Lagrangian function

$$z(\lambda) = v(IP(\lambda))$$

provides a lower bound on the minimum total error, see objective function (3.20). To profit from the sharpest possible bound in the branch-and-bound framework we are interested in solving the Lagrangian dual problem

$$z^* = \max_\lambda z(\lambda).$$

Similar as for the multiple sequence alignment problem (see Section 2.6.1), we apply the subgradient method to obtain near-optimal Lagrangian multipliers. Following the approach by Held and Karp [HK71] we iteratively determine values $\lambda^{\ell+1}$ for $\ell = 0, 1, \ldots$, of the Lagrangian multipliers by moving in the direction of a subgradient with "step length" $\mu_\ell$:

$$\lambda^{\ell+1} = \lambda^\ell + \mu_\ell \left( \sum_{c \in [k]} \mathbf{y}_c^\ell - \mathbf{p} \right),$$

where $(\mathbf{y}_c^\ell)_{c \in [k]}$ is any optimal solution to $IP(\lambda^\ell)$. In comparison to the update formula (2.22), Lagrangian multipliers $\lambda$ are not constrained to be nonnegative, since we are

dualizing equality constraints. The heuristic for computing the step length does not differ from the one used in (2.22):

$$\mu_\ell = \frac{\theta_\ell \left(UB - z(\lambda^k)\right)}{\| \sum_{c \in [k]} \mathbf{y}_c^\ell - \mathbf{p} \|^2} \ .$$

However, we experienced a faster convergence to near-optimal Lagrangian multipliers when following the classical Held-Karp method to choose the step size scalar $\theta$: We start with $\theta_0 = 2$ and half $\theta_\ell$ whenever the best Lagrangian bound $v(IP(\lambda))$ found so far has not increased in a certain number of iterations. As soon as the step size scalar falls below a specified threshold or the number of iterations exceeds a certain limit (which is adaptive with respect to the depth of the B&B node), we branch on a variable $y_{I,c}$ such that $\bar{y}_{I,c} - \lfloor \bar{y}_{I,c} \rfloor$ is close to 0.5, where $\bar{y}_{I,c}$ is the average value of variable $y_{I,c}$ in the last $h = 10$ Lagrangian solutions. Since we aim to find *all* optimal colorings, we also branch on variables that are integral. Section 3.2.3 describes the modifications required by the standard B&B algorithm in order to enumerate all optimal solutions. In particular, the LP relaxation in the flow chart given in Figure 3.3 has to be replaced by the proposed Lagrangian relaxation approach.

Experiments show (see Section 3.6.4), that incorporating the Lagrangian approach as a lower bounding scheme into a branch-and-bound framework, allows to enumerate all optimal colorings significantly faster than the B&B method based on LP relaxation.

## 3.6 Experiments

We have implemented the exact branch-and-bound (B&B) method based on the LP relaxation of the basic-BIP and the improved-ILP (see Section 3.2.3), the B&B approach based on the Lagrangian relaxation of the improved-ILP (see Section 3.5), as well as the combinatorial algorithm providing approximate solutions to the general case (see Section 3.4), in C++ using the LEDA library [MN95]. In the implementation of the former we used the C++ library SCIL [ABE+02] to solve integer linear programs. SCIL is based on libraries LEDA and SCIP [Ach04], which in turn uses CPLEX [ILO06] or SoPlex [Wun96] as solver for linear programs. We applied our solution methods to several instances obtained from real experiments as well as to randomly generated problem instances.

### 3.6.1 Basic-BIP versus Improved-ILP

Table 3.2 reports the results of our ILP-based branching approach on real-world instances, which contained between 28 and 57 residues and between 16 and 50 fragments. Residue *proline* was removed from the protein sequences, since it does not possess an amide hydrogen atom. The instances distinguish low, medium, and high exchange rates, i.e. $k = 3$. A coloring with minimum total error could be computed in less than 0.1 second for all those instances. All nonequivalent colorings (see Definition 3.1) with minimum total error, between 6 and 62 in number, could be determined in less than 5 seconds, where the running time greatly depends on the actual number of solutions. Using the improved-ILP instead of the basic-BIP formulation reduces the time required to compute all solutions significantly.

| Instance | | | | Basic-BIP | | | Improved-ILP | | |
|---|---|---|---|---|---|---|---|---|---|
| $n$ | $\lvert\mathcal{I}\rvert$ | $\lvert\mathcal{P}_\mathcal{I}\rvert$ | $\epsilon$ | T(One) | T(All) | #Sol | T(One) | T(All) | #Sol |
| 28 | 16 | 12 | 19 | 0.02 | 418.75 | 102600 | 0.02 | 0.13 | 11 |
| 57 | 34 | 28 | 10 | 0.05 | >3600 | > 300000 | 0.02 | 0.32 | 6 |
| 57 | 50 | 37 | 35 | 0.06 | 215.13 | 8568 | 0.04 | 4.32 | 62 |
| 37 | 17 | 16 | 9 | 0.03 | 2084.12 | 4529151 | 0.01 | 0.22 | 18 |

Table 3.2: The characteristics of the instances are given by the number of residues $n$, the number of fragments $|\mathcal{I}|$, the size of the partition $|\mathcal{P}_\mathcal{I}|$, and the minimum possible total error of a coloring $\epsilon$. For the basic-BIP and the improved-ILP formulation we give the solution times for finding one optimal coloring T(One) and for finding all optimal colorings T(All) in seconds, as well as the number of solutions #Sol found. The number of solutions to the improved-ILP is considerably smaller than the number of solutions to the basic-BIP since equivalent colorings are considered only once.

The results for the real-world instances are very promising as the small number of easily interpretable classes of equivalent solutions can be used in protein structure prediction tools and for manual inspection.

### 3.6.2 Improved-ILP on Random Instances

To be able to evaluate the performance of our B&B approach based on the improved-ILP formulation more accurately, we additionally created random instances as follows. We generated a sequence with a given number of residues. For each residue, we chose a color at random independently of one another. In our model, a color representing a low exchange rate is assigned with probability 0.6, whereas a color corresponding to medium or high exchange rate is selected with probability 0.2. These probabilities approximately reflect the numbers we observed in the real-world problem instances. The random fragments are obtained by selecting, $n/2$ times, $i, j \in \{1, \ldots, n\}$ with $i < j$ at random. In our experiments we use $n = 50, 100, 150, 200, 250, 300, 500, 1000$. Note that the actual number of variables in the improved-ILP formulation is less than $n$. However, the difference is only marginal due to our random choice of the relatively high number of fragments. For each fragment, we counted the numbers of residues having a certain color and added a random Gaussian noise to reflect the errors arising from the experimental measurements. We generated one series of instances without noise and three further series with Gaussian noise of mean 0 and different standard deviations. Care must be taken when using those artificial instances to support quantitative analysis, since they are generated by a rather simple model of the real experiments.

Figure 3.5 relates the computation time of our improved-ILP-based B&B approach required to find one optimal coloring for problem IntervalColoring$_{\min}$ to the number of variables used in our model. We observe a growing effect of noise the more variables are required to capture the instance. The lines fitting the data in the log-log plot of each series indicate that the running time obeys a power law. For the sake of illustration, we only show the straight line that supposedly best fits the running times for the instances without noise. It demonstrates, however, that the data points tend to follow a slight convex curvature. Moreover, the slope of this straight-line fit is roughly 2.1, whereas for the other series the exponent of the best fit power law tends towards 3 with growing noise. Although solving integer linear programs requires exponential time in general, this is apparently not the case for our approach in the considered range $n \leq 1000$. The overall running time of our branching algorithm seems to be dominated by the time needed to solve the linear programming relaxations. Since this involves solving systems of linear equations, we expect the number of variables and the running times to be related by a cubic polynomial for instances with growing noise. In fact, Figure 3.5 shows that the cubic-polynomial fit provides a good approximation to the running times on instances with high noise.

The effect of noise on the running times of the B&B approach becomes more apparent when we enumerate all optimal colorings (see Figure 3.6). Intuitively, a higher minimum total error admits a larger number of optimal colorings. Besides, the number of optimal colorings also grows with an increasing number of variables. The straight lines in the log-log plot are fits of power laws to the running times with exponents ranging from roughly 3.2 to about 4.3.

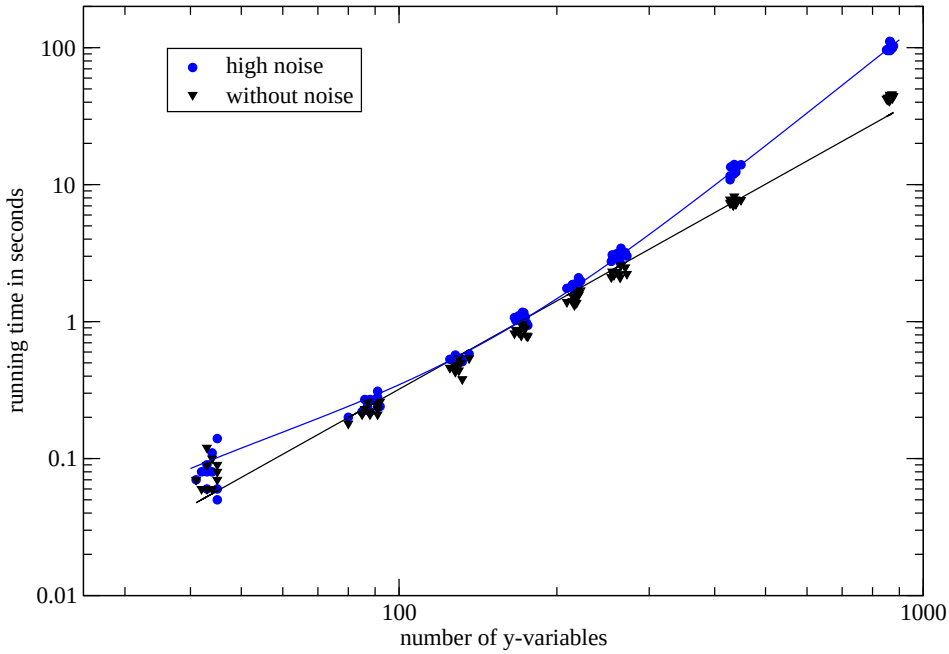



Figure 3.5: Running time of the improved-ILP-based B&B algorithm on two data sets. A straight line is fitted to the data on instances without noise. The curve fitted to the running times on instances with very high noise is a polynomial of degree 3.

### 3.6.3 Improved-ILP versus Combinatorial Approach

In the following, we compare our branching approach from Section 3.2.3 based on the improved-ILP formulation with the combinatorial method (see Section 3.3) providing approximate solutions. In the implementation of the latter, we refined algorithm ApproxGeneral to allow for different strategies of merging colors in line **4**. We applied the modified version of our algorithm for all three possibilities of merging two out of three colors and took the minimum overall error over all three objective function values. Note that while in the original formulation procedure ApproxGeneral merges colors 1 and 2 in line **4**, it can easily be adapted to the other cases.

In Figure 3.7 we evaluate the running times of the two approaches needed to find one optimal coloring. The log-log plot shows that both running times are in good agreement with the signature of a power law. However, the combinatorial approximation approach is considerably faster as its running time follows a power law with a scaling exponent of 1.2, while the running time of the ILP-based method scales with an exponent of roughly 2.6.

It remains to show that the advantage of the combinatorial approach with respect to the running time is not too much on the expense of the quality of the solution. To this end, we present a histogram in Figure 3.8 giving the number of instances (in percentage) for which the approximate solution has an error that differs from the minimum total error

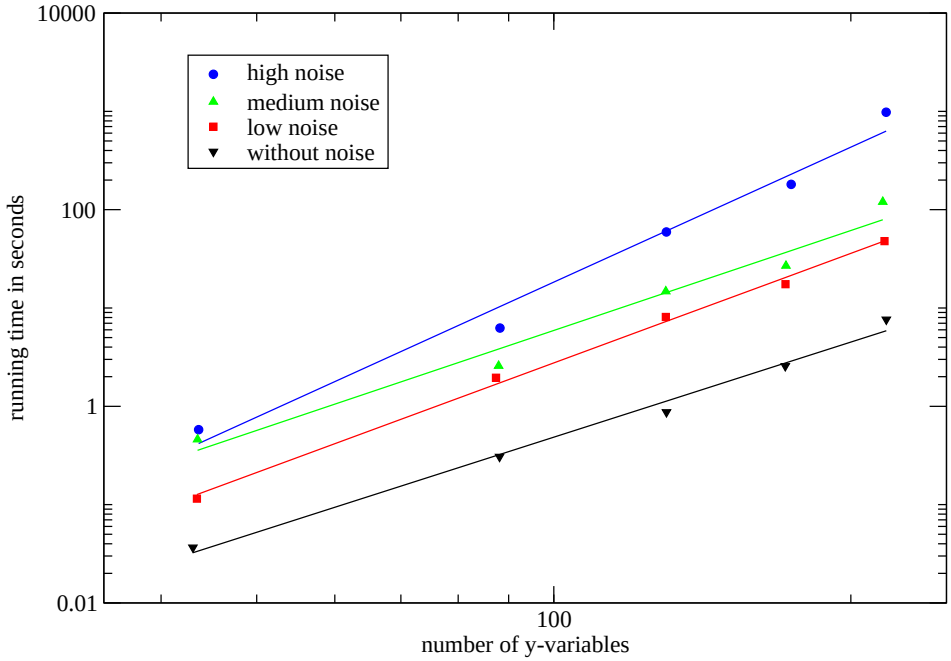



Figure 3.6: Time required by the improved-ILP-based B&B algorithm to enumerate all colorings with minimum error. For the sake of illustration, we only show the average running times on similarly large instances.

by a certain amount. Our combinatorial approach finds the optimal coloring in 78% of all instances, and in about 86% of the instances the error of our coloring exceeds the minimum total error by at most 2. It comes as no surprise that the error of our solution always deviates from the optimum by an even number. Intuitively, changing the color of a residue in an optimal coloring involves an increase or decrease in error with respect to two colors per fragment spanning that residue.

Moreover, the experiments show that if there exists a coloring with total error 0, then the combinatorial algorithm finds a optimal coloring in more than 99.5% of these cases. If the minimum total error is strictly positive, our approach achieves an average approximation ration of 1.01 and $\frac{11}{9}$ in the worst case. We conclude that the combinatorial approach is well suited for practical purposes. In particular, it can be integrated into a B&B framework to enumerate several "good" solutions if we allow a slight deviation from the minimal total error. Since it achieves an order of magnitude improvement in running time over the (exact) ILP approach, we can afford to enumerate a significantly higher number of nodes in the search tree.

An ultimate assessment of the practical performance of both approaches requires further experiments on real-world problem instances. However, creating instances by real experiments involves a considerable effort. For the sake of completeness, we briefly explain the applied techniques in Section 3.6.5

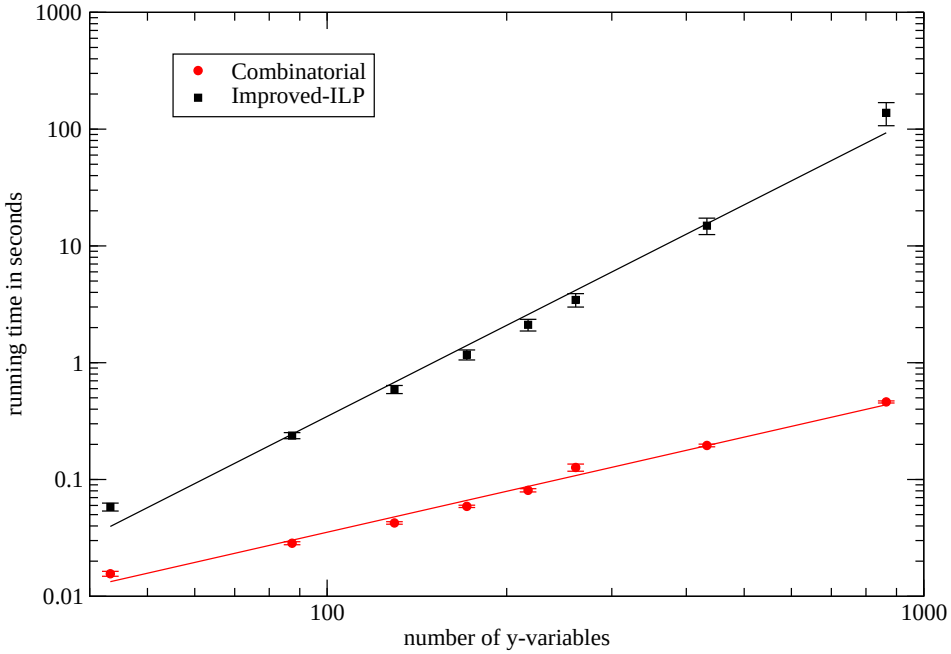



Figure 3.7: Comparison of running times of the branching method based on the improved-ILP formulation and the combinatorial approach.

### 3.6.4 B&B based on Lagrangian Relaxation

We have implemented the Lagrangian relaxation approach introduced in Section 3.5 in C++. We used the LEDA library [MN95] to solve the Lagrangian subproblem, a set of minimum cost flow problems, by an algorithm based on capacity scaling and successive shortest path computation [AMO93]. We improve the resulting bounds by the subgradient optimization method described in Section 3.5.1 and incorporate the overall approach into a branch-and-bound algorithm as the lower bounding scheme. In the experiments it turns out, that initializing the vector of Lagrangian multipliers $\lambda^0$ to the length $\mathbf{p}$ of the corresponding intervals in $\mathcal{P}_\mathcal{I}$ increases the convergence rate dramatically.

We applied the Lagrangian approach to the four data sets of random instances, whose generation is described in detail in Section 3.6.2. Although the running times required to compute *one* optimal coloring seem to obey a power law (Figure 3.9), with the exponent ranging from roughly 2.1 to 2.5, they are significantly higher than the computation times of the branching approach based on the improved-ILP formulation (see Figure 3.5). This comes as no surprise, since the linear programs in the B&B nodes can be solved relatively efficient while providing strong bounds on the minimal total error.

Nevertheless, the advantage of determining a bound on the minimal error by a combinatorial algorithm becomes apparent when enumerating *all* optimal colorings. The slopes of the straight-line fits in Figure 3.10 range from roughly 1.9 for instances without noise (compared to 3.2 for the improved-ILP-based branching method) to roughly

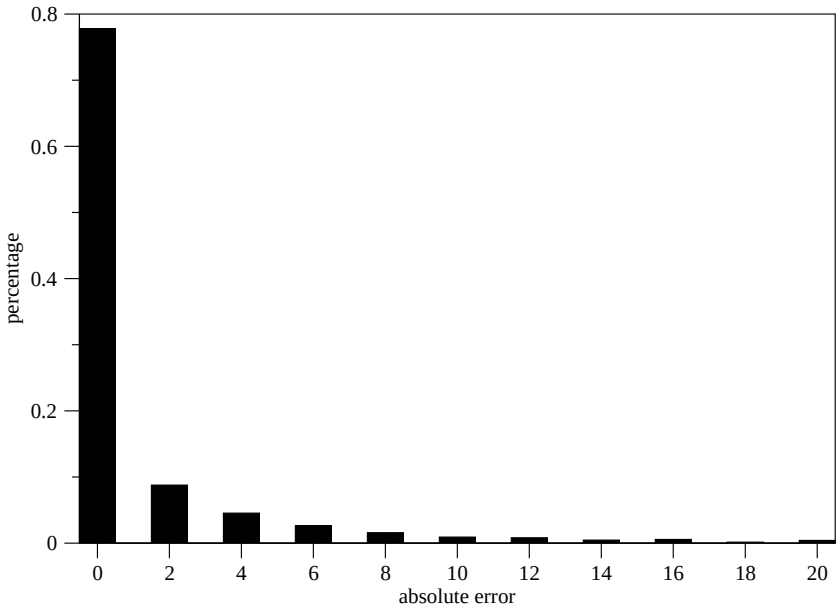



Figure 3.8: For even numbers $x$ between 0 and 20, the histogram shows the percentage of instances for which the error of the approximate solution exceeds the minimum total error by $x$.

2.7 for instances with high noise (compared to 4.3). Since the enumeration of all optimal solutions requires a high number of nodes in the search tree to be evaluated, we profit from the efficiency of algorithms to solve a minimum cost flow problem compared to a general purpose algorithm for solving a linear program.

### 3.6.5 HDX - Experimental Setting

The entire HDX experiment was automated with a LEAP robot (HTS PAL, Leap Technologies, Carrboro, NC). Automation of the experiment reduces human error and reduces deuterium for hydrogen back-exchange. All time points where interlaced and performed in triplicate to ensure experimental reproduceability. After digestion, the protein digest was injected from a 10 $\mu$L loop to either a 1 mm x 50 mm C5 column (Phenomenex) or a Pro-Zap Prosphere HP C18 HR 1.5u 10 mm x 2.1 mm (Alltech). A rapid gradient 2% B to 95% B in 1.5 min (A: acetonitrile/H2O/formic acid 5/94.5/0.5, B: acetonitrile/H2O/formic acid 95/4.5/0.5) was used to elute peptides. The eluent was post-column split and infused by microelectrospray ionization into a custom built 14.5 T LTQ FT-ICR mass spectrometer.

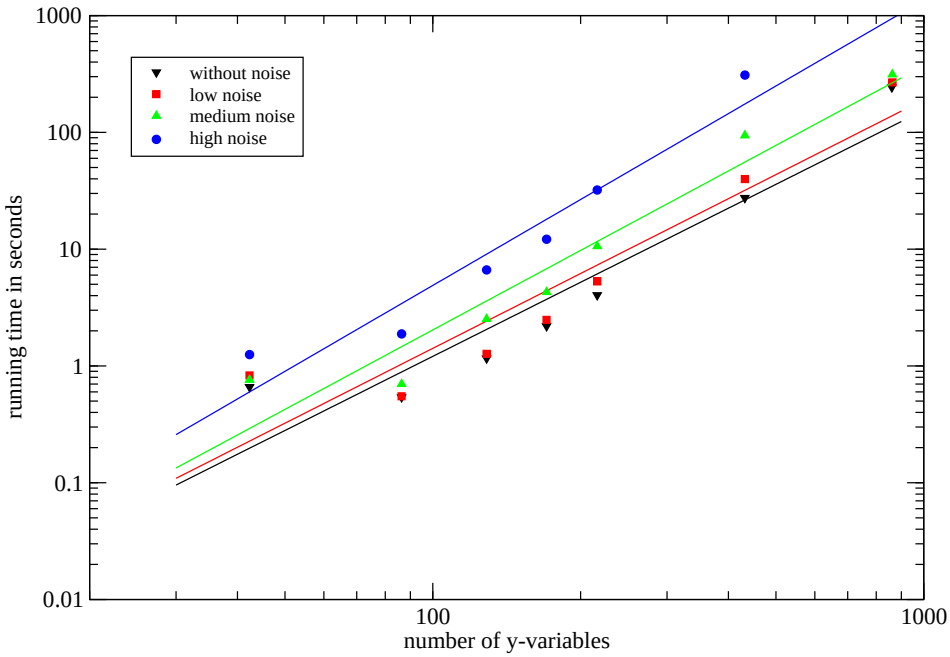



Figure 3.9: Running time of the Lagrangian-based B&B algorithm needed to compute one optimal coloring for the four sets of random instances. For the sake of illustration, we only show the average running times on similarly large instances.

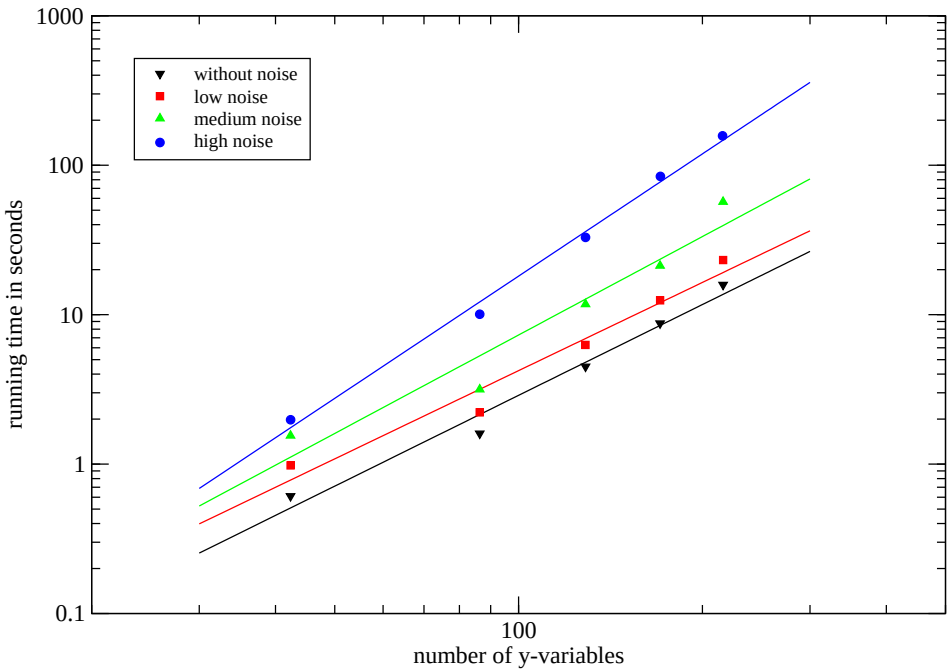



Figure 3.10: Running time of the Lagrangian-based B&B algorithm needed to compute all optimal colorings for the four sets of random instances. For the sake of illustration, we only show the average running times on similarly large instances.

## 3.7 A ±1 Guarantee

In this section, we present an algorithm that rounds a fractional solution to the LP relaxation polytope of ILP (3.5) in order to obtain an approximate solution to problem IntervalColoring that satisfies all coloring requirements (second constraint in ILP (3.5)) within a mere additive error of one.

Let $\mathcal{P}$ be the polytope obtained by relaxing the integrality constraint in integral problem (3.5). That is, $\mathcal{P}$ is the set of values of $\mathbf{x}$ obeying

$$\begin{aligned} \textstyle\sum_{c\in[k]} x_{i,c} &= 1, & \forall\, i \in [n],\\ \textstyle\sum_{i\in I} x_{i,c} &= r(I,c), & \forall\, I \in \mathcal{I}, c \in [k],\\ 0 \le x_{i,c} &\le 1, & \forall\, i \in [n], c \in [k]. \end{aligned}$$

Let $x$ be a fractional solution in $\mathcal{P}$. We use the scheme of Gandhi *et al.* [GKPS06] to round $x$ to an integral solution $\hat{x}$.

**Theorem 3.1.** *Given a fractional solution $x \in \mathcal{P}$, we can construct in polynomial time an integral solution $\hat{x}$ with the following properties*

*(P1) For every $i \in [n]$ there exists $c \in [k]$ such that $\hat{x}_{i,c} = 1$ and $\hat{x}_{i,d} = 0$ for all $d \neq c$.*

*(P2) For every $I \in \mathcal{I}$ and $c \in [k]$ we have $|\sum_{i\in I} \hat{x}_{i,c} - r(I,c)| \le 1$.*

*(P3) Every $I \in \mathcal{I}$ is satisfied with probability at least $\gamma_k = \frac{k(k+1-H_{k-1})}{(k+1)!}$.*

In other words, each position gets exactly one color (*P1*), every coloring requirement is off by at most one from the prescribed number (*P2*), and all the requirements for a given interval $I$ are satisfied *exactly* ($\sum_{i\in I} \hat{x}_{i,c} = r(I,c)$ for all $c \in [k]$) with probability at least $\gamma_k$ (*P3*). An interesting corollary of this theorem is that if $\mathcal{P}$ is non-empty then there always exists a coloring satisfying at least $\gamma_k|\mathcal{I}|$ intervals, and such a coloring can be found in polynomial time.

The high level idea is to simplify the polytope $\mathcal{P}$ into another integral polytope with basic solutions satisfying (*P1*) and (*P2*). Then we show how to select a basic solution satisfying (*P3*). This is done by defining a set of *blocks* and then setting up an assignment problem instance between $[n]$ and the set of blocks, whose polytope is integral.

For each color class $c \in [k]$ we choose a real number $\alpha_c \in [0,1]$, to be specified shortly. Let us define blocks $B_1^c, B_2^c, \ldots, B_{b_c}^c$: For color $c$ and $j = 2, \ldots, b_c - 1$

$$B_j^c = \left[\min\{t \mid \textstyle\sum_{i\le t} x_{i,c} > j - 2 + \alpha_c\},\ \min\{t \mid \textstyle\sum_{i\le t} x_{i,c} \ge j - 1 + \alpha_c\}\right).$$

The first and last blocks, $B_1^c$ and $B_{b_c}^c$, are defined similarly, but starting at 1 and ending at $n$ respectively.

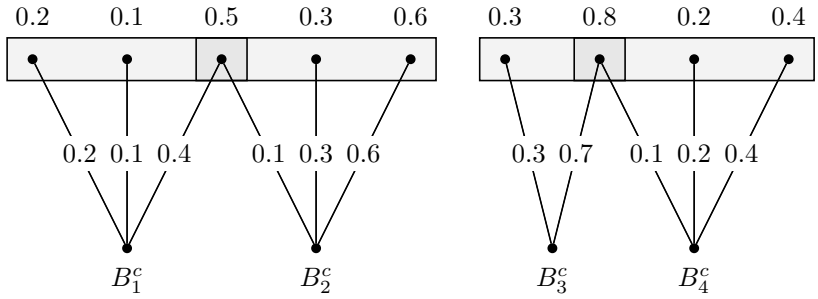



Figure 3.11: The construction of blocks $B_j^c$. The $x_{i,c}$ values appear on top and the $y_{i,(c,j)}$ values appear on the edges. Note that a block can only overlap with its predecessor or successor. In this case $\alpha_c = 0.7$.

For each $i \in B_j^c$ we define a variable $y_{i,(c,j)}$. If $i$ belongs to a single block $B_j^c$ of color $c$, then we set $y_{i,(c,j)} = x_{i,c}$. Otherwise, $i$ belongs to two adjacent blocks $B_{j+1}^c$ and $B_j^c$, in which case we set $y_{i,(c,j+1)} = \sum_{t \le i} x_{t,c} - (j - 1 + \alpha_c)$ and $y_{i,(c,j)} = x_{i,c} - y_{i,(c,j+1)}$. See Figure 3.11 for an example of how the blocks and the solution $y$ are constructed. Another, equivalent, way to define $y$ is to ask that $x_{i,c} = \sum_j y_{i,(c,j)}$, $\sum_{i \in B_1^c} y_{i,(c,1)} = \alpha_c$ and $\sum_{i \in B_j^c} y_{i,(c,j)} = 1$ for every $1 < j < b_c$. Thus $y$ defines a feasible fractional assignment between $[n]$ and the set of blocks. Let $\mathcal{Q}$ be the polytope of this assignment problem, namely,

$$\sum_{B_j^c \ni i} y_{i,(c,j)} = 1, \qquad \forall\, i \in [n], \tag{3.28}$$

$$\sum_{i \in B_j^c} y_{i,(c,j)} = 1, \qquad \forall\, c \in [k] \text{ and } 1 < j < b_c, \tag{3.29}$$

$$\sum_{i \in B_j^c} y_{i,(c,j)} \le 1, \qquad \forall\, c \in [k] \text{ and } j \in \{1, b_c\}, \tag{3.30}$$

$$y_{i,(c,t)} \ge 0, \qquad \forall\, i \in [n], c \in [k], t \in [b_c]. \tag{3.31}$$

Because $\mathcal{Q}$ is integral, any fractional solution $y \in \mathcal{Q}$ can be turned into an integral solution $\hat{y} \in \mathcal{Q}$; this can even be done in polynomial time. Notice that an integral solution $\hat{y}$ to $\mathcal{Q}$ induces an integral solution $\hat{x}$ by setting $\hat{x}_{i,c} = 1$ if and only if $y_{i,(c,j)} = 1$. Constraint (3.28) implies that $\hat{x}$ satisfies (*P1*). Furthermore, $\hat{x}$ also satisfies (*P2*).

**Lemma 3.1.** *Let $\hat{y}$ be an integral solution for $\mathcal{Q}$ and let $\hat{x}$ be the coloring induced by $\hat{y}$. Then $|\sum_{i \in I} \hat{x}_{i,c} - r(I,c)| \le 1$ for all $I \in \mathcal{I}$ and $c \in [k]$.*

*Proof.* Since $\sum_{i \in I} x_{i,c} = r(I,c)$, the number of blocks of color $c$ that intersect $I$ is either $r(I,c)$ or $r(I,c)+1$. Furthermore, at least $r(I,c)-1$ of these blocks lie entirely within $I$, and at most two blocks intersect $I$ partially. Due to constraints (3.28) and (3.29), each internal block will force a different position in $I$ to be colored $c$. On the other hand, the fringe blocks, if any, can force at most two additional positions in $I$ to be colored $c$. Hence, the lemma follows. □

It only remains to prove that $\hat{x}$ obeys ($P3$). To do so, we need to introduce some randomization in our construction. First, we will choose the offset $\alpha_c$ of each color $c \in [k]$ independently and uniformly at random. Second, instead of choosing any extreme point of $\mathcal{Q}$, we choose one using a randomized rounding procedure.

Gandhi *et al.* [GKPS06] showed that any fractional solution $y \in \mathcal{Q}$ can be rounded to an integral solution $\hat{y} \in \mathcal{Q}$ such that the probability that $\hat{y}_{i,(c,j)} = 1$ is exactly $y_{i,(c,j)}$. It is important to note that these events *are not independent* of each other.

**Lemma 3.2.** *Let $\hat{y}$ be the solution output by the randomized rounding procedure and $\hat{x}$ the coloring induced by it. For any interval $I \in \mathcal{I}$, the probability that $\sum_{i \in I} \hat{x}_{i,c} = r(I,c)$ for all $c \in [k]$ is at least $\frac{k(k+1-H_{k-1})}{(k+1)!}$.*

*Proof.* Let $I$ be an arbitrary, but fixed, interval throughout the proof and for the time being let us concentrate on a fixed, but arbitrary, color $c \in [k]$. Let $f$ and $l$ be the indices of the first and last blocks of color class $c$ that intersect $I$ and define $\beta_c = \sum_{i \in I \cap B_f^c} y_{i,(c,f)}$, or, equivalently, $\sum_{i \in I \cap B_l^c} y_{i,(c,l)} = 1 - \beta_c$.

Intuitively, the probability that $\sum_{i \in I} \hat{x}_{i,c} = r(I,c)$ should be greater when the blocks of $c$ are aligned with $I$ (when $\beta_c$ is close to 0 or 1) and it should be low when they are not (when $\beta_c$ is around 0.5). By choosing $\alpha_c$ uniformly at random, $\beta_c$ also becomes a random variable uniformly distributed in $[0,1]$. Thus, we have a decent chance of getting a "good value" of $\beta_c$.

Let us formalize and make more precise the above idea. Denote with $\xi_f$ and $\xi_l$ the events $\sum_{i \in I \cap B_f^c} \hat{y}_{i,(c,f)} = 1$ and $\sum_{i \in I \cap B_l^c} \hat{y}_{i,(c,l)} = 1$ respectively. Let $\beta = (\beta_1, \ldots, \beta_k)$ be the vector of offset for the color classes. For brevity, we denote $\Pr[\xi \mid \beta]$ with $\Pr_\beta[\xi]$.

$$\begin{aligned}
\Pr_\beta\left[\textstyle\sum_{i \in I} \hat{x}_{i,c} \neq r(I,c)\right] &= \Pr_\beta\left[\xi_f \xi_l \vee \overline{\xi_f}\,\overline{\xi_l}\right] \\
&= \Pr_\beta[\xi_f \xi_l] + \Pr_\beta\left[\overline{\xi_f}\,\overline{\xi_l}\right] \\
&\leq \min\{\Pr_\beta[\xi_f], \Pr_\beta[\xi_l]\} + \min\{\Pr_\beta\left[\overline{\xi_f}\right], \Pr_\beta\left[\overline{\xi_l}\right]\}.
\end{aligned}$$

Since $\Pr_\beta[\xi_f] = \beta_c$ and $\Pr_\beta[\xi_l] = 1 - \beta_c$, it follows that

$$\Pr_\beta\left[\textstyle\sum_{i \in I} \hat{x}_{i,c} \neq r(I,c)\right] \leq 2 \min\{\beta_c, 1 - \beta_c\}. \tag{3.32}$$

We first show that the probability that all requirements for $I$ are fulfilled is at least $\frac{1}{(k+1)!}$. Denote with $\tau$ the event $\forall c : \sum_{i \in I} \hat{x}_{i,c} = r(I,c)$. Recall that the vector $\beta$ is distributed uniformly over the domain $D = [0,1]^k$. Conditioning on $\beta$ and averaging over $D$ gives the desired result.

$$\begin{aligned}
\Pr[\tau] &= \int_D \Pr_\beta\left[\forall c : \textstyle\sum_{i\in I} \hat{x}_{i,c} = r(I,c)\right] \mathrm{d}\beta_1 \cdots \mathrm{d}\beta_k \\
&\geq \int_D 1 - \textstyle\sum_{c\in[k]} \Pr_\beta\left[\sum_{i\in I} \hat{x}_{i,c} \neq r(I,c)\right] \mathrm{d}\beta_1 \cdots \mathrm{d}\beta_k \\
&\geq \int_D \max\left\{0, 1 - 2\textstyle\sum_{c\in[k]} \min\{\beta_c, 1-\beta_c\}\right\} \mathrm{d}\beta_1 \cdots \mathrm{d}\beta_k \\
&= 2\int_D \max\left\{0, \tfrac{1}{2} - \textstyle\sum_{c\in[k]} \min\{\beta_c, 1-\beta_c\}\right\} \mathrm{d}\beta_1 \cdots \mathrm{d}\beta_k.
\end{aligned}$$

The second inequality follows from the union bound and the third follows from (3.32). A moment's thought reveals that the function inside the integral is symmetrical in the $2^k$ orthants around the point $(\frac{1}{2}, \ldots, \frac{1}{2}) \in D$. Therefore, setting $D' = [0, \frac{1}{2}]^k$ we get

$$\Pr[\tau] \geq 2^{k+1} \int_{D'} \max\left\{0, \tfrac{1}{2} - \textstyle\sum_{c\in[k]} \beta_c\right\} \mathrm{d}\beta_1 \cdots \mathrm{d}\beta_k.$$

The right hand side of the above inequality can be interpreted as the volume of a $(k+1)$-dimensional simplex.

$$\begin{aligned}
\Pr[\tau] &\geq 2^{k+1} \operatorname{Vol}\left(\lambda \in R_+^{k+1} \;\middle|\; \textstyle\sum_{i\in[k+1]} \lambda_i \leq \frac{1}{2}\right) \\
&= 2^{k+1} \frac{(\frac{1}{2})^{k+1}}{(k+1)!} \\
&= \frac{1}{(k+1)!}.
\end{aligned}$$

In order to get the stronger bound in the statement of the lemma, we need several improvements. First, we claim that we only need to condition on fulfilling $k-1$ requirements: Since $\sum_{c\in[k]} r(I,c) = |I|$, once we get $k-1$ colors right, the $k$th requirement must be satisfied as well. Second, since we can condition on any $k-1$ colors, we had better condition on the ones with smallest offset, that is, those that are close to 0 or 1.

$$\begin{aligned}
\Pr[\tau] &= \int_D \Pr_\beta\left[\forall c : \textstyle\sum_{i\in I} \hat{x}_{i,c} = r(I,c)\right] \mathrm{d}\beta_1 \cdots \mathrm{d}\beta_k \\
&\geq \int_D \max_{d\in[k]} \left\{1 - \textstyle\sum_{c\neq d} \Pr_\beta\left[\sum_{i\in I} \hat{x}_{i,c} \neq r(I,c)\right]\right\} \mathrm{d}\beta_1 \cdots \mathrm{d}\beta_k \\
&\geq \int_D \max_{d\in[k]} \left\{\max\left\{0, 1 - 2\textstyle\sum_{c\neq d} \min\{\beta_c, 1-\beta_c\}\right\}\right\} \mathrm{d}\beta_1 \cdots \mathrm{d}\beta_k \\
&= 2^k \int_{D'} \max_{d\in[k]} \left\{\max\left\{0, 1 - 2\textstyle\sum_{c\neq d} \beta_c\right\}\right\} \mathrm{d}\beta_1 \cdots \mathrm{d}\beta_k \\
&= 2^{k+1} \int_{D'} \max\left\{0, \tfrac{1}{2} - \textstyle\sum_{c\in[k]} \beta_c + \max_{d\in[k]} \beta_d\right\} \mathrm{d}\beta_1 \cdots \mathrm{d}\beta_k.
\end{aligned}$$

The last integral can be simplified by assuming that the maximum $\beta_d$ is attained by the last variable. Of course, the maximum can be any of the $k$ variables, thus these two quantities are related by a factor of $k$.

$$\Pr[\tau] \geq k\,2^{k+1} \int_0^{\frac{1}{2}} \left[ \int_{[0,z]^{k-1}} \max\left\{0, \tfrac{1}{2} - \textstyle\sum_{c\in[k-1]} \beta_c\right\} \mathrm{d}\beta_1 \cdots \mathrm{d}\beta_{k-1} \right] \mathrm{d}z.$$

Let $T(z)$ denote $\mathrm{Vol}\left(\lambda \in R_+^k \;\middle|\; \sum_{i=1}^k \lambda_i \leq \frac{1}{2} \text{ and } \lambda_1, \ldots, \lambda_{k-1} \leq z\right)$. Then we can rewrite the above integral as

$$\Pr[\tau] \geq k\,2^{k+1} \int_0^{\frac{1}{2}} T(z)\ \mathrm{d}z. \tag{3.33}$$

The volume computed by $T(z)$ is not a simplex, but it can be reduced to a summation involving only the volume of simplices using the *principle of inclusion-exclusion*.

Let $V(\rho)$ denote the volume $\mathrm{Vol}\left(\lambda \in R_+^k \;\middle|\; \sum_{i=1}^k \lambda_i \leq \rho\right)$ and recall that $V(\rho) = \frac{\rho^k}{k!}$. Consider what happens when $z \in \{x \in \mathbb{R} \mid \frac{1}{4} \leq x < \frac{1}{2}\}$; clearly $T(z) < V(\frac{1}{2})$ since $V(\frac{1}{2})$ includes points $\lambda \in \mathbb{R}_+^k$ such that $\lambda_i > z$ for exactly one coordinate $i \in [k-1]$ (since $z \geq \frac{1}{4}$). Notice that

$$\mathrm{Vol}\left(\lambda \in R_+^k \;\middle|\; \textstyle\sum_{i=1}^k \lambda_i \leq \frac{1}{2} \text{ and } \lambda_i > z\right) = V(\tfrac{1}{2} - z).$$

Thus $T(z) = V(\frac{1}{2}) - (k-1)V(\frac{1}{2} - z)$ for $z \in \{x \in \mathbb{R} \mid \frac{1}{4} \leq x \leq \frac{1}{2}\}$, but $T(z) > V(\frac{1}{2}) - (k-1)V(\frac{1}{2} - z)$ for $z \in \{x \in \mathbb{R} \mid 0 \leq x < \frac{1}{4}\}$ since the volume of points $y$ such the constraint $\lambda_i \leq z$ is violated for two coordinates is subtracted twice. To avoid cumbersome notation, assume $V(\rho) = 0$ if $\rho \leq 0$. A simple inclusion-exclusion argument yields

$$T(z) = \sum_{i=0}^{k-1} \binom{k-1}{i} (-1)^i\ V(\tfrac{1}{2} - iz). \tag{3.34}$$

Plugging (3.34) into (3.33) we get

$$\begin{aligned}
\Pr[\tau] &\geq 2^{k+1}k\left(\int_0^{\frac{1}{2}} V(\tfrac{1}{2})\,\mathrm{d}z + \sum_{i=1}^{k-1}\binom{k-1}{i}(-1)^i \int_0^{\frac{1}{2i}} V(\tfrac{1}{2}-iz)\,\mathrm{d}z\right)\\
&= 2^{k+1}k\left(\int_0^{\frac{1}{2}} \frac{(\frac{1}{2})^k}{k!}\,\mathrm{d}z + \sum_{i=1}^{k-1}\binom{k-1}{i}(-1)^i \int_0^{\frac{1}{2i}} \frac{(\frac{1}{2}-iz)^k}{k!}\,\mathrm{d}z\right)\\
&= 2^{k+1}k\left(\frac{1}{k!2^k}z\Big|_0^{\frac{1}{2}} + \sum_{i=1}^{k-1}\binom{k-1}{i}(-1)^i \frac{(\frac{1}{2}-iz)^{(k+1)}}{(k+1)!(-i)}\Big|_0^{\frac{1}{2i}}\right)\\
&= 2^{k+1}k\left(\frac{1}{k!2^{(k+1)}} + \sum_{i=1}^{k-1}\binom{k-1}{i}\frac{(-1)^i}{(k+1)!\,2^{k+1}\,i}\right)\\
&= \frac{k}{(k+1)!}\left(k+1+\sum_{i=1}^{k-1}\binom{k-1}{i}\frac{(-1)^i}{i}\right).
\end{aligned}$$

Using induction on $k$, it is straightforward to show that the sum in the last line adds up exactly to $-H_{k-1}$, where $H_{k-1}$ is the $(k-1)$th *harmonic number*. This gives us the desired bound of $\gamma_k$.

□

## 3.8 A Quasi-PTAS for IntervalColoring$_{\max}$

In this section, we describe how to compute an approximate solution to the problem IntervalColoring$_{\max}$ in quasi-polynomial time. Let $\textsc{Opt} \subseteq \mathcal{I}$ be a maximum weight subset of intervals for which a feasbile coloring exists. For $\alpha \in (0,1]$ and $\beta \geq 1$, an $(\alpha,\beta)$-approximation of the problem is given by a pair $(\chi, \mathcal{I}')$ of a subset $\mathcal{I}' \subseteq \mathcal{I}$, and a coloring $\chi : V \mapsto [k]$, such that $\sum_{I \in \mathcal{I}'} w(I) \geq \alpha \cdot w(\textsc{Opt})$, and $\frac{1}{\beta} r(I,c) \leq N_\chi(I,c) \leq \beta r(I,c)$, where $N_\chi(I,c)$ is the number of positions in $I$ colored $c$ by $\chi$, see (3.2).

**Theorem 3.2.** *Consider an instance $(V, \mathcal{I}, k, r)$ of* IntervalColoring$_{\max}$ *with $|V| = n$ and $|\mathcal{I}| = m$. Then we can find a $(1, 1+\epsilon)$-approximation in time $n^{O(\frac{k^2}{\epsilon} \log n \log m)}$, for any $\epsilon > 0$.*

Note that the above bound is quasi-polynomial for $k = \text{polylog}(n,m)$. To prove Theorem 3.2 we use a similar technique as in [ESZ07]. Our approach is based on two parts: (i) reducing the search space to *$\epsilon$-partial assignments* (to be defined), an effective abstraction of colorings that behave similarly with respect to quality of approximation and (ii) a *divide-and-conquer* (D&C) algorithm that finds the best $\epsilon$-partial assignments whose intersection represents a nonempty set of colorings. We explain these two steps in more detail in the next subsections.

### 3.8.1 Reducing the Search Space

Let $\epsilon > 0$ be a given constant. For a vertex $u \in V$ and a set of intervals $\mathcal{I}$ on $V$, denote respectively by $\mathcal{I}_L(u)$, $\mathcal{I}_R(u)$ and $\mathcal{I}(u)$, the subsets of intervals of $\mathcal{I}$ that lie to the left of $u$, lie to the right of $u$, and span $u$, that is

$$\begin{aligned} \mathcal{I}_L(u) &= \{[s,t] \in \mathcal{I} : \ t \leq u-1\}, \\ \mathcal{I}_R(u) &= \{[s,t] \in \mathcal{I} : \ s \geq u+1\}, \\ \mathcal{I}(u) &= \{[s,t] \in \mathcal{I} : \ s \leq u \leq t\}. \end{aligned}$$

Denote by $V_L(u)$ and $V_R(u)$ the sets of vertices that lie to the left and right of $u \in V$, respectively: $V_L(u) = \{i \in V : \ i < u\}$ and $V_R(u) = \{i \in V : \ i \geq u\}$.

**Definition 3.2** (Assignment)**.** *Let $V' = \{p, p+1, \ldots, q\}$. An* assignment *on $V'$ is a pair $\mathscr{A} = (\mathcal{I}, r)$ of intervals $\mathcal{I}$ on $V'$ and a function $r : \mathcal{I} \times [k] \mapsto \{0, 1, \ldots, |V'|\}$ such that*

*(C1) $r(I,c) \leq r(I',c)$ for all $I, I' \in \mathcal{I}$ with $I \subseteq I'$ and all $c \in [k]$, and*

*(C2) $\sum_{c \in [k]} r(I,c) = |I|$ for every $I \in \mathcal{I}$.*

*$\mathscr{A}$ is called a* left-assignment *(respectively,* right-assignment*) if all intervals in $\mathcal{I}$ start at $p$ (respectively, end at $q$).*

**Definition 3.3** ($\epsilon$-Partial Assignment)**.** *Let $u^* \in V'$ be a given vertex of $V' = \{p, p+1, \ldots, q\}$. A set of $h_1 + h_2 + 2$ intervals $\mathcal{I} = \mathcal{I}_L \cup \mathcal{I}_R$, $\mathcal{I}_L = \{I_1, \ldots, I_{h_1}, I_{h_1+1}\}$ and $\mathcal{I}_R = \{I'_1, \ldots, I'_{h_2}, I'_{h_2+1}\}$, together with a function $r : \mathcal{I} \times [k] \mapsto \{0, 1, \ldots, |V'|\}$, such that*

*(R1) all intervals start or end at $u^*$: $I_j = [u_j, u^*]$ for $j \in \{1, 2, \ldots, h_1\}$, $I_{h_1+1} = [p, u^*]$, $I'_j = [u^*, u'_j]$ for $j \in \{1, 2, \ldots, h_2\}$, and $I'_{h_2+1} = [u^*, q]$, where $u_{h_1} < u_{h_1-1} < \cdots < u_1 < u^* < u'_1 < u'_2 < \cdots < u'_{h_2}$,*

*(R2) $(\mathcal{I}_L, r)$ is a right-assignment on $\{p, \ldots, u^*\}$, and $(\mathcal{I}_R, r)$ is a left-assignment on $\{u^*, \ldots, q\}$,*

*(R3) for every $I \in \mathcal{I} \setminus \{I_{h_1+1}, I'_{h_2+1}\}$ there exists a color $c \in [k]$ and an integer $i \in \mathbb{Z}_+$ such that $r(I, c) = \lceil (1+\epsilon)^i \rceil$, and*

*(R4) for every $c \in [k]$ and $i \in \mathbb{Z}_+$ with $i \leq \lfloor (\log r(I_{h_1+1}, c)/\log(1+\epsilon) \rfloor$, there exists $I \in \mathcal{I}_L$ such that $r(I, c) = \lceil (1+\epsilon)^i \rceil$; likewise, for every $c \in [k]$ and $i \in \mathbb{Z}_+$ with $i \leq \lfloor (\log r(I'_{h_2+1}, c)/\log(1+\epsilon) \rfloor$, there exists $I' \in \mathcal{I}_R$ such that $r(I', c) = \lceil (1+\epsilon)^i \rceil$.*

*will be called an* $\epsilon$-partial assignment *with respect to $u^*$, denoted by $\mathscr{P} = (u^*, \mathcal{I}, r)$.*

From property (*C2*) of an assignment (with $|I| \leq n$) and property (*R3*) of an $\epsilon$-partial assignment it follows $h_1, h_2 \leq \lceil k \log n / \log(1+\epsilon) - 1 \rceil$. In Figure 3.12 vertices $\{p, p+1, \ldots, u^*\} \subseteq V'$ are shown along with four intervals from $\mathcal{I}_L$, all ending at $u^*$ (*R1*). Note that intervals $I_{j_1}$, $I_{j_2}$ and $I_{j_h}$ satisfy condition (*R4*) for color $c \in [k]$, since $r(I_{j_1}, c) = \lceil (1+\epsilon)^1 \rceil$, $r(I_{j_2}, c) = \lceil (1+\epsilon)^2 \rceil$ and $r(I_{j_h}, c) = \lceil (1+\epsilon)^h \rceil$, for $h = \lceil (\log r(I_{h_1+1}, c)/\log(1+\epsilon) - 1 \rceil$.

The total number $\mu(n)$ of possible $\epsilon$-partial assignments with respect to a given vertex $u^* \in V$ with $|V| = n$ can be bounded as follows: There are at most $n^{h_1+h_2}$ possible choices for the vertices $u_1, u_2, \ldots, u_{h_1}, u'_1, u'_2, \ldots, u'_{h_2}$. For each interval $I \in \mathcal{I}$, the number of nonnegative integer requirements $r(I, c)$, $c \in [k]$, satisfying (*C2*) is

$$\begin{aligned}
\binom{|I|+k-1}{k-1} &= \prod_{i=1}^{k-1} \left(1 + \frac{|I|}{i}\right) \\
&\leq \left(\frac{1}{k-1} \sum_{i=1}^{k-1} \left(1 + \frac{|I|}{i}\right)\right)^{k-1} \\
&= \left(1 + \frac{|I|}{k-1} H_{k-1}\right)^{k-1} \\
&= \left(1 + \frac{|I|}{k-1}(1 + \ln(k-1))\right)^{k-1},
\end{aligned}$$

where $H_{k-1}$ is the $(k-1)$th *harmonic number*. Letting $i_1 = |I_1|$, $i_j = |I_j \setminus I_{j-1}|$, for $2 \leq j \leq h_1 + 1$, and similarly $i'_1 = |I'_1| - 1$, $i'_j = |I'_j \setminus I'_{j-1}|$, for $2 \leq j \leq h_2 + 1$, and $\zeta := (k-1)(h_1 + h_2 + 2)$, we observe by (*C1*) and (*C2*) in Definition 3.2 that

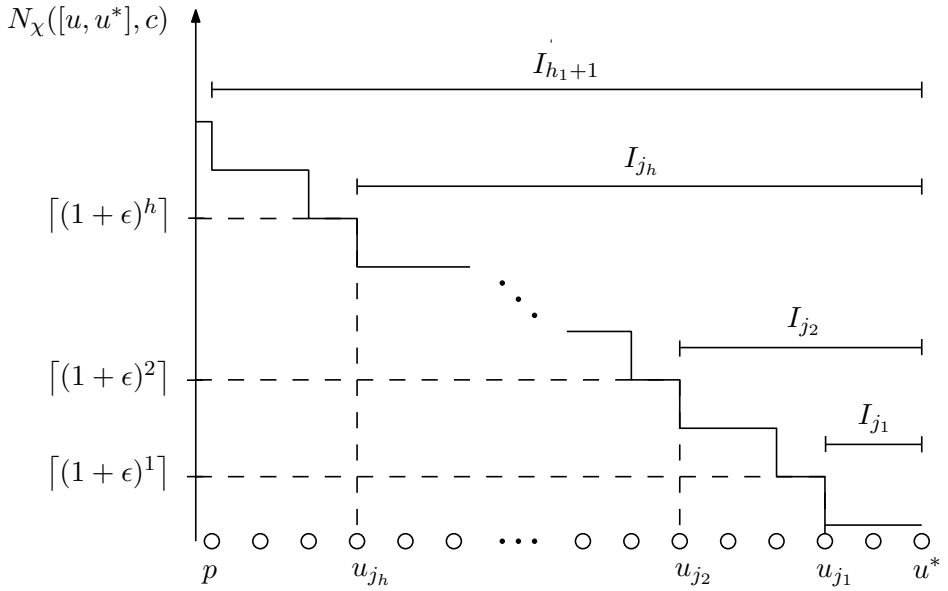



Figure 3.12: The number of vertices in interval $[u,u^*]$ colored $c \in [k]$ by $\chi$ is monotonically increasing on $u-p$. According to $(R4)$, an $\epsilon$-partial assignment consistent with $\chi$ has to contain intervals $I_{j_1}$, $I_{j_2}$ and $I_{j_h}$ with $r(I_{j_1},c) = \lceil(1+\epsilon)^1\rceil$, $r(I_{j_2},c) = \lceil(1+\epsilon)^2\rceil$ and $r(I_{j_h},c) = \lceil(1+\epsilon)^h\rceil$, for $h = \lceil(\log r(I_{h_1+1},c)/\log(1+\epsilon) - 1\rceil$. Interval $I_{h_1+1} = [p,u^*]$, see $(R1)$.

$$
\begin{aligned}
\mu(n) &\leq n^{h_1+h_2} \prod_{j=1}^{h_1+1} \binom{i_j+k-1}{k-1} \prod_{j=1}^{h_2+1} \binom{i'_j+k-1}{k-1} \\
&\leq n^{h_1+h_2} \prod_{j=1}^{h_1+1} \left(1+\frac{i_j}{k-1}(\ln k+1)\right)^{k-1} \prod_{j=1}^{h_2+1} \left(1+\frac{i'_j}{k-1}(\ln k+1)\right)^{k-1} \\
&\leq n^{h_1+h_2} \left(\frac{\sum_{j=1}^{h_1+1}\left(1+\frac{i_j}{k-1}(\ln k+1)\right) + \sum_{j=1}^{h_2+1}\left(1+\frac{i'_j}{k-1}(\ln k+1)\right)}{h_1+h_2+2}\right)^{\zeta} \\
&= n^{h_1+h_2} \left(1+\frac{\ln k+1}{k-1}\cdot\frac{n}{h_1+h_2+2}\right)^{(k-1)(h_1+h_2+2)} \\
&\leq n^{2k^2\frac{\log n}{\log(1+\epsilon)}+4k-2} \cdot \left(2\cdot\frac{\ln k+1}{k-1}\right)^{(k-1)\left(2k\frac{\log n}{\log(1+\epsilon)}+4\right)}, \qquad (3.35)
\end{aligned}
$$

which is $n^{\text{polylog}(n)}$ for every fixed $\epsilon > 0$ and $k = \text{polylog}(n)$. Note that in the last step of the calculation the $+1$ summand in big brackets can be replaced by a multiplicative factor of 2 for sufficiently large $n$.

**Definition 3.4** (Consistent Assignment)**.** *Let $\chi : V \mapsto [k]$ be a coloring of $V$. We say that an assignment $\mathscr{A} = (\mathcal{I}, r)$ is* consistent *with $\chi$ if $N_\chi(I, c) = r(I, c)$ for all $c \in [k]$ and $I \in \mathcal{I}$. Two assignments $\mathscr{A}_1$ and $\mathscr{A}_2$ are said to be* consistent *if there exists a coloring $\chi$ with which both are consistent.*

**Lemma 3.3.** *Let $\chi$ be a coloring of $V'$ and $u^* \in V'$ be an arbitrary vertex. Then there exists an $\epsilon$-partial assignment $\mathscr{P} = (u^*, \mathcal{I}, r)$ on $V'$ with respect to $u^*$ that is consistent with $\chi$.*

*Proof.* Assume that $V' = \{p, p+1, \ldots, q\}$. Clearly, for every $c \in [k]$ the function $N_\chi([u^*, u], c)$ is monotonically increasing on $u \geq u^*$ with a smallest positive increment of 1. This allows us to define $\mathscr{P}$ as follows. Let $u'_0 = u^*$. For $j = 1, 2, \ldots, h_2$ let

$$u'_j = \min\Big\{u > u'_{j-1} \mid \exists\, i \in \mathbb{Z}_+, c \in [k] :\ N_\chi([u^*, u], c) = \big\lceil (1+\epsilon)^i \big\rceil \Big\}. \tag{3.36}$$

The highest index $j$ for which such an $u'_j < q$ exists determines the value of $h_2$. In accordance with condition *(R1)*, we set $I'_j = [u^*, u'_j]$ for $j = 1, 2, \ldots, h_2$ and $I_{h_2+1} = [u^*, q]$. In a similar way, we define $h_1$ and the intervals $I_j$ for $j = 1, 2, \ldots, h_1 + 1$ (see Figure 3.12). Finally, we define $r(I, c) = N_\chi(I, c)$ for all $c \in [k]$ and $I \in \mathcal{I}$, which naturally satisfies *(C1)* and *(C2)*. The definition of interval endpoints according to (3.36) guarantees *(R3)* and *(R4)*. □

We observe that an $\epsilon$-partial assignment $\mathscr{P}$ is an effective abstraction of the set of colorings that $\mathscr{P}$ is consistent with:

**Observation 3.1.** *Let $\mathscr{P} = (u^*, \mathcal{I}_\mathscr{P}, r_\mathscr{P})$ be an $\epsilon$-partial assignment on $V$. Given an interval $I = [s, t] \in \mathcal{I}$ of the problem instance with $u^* \in I$, we let $j(I, \mathscr{P})$ and $\ell(I, \mathscr{P})$ be, respectively, the smallest and largest indices such that $[u_{j(I,\mathscr{P})}, u'_{\ell(I,\mathscr{P})}] \subseteq I$, i.e. $j(I, \mathscr{P}) = \min\{i :\ u_i \geq s\}$ and $\ell(I, \mathscr{P}) = \max\{i :\ u'_i \leq t\}$ (see Figure 3.13). If either of these indices does not exist, we set the corresponding value of $r_\mathscr{P}(I_{\ell(I,\mathscr{P})}, c)$ or $r_\mathscr{P}(I_{j(I,\mathscr{P})}, c)$ to 0. Then by property (R4) of an $\epsilon$-partial assignment*

$$r_\mathscr{P}(I_{\ell(I,\mathscr{P})}, c) + r_\mathscr{P}(I_{j(I,\mathscr{P})}, c) \leq N_{\chi'}(I, c) \leq (1+\epsilon)\big(r_\mathscr{P}(I_{\ell(I,\mathscr{P})}, c) + r_\mathscr{P}(I_{j(I,\mathscr{P})}, c)\big) \tag{3.37}$$

*holds for any $c \in [k]$ and coloring $\chi' : V \mapsto [k]$ such that $\mathscr{P}$ is consistent with $\chi'$.*

In the next section we show how to compute an assignment that represents $(1, 1+\epsilon)$-approximate colorings by recursively merging consistent $\epsilon$-partial assignments.

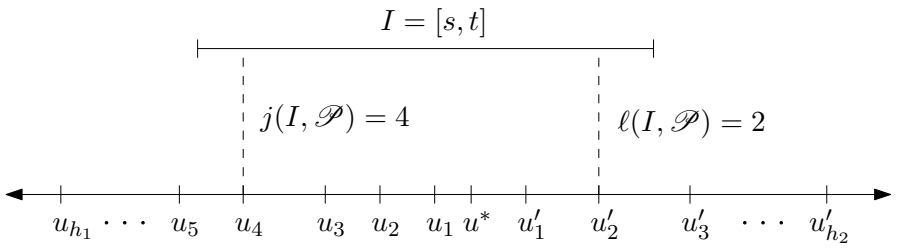



Figure 3.13: For an $\epsilon$-partial assignment $\mathscr{P}$ with respect to $u^*$ and a given interval $I \in \mathcal{I}$ of the problem instance, $j(I, \mathscr{P})$ and $\ell(I, \mathscr{P})$ are defined to be the smallest and largest indices, respectively, such that $[u_{j(I,\mathscr{P})}, u'_{\ell(I,\mathscr{P})}] \subseteq I$. The $u_i$ and $u'_j$ are as described in $(R1)$ in Definition 3.3 of an $\epsilon$-partial assignment.

### 3.8.2 A Divide-and-Conquer Algorithm

The pseudocode describing our divide-and-conquer (D&C) algorithm is presented below as a procedure called MaxColApprox, which takes as parameters an instance $(V, \mathcal{I}, k, r)$ of problem IntervalColoring$_{\max}$ and consistent left- and right-assignments $\mathscr{L}$ and $\mathscr{R}$. To compute an $(1, 1+\epsilon)$-approximation, we set $\mathscr{L}$ and $\mathscr{R}$ to be empty in the initial call. To simplify notation, we omit parameters $k$ and $r$ in the signature of procedure MaxColApprox as they are assigned identical values in all recursive calls.

Following the D&C paradigm, the algorithm picks a vertex $u^*$ in the middle of $V$ and evaluates all intervals containing $u^*$ to determine whether they should be taken into the solution. To do this evaluation conservatively, the procedure iterates over all $\epsilon$-partial assignments $\mathscr{P}$ with respect to the middle vertex $u^*$ that are consistent with $\mathscr{L}$ and $\mathscr{R}$, then recurses on the subsets of intervals to the left and right of $u^*$.

Procedure MaxColApprox uses two subroutines: MaxColSpecial checks whether a pair of a left- and a right-assignment is consistent, and if so, returns a feasible coloring; Reduce$(V_L(u^*), \mathscr{P}, \mathscr{L}, \mathscr{R})$ $\big($Reduce$(V_R(u^*), \mathscr{P}, \mathscr{L}, \mathscr{R})\big)$ combines the assignments $\mathscr{P}, \mathscr{L}$ and $\mathscr{R}$ into left- and right-assignments $\mathscr{L}', \mathscr{R}'$ on $V_L(u^*)$ $\big($respectively, on $V_R(u^*)\big)$. For a more detailed description of the two subroutines see below.

From the recursive calls in lines **7** and **8** we obtain two independent colorings $\chi_1 : V_L(u^*) \mapsto [k]$ and $\chi_2 : V_R(u^*) \mapsto [k]$, which are combined in line **9** into a coloring $\chi = \chi_1 \cup \chi_2$ defined in the obvious way: $\chi(u) = \chi_1(u)$ if $u \in V_L(u^*)$ and $\chi(u) = \chi_2(u)$ if $u \in V_R(u^*)$.

Given a left-assignment $\mathscr{L} = (\mathcal{I}_{\mathscr{L}}, r_{\mathscr{L}})$, a right-assignment $\mathscr{R} = (\mathcal{I}_{\mathscr{R}}, r_{\mathscr{R}})$, and an $\epsilon$-partial assignment $\mathscr{P} = (u^*, \mathcal{I}_L \cup \mathcal{I}_R, r_{\mathscr{P}})$ on a vertex set $V' = \{p, \ldots, q\}$, procedure Reduce constructs, considering the recursive call on $V'_L(u^*)$ in line **7**, a left-assignment $\mathscr{L}' = (\mathcal{I}_{\mathscr{L}'}, r_{\mathscr{L}'})$ and right-assignment $\mathscr{R}' = (\mathcal{I}_{\mathscr{R}'}, r_{\mathscr{R}'})$ on vertex set $\{p, \ldots, u^*\}$ by cutting intervals at $u^*$ as follows (see Figure 3.14):

**Algorithm 2**: MaxColApprox($V, \mathcal{I}, \mathscr{L}, \mathscr{R}$)

**Data**: An instance $(V, \mathcal{I}, k, r)$ of IntervalColoring$_{\max}$
**Result**: An $(1, 1+\epsilon)$-approximation $(\chi, \mathcal{J})$

```
1  if |I| = 0 then
2      χ ← MaxColSpecial(L, R)
3      return (χ, ∅)
4  let u* ∈ V be such that |I_L(u*)| ≤ m/2 and |I_R(u*)| ≤ m/2
5  forall ε-partial assignments P = (u*, I_P, r_P) do
6      if P is consistent with L and R then
7          (χ1, J1) ← MaxColApprox(V_L(u*), I_L(u*), Reduce(V_L(u*), P, L, R))
8          (χ2, J2) ← MaxColApprox(V_R(u*), I_R(u*), Reduce(V_R(u*), P, L, R))
9          χ ← χ1 ∪ χ2
10         K ← {I ∈ I(u*) : r(I,c)/(1+ε) ≤ r_P(I_ℓ(I,P), c) + r_P(I_j(I,P), c) ≤ r(I,c)  ∀ c ∈ [k]}
11         J ← K ∪ J1 ∪ J2
12         store (χ, J)
13 return the recorded solution with largest w(J) value
```

(a) $\mathcal{I}_{\mathscr{L}'} = \{[p,t] \in \mathcal{I}_{\mathscr{L}} \mid t \le u^*\}$,

(b) $\mathcal{I}_{\mathscr{R}'} = \mathcal{I}_L \cup \{[s,u^*] \mid \exists\, [s,q] \in \mathcal{I}_{\mathscr{R}} : s < u^*\}$,

(c) $r_{\mathscr{L}'}(I,c) = r_{\mathscr{L}}(I,c)$ for $I \in \mathcal{I}_{\mathscr{L}'}$ and $r_{\mathscr{R}'}(I,c) = r_{\mathscr{P}}(I,c)$ for $I \in \mathcal{I}_L$, $\forall\, c \in [k]$,

(d) $r_{\mathscr{R}'}([s,u^*],c) = r_{\mathscr{R}}([s,q],c) - r_{\mathscr{P}}([u^*,q],c) + 1$ for $[s,q] \in \mathcal{I}_{\mathscr{R}}$, $s < u^*$, $\forall\, c \in [k]$.

In the recursive call in line **8**, procedure Reduce combines the given assignments according to a symmetric schema. Notice that $\mathcal{I}_{\mathscr{L}} = \emptyset$ in the leftmost and $\mathcal{I}_{\mathscr{R}} = \emptyset$ in the rightmost path of the recursion tree.

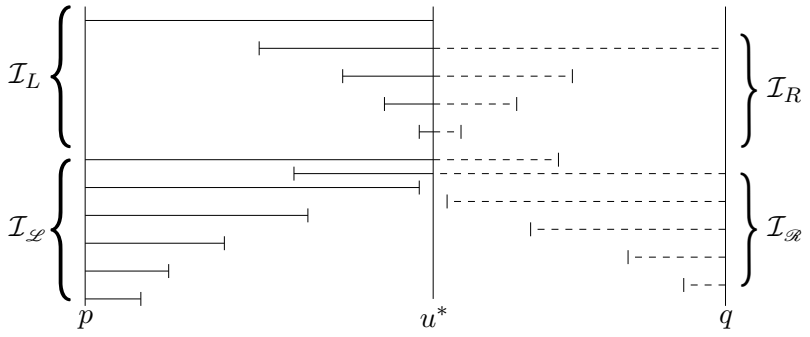



Figure 3.14: For given left-assignment $\mathscr{L} = (\mathcal{I}_{\mathscr{L}}, r_{\mathscr{L}})$, right-assignment $\mathscr{R} = (\mathcal{I}_{\mathscr{R}}, r_{\mathscr{R}})$ and an $\epsilon$-partial assignment $\mathscr{P} = (u^*, \mathcal{I}_L \cup \mathcal{I}_R, r_{\mathscr{P}})$, in the recursive call on $V_L(u^*)$ procedure Reduce cuts intervals on the vertical line at index $u^*$ such that the new left- and right-assignments $\mathscr{L}'$ and $\mathscr{R}'$ contain the intervals shown by solid lines. Interval $[p,q]$, contained both in $\mathcal{I}_{\mathscr{L}}$ and $\mathcal{I}_{\mathscr{R}}$, is omitted.

In the following Lemma 3.4 and Theorem 3.3 we show how procedure MaxColSpecial can check consistency of assignments $\mathscr{L} = (\mathcal{I}_{\mathscr{L}}, r_{\mathscr{L}})$ and $\mathscr{R} = (\mathcal{I}_{\mathscr{R}}, r_{\mathscr{R}})$ on vertex set $V$ in line **2** in time $\mathcal{O}(|\mathcal{I}_{\mathscr{L}}| + |\mathcal{I}_{\mathscr{R}}|)$. Note that sets $\mathcal{I}_{\mathscr{L}}$ and $\mathcal{I}_{\mathscr{R}}$ each contain an interval spanning all vertices in $V$. This is due to intervals $I_{h_1+1}$ and $I'_{h_2+1}$ in Definition 3.3 of an $\epsilon$-partial assignment and due to the specific structure of the assignments constructed by procedure Reduce (see Figure 3.14)

Following the terminology introduced in Section 3.1.2 we call a coloring $\chi$ *feasible* for an assignment $\mathscr{A} = (\mathcal{I}, r)$, if $N_\chi(I, c) = r(I, c)$ for all $c \in [k]$ and $I \in \mathcal{I}$. In other words, $\chi$ is feasible for $\mathscr{A}$, if $\mathscr{A}$ is consistent with $\chi$. We call two assignments $\mathscr{A}$ and $\mathscr{A}'$ *equivalent*, if a coloring $\chi$ is feasible for $\mathscr{A}$ if and only if $\chi$ is feasible for $\mathscr{A}'$.

**Lemma 3.4.** *Let $\mathscr{A} = (\mathcal{I}, r)$ be an assignment on $V = \{1, 2, \ldots, n\}$, where set $\mathcal{I}$ can be partitioned into two sets $\mathcal{I}_1$ and $\mathcal{I}_2$, such that for $p \in \{1, 2\}$ it holds*

*(P1) $I_i \cap I_j = \emptyset$, $\forall I_i, I_j \in \mathcal{I}_p$, i.e. intervals are disjoint and*

*(P2) $\bigcup_{I \in \mathcal{I}_p} I = [1, n]$, i.e. the intervals span all vertices.*

*Then it can be decided in time $\mathcal{O}(|\mathcal{I}|)$ whether a feasible coloring $\chi : V \mapsto [k]$ for $\mathscr{A}$ exists.*

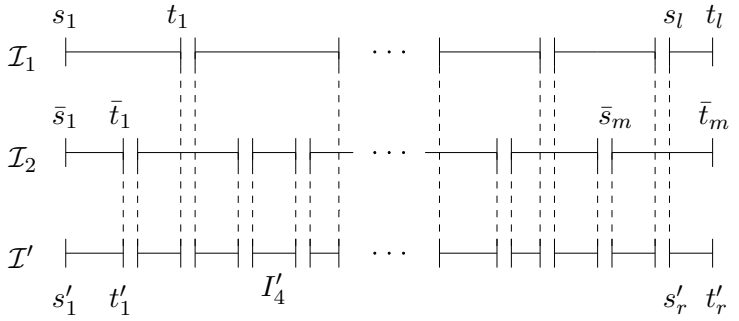
*Proof.* We represent interval set $\mathcal{I}_1$ as sequence $\big([s_1, t_1], [s_2, t_2], \ldots, [s_l, t_l]\big)$ and set $\mathcal{I}_2$ as sequence $\big([\bar{s}_1, \bar{t}_1], [\bar{s}_2, \bar{t}_2], \ldots, [\bar{s}_m, \bar{t}_m]\big)$, where $s_i = t_{i-1} + 1$ for $2 \leq i \leq l$, and similarly $\bar{s}_i = \bar{t}_{i-1} + 1$ for $2 \leq i \leq m$. Property (*P2*) implies $s_1, \bar{s}_1 = 1$ and $t_l, \bar{t}_m = n$. For $1 \leq i \leq l$, we denote $[s_i, t_i]$ by $I_i$, and for $1 \leq i \leq m$, we denote $[\bar{s}_i, \bar{t}_i]$ by $\bar{I}_i$.

From assignment $\mathscr{A}$ we construct an equivalent assignment $\mathscr{A}' = (\mathcal{I}', r')$, where intervals in $\mathcal{I}'$ are disjoint and therefore feasibility of $\mathscr{A}'$ can be determined by verifying for every interval $[s, t] \in \mathcal{I}'$ that

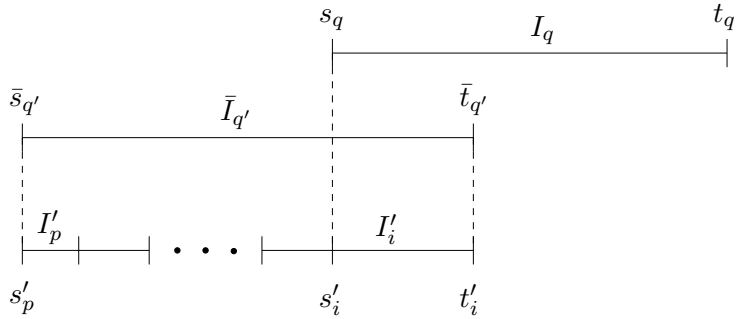
$$\sum_{c \in [k]} r'([s, t], c) = t - s + 1.$$

We define $\mathcal{I}'$ to be the partition of $\{1, 2, \ldots, n\}$ into a minimal number of intervals, such that for each interval $I' \in \mathcal{I}'$ and each element $I \in \mathcal{I}$ either $I' \subseteq I$ or $I' \cap I = \emptyset$ (see Figure 3.15(a)). We represent $\mathcal{I}'$ by sequence $\big([s'_1, t'_1], [s'_2, t'_2], \ldots, [s'_r, t'_r]\big)$ and again denote $[s'_i, t'_i]$ by $I'_i$ for $1 \leq i \leq r$.

What remains is the assignment of requirements to intervals in $\mathcal{I}'$, i.e. the definition of $r' : \mathcal{I}' \times [k] \mapsto \{1, 2, \ldots, n\}$. We will define function $r'$ recursively, i.e. for $c \in [k]$ the value $r'(I'_i, c)$ might depend on values $r'(I'_j, c)$ for $j < i$. Due to the minimality of $\mathcal{I}'$, $t'_1 = \min(t_1, \bar{t}_1)$ and interval $I'_1$ will coincide with either $I_1$ or $\bar{I}_1$. In Figure 3.15(a) the latter case holds. Therefore any coloring $\chi$ feasible for assignment $\mathscr{A}$ will satisfy $N_\chi(I'_1, c) = r'(I'_1, c)$, if and only if $r'(I'_1, c) = r(I_1, c)$, respectively $r'(I'_1, c) = r(\bar{I}_1, c)$, for all $c \in [k]$. Now consider an interval $I'_i$ for arbitrary $2 \leq i \leq r$. If $I'_i \in \mathcal{I}_1$ or $I'_i \in \mathcal{I}_2$, as e.g. $I'_4 \in \mathcal{I}_2$ in Figure 3.15(a), for assignment $\mathscr{A}'$ to be equivalent with assignment




(a) Definition of intervals




(b) Definition of coloring requirements

Figure 3.15: (a) Set $\mathcal{I}_1$ and $\mathcal{I}_2$ satisfy (*P1*) and (*P2*) in Lemma 3.4. For each interval $I' \in \mathcal{I}'$ and each element $I \in \mathcal{I}$, $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2$, either $I' \subseteq I$ or $I' \cap I = \emptyset$. (b) In the construction of an equivalent assignment $\mathscr{A}'$ in the proof of Lemma 3.4 the number of vertices that have to be colored $c$ in interval $I'_i$ is obtained by equation (3.38).

$\mathscr{A}$ it must hold $r'(I'_i, c) = r(I'_i, c)$, for all $c \in [k]$. Otherwise, without loss of generality assume $s'_i = s_q$ for some $I_q \in \mathcal{I}_1$ and $t'_i = \bar{t}_{q'}$ for some $\bar{I}_{q'} \in \mathcal{I}_2$. Let $p$ be such that $I'_p \in \mathcal{I}'$ and $s'_p = \bar{s}_{q'}$ (see Figure 3.15(b)). If we assume that any coloring $\chi$ feasible for $\mathscr{A}$ satisfies $N_\chi(I'_j, c) = r'(I'_j, c)$ for all intervals $I'_j$ with $1 \leq j \leq i-1$, then $\chi$ will satisfy $N_\chi(I'_i, c) = r'(I'_i, c)$ if and only if

$$r'(I'_i, c) = r(\bar{I}_{q'}, c) - \sum_{j=p}^{i-1} r'(I'_j, c), \text{ for all } c \in [k]. \tag{3.38}$$

□

Clearly the above lemma can be generalized to the case where $\mathcal{I}$ can be partitioned into an arbitrary number of sets, each satisfying conditions (*P1*) and (*P2*).

**Theorem 3.3.** *Let $V = \{1, 2, \ldots, n\}$. For given left assignment $\mathscr{L} = (\mathcal{I}_{\mathscr{L}}, r_{\mathscr{L}})$ with $[1, n] \in \mathcal{I}_{\mathscr{L}}$ and right assignment $\mathscr{R} = (\mathcal{I}_{\mathscr{R}}, r_{\mathscr{R}})$ with $[1, n] \in \mathcal{I}_{\mathscr{R}}$, it can be decided in time $\mathcal{O}(|\mathcal{I}_{\mathscr{L}}| + |\mathcal{I}_{\mathscr{R}}|)$ whether $\mathscr{L}$ and $\mathscr{R}$ are consistent.*

*Proof.* Let set $\mathcal{I}_{\mathscr{L}} = \big([1, t_1], [1, t_2], \ldots, [1, t_p]\big)$ with $t_p = n$ and $\mathcal{I}_{\mathscr{R}} = \big([s_1, n], [s_2, n], \ldots, [s_q, n]\big)$ with $s_1 = 1$ be sorted with respect to "$\subseteq$" and "$\supseteq$", respectively, in non-

decreasing order. Then assignments $\mathscr{L}$ and $\mathscr{R}$ are consistent if and only if the following assignments $\mathscr{L}' = (\mathcal{I}_{\mathscr{L}'}, r_{\mathscr{L}'})$ (see Figure 3.16) and $\mathscr{R}' = (\mathcal{I}_{\mathscr{R}'}, r_{\mathscr{R}'})$ are consistent:

(a) $\mathcal{I}_{\mathscr{L}'} = \big([1, t_1], [t_1 + 1, t_2], \ldots, [t_{p-1} + 1, t_p]\big)$,

(b) $r_{\mathscr{L}'}([1, t_1], c) = r_{\mathscr{L}}([1, t_1], c)$ and $r_{\mathscr{L}'}([t_{i-1} + 1, t_i], c) = r_{\mathscr{L}}([1, t_i], c) - r_{\mathscr{L}}([1, t_{i-1}], c)$, for $2 \leq i \leq p$ and $c \in [k]$.

(c) $\mathcal{I}_{\mathscr{R}'} = \big([s_1, s_2 - 1], [s_2, s_3 - 1], \ldots, [s_q, n]\big)$.

(d) $r_{\mathscr{R}'}([s_q, n], c) = r_{\mathscr{R}}([s_q, n], c)$ and $r_{\mathscr{R}'}([s_i, s_{i+1} - 1], c) = r_{\mathscr{R}}([s_i, n], c) - r_{\mathscr{R}}([s_{i+1}, n], c)$, for $1 \leq i < q$ and $c \in [k]$.

Interval sets $\mathcal{I}_{\mathscr{L}'}$ and $\mathcal{I}_{\mathscr{R}'}$ satisfy conditions $(P1)$ and $(P2)$ in Lemma 3.4 and therefore the claim follows. □

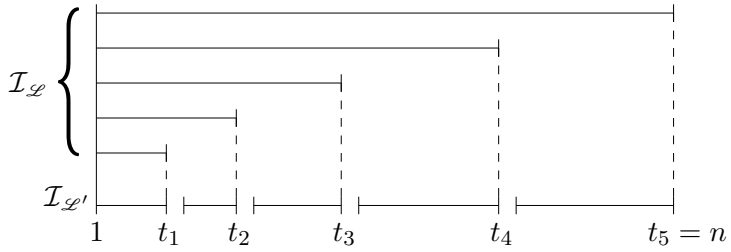



Figure 3.16: Left-assignment $\mathscr{L} = (\mathcal{I}_{\mathscr{L}}, r_{\mathscr{L}})$ can be transformed into an equivalent assignment $\mathscr{L}' = (\mathcal{I}_{\mathscr{L}'}, r_{\mathscr{L}'})$. For every interval in $\mathcal{I}_{\mathscr{L}'} \setminus \{[1, t_1]\}$ its requirement $r_{\mathscr{L}'}$ is equal to the difference between the requirements $r_{\mathscr{L}}$ of its defining intervals in $\mathcal{I}_{\mathscr{L}}$ (see proof of Theorem 3.3).

Since intervals in $\mathcal{I}'$ of assignment $\mathscr{A}'$ in the proof of Lemma 3.4 are disjoint, procedure MaxColSpecial can determine coloring $\chi$ in line **2** on vertices in each interval $I' \in \mathcal{I}'$ independently, respecting only $N_\chi(I', c) = r'(I', c)$ for all colors $c \in [k]$. Therefore procedure MaxColSpecial runs in time $\mathcal{O}(|V|)$.

In line **6** of procedure MaxColApprox, consistency of an $\epsilon$-partial assignment $\mathscr{P} = (u^*, \mathcal{I}_L \cup \mathcal{I}_R, r_{\mathscr{P}})$ and left- and right-assignments $\mathscr{L}$ and $\mathscr{R}$ has to be verified. From the definition of an $\epsilon$-partial assignment (see Definition 3.3) it follows that $(\mathcal{I}_L, r_{\mathscr{P}})$ forms a right-assignment on $V_L(u^*)$ and $(\mathcal{I}_R, r_{\mathscr{P}})$ forms a left-assignment on $V_R(u^*)$, where every vertex is spanned by at least one interval. As such, similar as in the proof of Theorem 3.3, they can be transformed into equivalent assignments containing only disjoint intervals. As intervals in $\mathcal{I}_L$ and $\mathcal{I}_R$ only intersect in $u^*$, this transformation results in a single set of intervals $\widetilde{\mathcal{I}}$ satisfying conditions $(P1)$ and $(P2)$ in Lemma 3.4. As shown above in the description of procedure MaxColSpecial, checking consistency of assignments $\mathscr{L}$ and $\mathscr{R}$ can be reduced to a feasibility problem of an assignment $\mathscr{A}' = (\mathcal{I}', r')$ (see proof of Lemma 3.4), where $\mathcal{I}'$ itself satisfies $(P1)$ and $(P2)$ in Lemma 3.4. In summary, consistency of $\mathscr{P}$, $\mathscr{L}$ and $\mathscr{R}$ can be verified in line **6** by applying Lemma 3.4 to sets $\widetilde{\mathcal{I}}$

and $\mathcal{I}'$ in time $\mathcal{O}\left(|\widetilde{\mathcal{I}}| + |\mathcal{I}'|\right)$, which is, since intervals in the respective sets are disjoint, $\mathcal{O}(|V|)$.

**Theorem 3.4.** *Given an instance $(V, \mathcal{I}, k, r)$ of* IntervalColoring$_{\max}$ *with $|V| = n$ and $|\mathcal{I}| = m$, algorithm* MaxColApprox *runs in time $T(n,m) = n^{O(\frac{k^2}{\epsilon}\log n \log m)}$.*

*Proof.* The number of possible $\epsilon$-partial assignments is at most $\mu(n)$, bounded in (3.35). This gives the recurrence

$$T(n,m) \leq \text{poly}(n,m) + 2\mu(n) \cdot T\left(\frac{m}{2}\right).$$

The theorem follows by the master theorem [CLRS01]. □

**Theorem 3.5.** *Given an instance $(V, \mathcal{I}, k, r)$ of* IntervalColoring$_{\max}$*, algorithm* MaxColApprox *returns a coloring $\chi : V \mapsto [k]$ and a subset of intervals $\mathcal{J} \subseteq \mathcal{I}$ such that $w(\mathcal{J}) \geq w(\text{Opt})$ and for all $I \in \mathcal{J}$ and $c \in [k]$*

$$\frac{r(I,c)}{(1+\epsilon)} \;\leq\; N_\chi(I,c) \;\leq\; (1+\epsilon)\, r(I,c).$$

*Proof.* Let the pair $(\chi^*, \text{Opt})$ be an optimal solution to instance $(V, \mathcal{I}, k, r)$ of problem IntervalColoring$_{\max}$. By Lemma 3.3, there is an $\epsilon$-partial assignment $\mathscr{P}$ consistent with $\chi^*$, which will be eventually considered by the algorithm in line **5**. If $I \in \text{Opt}(u^*)$, then $N_{\chi^*}(I,c) = r(I,c)$ for all $c \in [k]$ and thus (3.37) implies, for $\chi' = \chi^*$, that $I$ belongs to the set $\mathcal{K}$ selected by the algorithm in line **10**, i.e. $\text{Opt}(u^*) \subseteq \mathcal{K}$, and hence $w(\mathcal{K}) \geq w\big(\text{Opt}(u^*)\big)$. Since $\mathscr{P}$ is consistent with the coloring $\chi$ obtained in line **9**, we also know, by using $\chi' = \chi$ in (3.37), that

$$\frac{r(I,c)}{(1+\epsilon)} \;\leq\; N_\chi(I,c) \;\leq\; (1+\epsilon)\, r(I,c) \text{ for } I \in \mathcal{K}.$$

By induction, we have $w(\mathcal{J}_1) \geq w\big(\text{Opt}_L(u^*)\big)$ and $w(\mathcal{J}_2) \geq w\big(\text{Opt}_R(u^*)\big)$. Furthermore, we know that

$$\frac{r(I,c)}{(1+\epsilon)} \;\leq\; N_{\chi_1}(I,c) \leq\; (1+\epsilon)\, r(I,c) \text{ for } I \in \mathcal{J}_1$$

and

$$\frac{r(I,c)}{(1+\epsilon)} \;\leq\; N_{\chi_2}(I,c) \;\leq\; (1+\epsilon)\, r(I,c) \text{ for } I \in \mathcal{J}_2.$$

The theorem follows. □

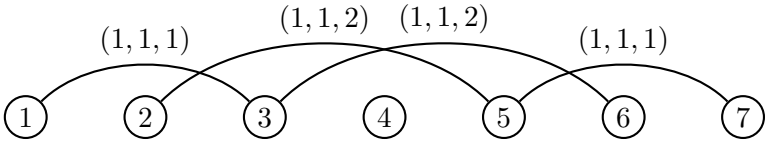



Figure 3.17: An interval coloring instance with $|V| = 7$, $|\mathcal{I}| = 4$, and $k = 3$. For each interval $I$, the coloring requirements are denoted by a triplet $\big(r(I,1), r(I,2), r(I,3)\big)$. Then a fractional vertex of the polytope obtained by relaxing the integrality constraint in (3.5) is given by $\mathbf{x}_1 = \big(\frac{1}{2}, 0, \frac{1}{2}, 0, \frac{1}{2}, 0, \frac{1}{2}\big)$, $\mathbf{x}_2 = \big(\frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}\big)$, $\mathbf{x}_3 = \big(0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0\big)$.

## 3.9 Hardness

Since the constraint matrix of the integer linear program (3.17) capturing the 2-color case is totally unimodular, the corresponding polytope $\mathcal{P}$ is integral for $k = 2$. For $k \geq 3$, the total unimodularity of the constraint matrix is destroyed by constraints (3.7) in the basic-BIP and by constraints (3.14) in the improved-ILP, i.e. there are instances with fractional vertices (see Figure 3.17 for an example).

The complexity status of the *interval constraint coloring* problem for three or more colors remains open. If the length of the intervals is bounded by a constant, an optimal coloring can be computed in polynomial time by a dynamic programming algorithm. However, in this section we show that deciding whether a feasible coloring exists is $\mathcal{NP}$-complete when $k$ is part of the input. Moreover, we show that the maximization variant $\textsc{IntervalColoring}_{\max}$ with $k = 2$ colors is $\mathcal{APX}$-hard and hence does not admit a PTAS (unless $\mathcal{P} = \mathcal{NP}$), even if the intervals are required to intersect in at least one residue.

### 3.9.1 $\mathcal{NP}$-Completeness of IntervalColoring

**Theorem 3.6.** *The problem of testing the feasibility of an instance of the interval constraint coloring problem is $\mathcal{NP}$-complete when the number of colors is part of the input.*

*Proof.* Clearly, the problem belongs to $\mathcal{NP}$. To prove that the problem is $\mathcal{NP}$-hard, we reduce a known $\mathcal{NP}$-hard problem to it using the approach of Chang *et al.* [CEGK08]. In the *exact coverage* problem, we are given a ground set $\mathcal{U}$ and a collection $\mathcal{S}$ of subsets of $\mathcal{U}$, and we want to know whether there exists a subcollection $\mathcal{C} \subseteq \mathcal{S}$ of size $t$, which forms a partition of $\mathcal{U}$; that is, $\bigcup_{A \in \mathcal{C}} A = \mathcal{U}$ and for any two elements $A, B \in \mathcal{C}$ if $A \neq B$ then $A \cap B = \emptyset$. It is well known that the exact coverage problem is $\mathcal{NP}$-complete [GJ79a] even when the cardinality of sets in $\mathcal{S}$ is 3.

Let $u = |\mathcal{U}|$ and $s = |\mathcal{S}|$. We map the exact coverage instance to an instance of the coloring problem, by dividing $V = [n]$ into $u$ blocks $B_1, \ldots, B_u$ each of length $s$; thus, $n = us$ and $B_i = [(i-1)s + 1,\, i\,s]$. Each color $c \in [k]$ is associated with a specific set $S_c$

in $\mathcal{S}$; therefore, $k = s$. Let $\mathcal{U} = \{x_1, \ldots, x_u\}$ and suppose that $x_i$ is contained in $r_i$ sets. For every $i \in [u]$ we have

$$I_i = [s\,(i-1)+1,\ s\,i] \quad \text{and} \quad r(I_i, c) = 1 \text{ for all } c \in [k],$$
$$I_i'' = [s\,i - t - r_i + 2,\ s\,i - t + 1] \quad \text{and} \quad r(I_i'', c) = 1 \text{ if and only if } x_i \in S_c,$$

and for every $i \in [u-1]$ we have

$$I_i' = [s\,i - t + 1,\ s\,(i+1) - t] \quad \text{and} \quad r(I_i', c) = 1 \text{ for all } c \in [k].$$

Realize that any coloring satisfying all the $I_i$ and $I_i'$ intervals must use the same set of $t$ colors for the last $t$ vertices of every block and the remaining $s-t$ colors for the first $s-t$ vertices of every block. We therefore encode the partition $\mathcal{C}$ with the last $t$ colors of each block (see Figure 3.18). To enforce $\mathcal{C}$ to be a partition, i.e. every element $x_i \in \mathcal{U}$ to be contained in *exactly* one set of $\mathcal{C}$, we include the interval $I_i'' = [s\,i - t - r_i + 2, s\,i - t + 1]$ and require $r(I_i'', c) = 1$ if and only if $x_i \in S_c$. Clearly, a feasible coloring encodes a solution for the exact coverage problem and vice versa. It follows that testing feasibility is $\mathcal{NP}$-hard. □

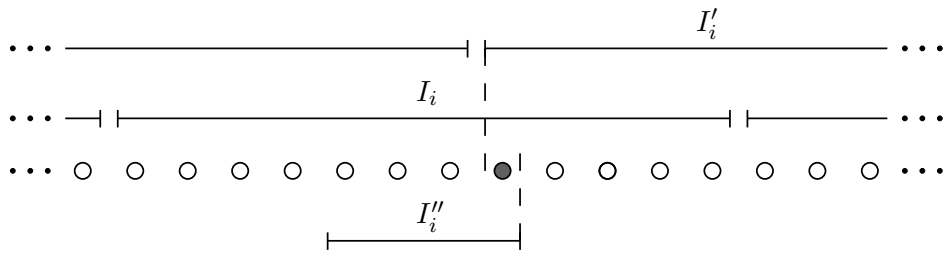



Figure 3.18: Interval $I_i$ spans the vertices of block $B_i$. The colors assigned to the $t = 5$ vertices in the intersection of intervals $I_i$ and $I_i'$ encode the partition $\mathcal{C}$. The color of the single vertex in the intersection of intervals $I_i'$ and $I_i''$ (denoted by the filled vertex) will correspond to the unique set in $\mathcal{C}$ that contains $x_i$.

### 3.9.2 $\mathcal{APX}$-Hardness of IntervalColoring$_{\max}$

In this section, we show that IntervalColoring$_{\max}$ with $k = 2$ colors is $\mathcal{APX}$-hard. The reduction is from Max2SAT, whose input is a boolean formula $\varphi$ in conjunctive normal form in which each clause is a disjunction of 2 literals (2CNF). Literals are variables and their negations taken from a ground set of boolean variables $X = \{x_1, x_2, \ldots, x_n\}$. The optimization problem asks for the maximum number of clauses which can be satisfied by any truth assignment to variables in $X$. It is known that Max2SAT is $\mathcal{APX}$-hard [Hås97].

In other words, Max2SAT is $\mathcal{NP}$-hard to approximate to within a certain factor, i.e., its approximability exhibits a certain gap. We propose a gap preserving reduction $\tau$ from Max2SAT to IntervalColoring$_{\max}$ with $k = 2$. The construction part of our

reduction uses some ideas from [ERRS08] where the authors show $\mathcal{APX}$-hardness for the *maximum feasible subsystem* problem where the constraint matrix is a clique.[1] In particular, we use *variable gadgets* to locally replace variables of a boolean formula $\varphi$ by a set of intervals of an IntervalColoring$_{\max}$ instance $\tau(\varphi)$ and *clause gadgets* to translate the clauses of a Max2SAT instance $\varphi$ into intervals of $\tau(\varphi)$ that intersect intervals of the corresponding variable gadgets in an appropriate way.

In the following, we describe how we construct from a 2CNF formula $\varphi$ containing $m$ clauses an instance $\tau(\varphi)$ containing $14m$ intervals.

### Variable Gadgets

Each variable $x_i$ in the boolean formula $\varphi$ is represented by a collection of 3 intervals in instance $\tau(\varphi)$, defined on a vertex set $V$, as depicted in Figure 3.19. Intervals $I_a$ and $I_b$ span the same set of vertices and require 4 vertices more to be colored in color 1, respectively in color 2, than in the other color. Interval $I_c$ symmetrically extends intervals $I_a$ and $I_b$ by a total of 4 vertices and establishes parity of vertices colored 1 and 2. More precisely, if interval $I_a$ ($I_b$) spans vertices $\{p, p+1, \ldots, q\}$ with $q - p + 1 = 4(2i-1)$, then $I_c$ spans vertices $\{p-2, \ldots, q+2\}$.

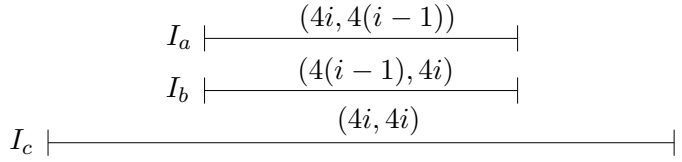



Figure 3.19: The variable gadget for variable $x_i$ consists of 3 intervals. Associated with each interval $I$ is a tuple $(x, y)$ that defines the requirement function on $I$ by $x = r(I, 1)$ and $y = r(I, 2)$. The requirement is a function of the index of the corresponding variable.

**Lemma 3.5.** *There are exactly two optimal solutions to* IntervalColoring$_{\max}$ *for a variable gadget, both consisting of two intervals.*

*Proof.* Since intervals $I_a$ and $I_b$ span an identical set of vertices but impose different requirements, there exists no coloring of $V$ satisfying (3.1) for both intervals $I_a$ and $I_b$. At the same time, any coloring of vertices spanned by $I_a$ ($I_b$) that satisfies (3.1) for either $I_a$ or $I_b$ can be extended to a coloring of vertices spanned by interval $I_c$ that satisfies (3.1) for $I_c$. □

[1] The *maximum feasible subsystem* problem is defined as follows. Given a linear system $c \leq Ax \leq b$, we want to find the largest system of inequalities that can be simultaneously satisfied. In [ERRS08], the authors show that even when the matrix $A$ is a 0/1 matrix with the consecutive ones property, such that the underlying graph is a clique, the problem remains $\mathcal{APX}$-hard.

**Definition 3.5** (TRUE/FALSE state)**.** *For the optimal solution satisfying* (3.1) *for intervals $I_a$ and $I_c$ we call the variable gadget to be in* TRUE *state, and for the solution satisfying* (3.1) *for intervals $I_b$ and $I_c$ to be in* FALSE *state.*

In the following, we denote the number of clauses variable $x_i$ appears in by $m_i$, and interval $I_a$ ($I_b$, $I_c$) of the gadget representing variable $x_i$ by $I_a^i$ (respectively $I_b^i$, $I_c^i$). To simplify terminology, we call an interval satisfied if really equation (3.1) is satisfied for that interval.

When putting the variable gadgets together, each gadget is replicated $2m_i$ times, that is, there are a total of $2m_i$ gadgets of the type shown in Figure 3.19 representing $x_i$, where each replicated interval spans the same set of vertices. For every $j > i$, intervals in the gadgets for variable $x_j$ symmetrically extend intervals in the gadgets for variable $x_i$, i.e. if $I_c^i$ spans vertices $\{p, p+1, \ldots, q\}$, then $I_c^j$ spans vertices $\{p-4(j-i), \ldots, q+4(j-i)\}$. In the illustration in Figure 3.20 the replicated gadgets are omitted.

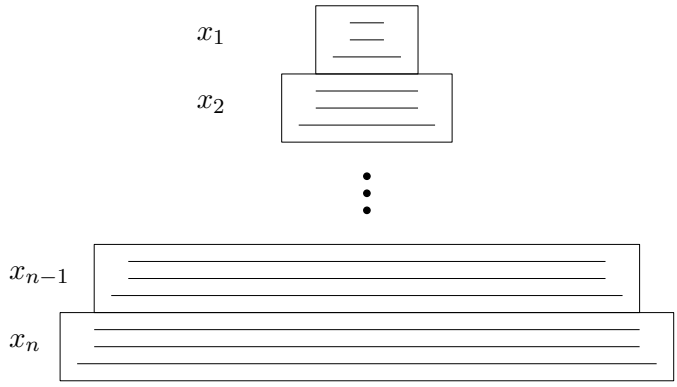



Figure 3.20: Putting variable gadgets together. Replicated gadgets are omitted.

Let a coloring $\chi : V \mapsto \{1, 2\}$ of vertices $V = \{1, 2, \ldots, n\}$ with two colors be denoted by string $\chi(1)\chi(2)\ldots\chi(n)$. Then a coloring for which the gadget representing variable $x_1$ is in TRUE or FALSE state is given by string 22111122, respectively 11222211. Now assume that gadgets for variables $x_1, \ldots, x_{i-1}$ are in TRUE or FALSE state under some coloring $\chi$. It can be easily verified that a coloring $\chi'$ extending $\chi$ to the 8 vertices spanned by interval $I_c^i$ but not by interval $I_c^{i-1}$ has to be of the form $2211\ldots1122$ or $1122\ldots2211$ to cause the gadget representing variable $x_i$ to be in TRUE or FALSE state, respectively.

The next lemma shows that the state of the variable gadgets can indeed be chosen independently of each other.

**Lemma 3.6.** *Let the ground set of boolean variables be $X = \{x_1, x_2, \ldots, x_n\}$. Then for every boolean assignment $\nu : X \mapsto \{true, false\}$ there exists a coloring $\chi$ such that for all $1 \leq i \leq n$ the variable gadget representing $x_i$ is in* TRUE *state if $\nu(x_i) = true$ and in* FALSE *state if $\nu(x_i) = false$.*

*Proof.* We proof the claim of the lemma by induction on the index of the variables. To variable $x_1$ Lemma 3.5 directly applies, and we color the vertices spanned by $I_c^1$ such that the state of the variable gadget reflects the value of $x_1$. Now assume variable gadgets representing $x_1, \ldots, x_{i-1}$ are all in TRUE or FALSE state, in accordance with the boolean assignment to the variables. From the definition of a TRUE/FALSE state we know that interval $I_c^{i-1}$ is satisfied. Therefore, we can satisfy interval $I_a^i$ or $I_b^i$ by coloring the four vertices spanned by $I_a^i$ and $I_b^i$ but not by $I_c^{i-1}$ all in color 1 or all in color 2, respectively. In accordance with the proof of Lemma 3.5, we can extend $\chi$ such that it satisfies interval $I_c^i$ at the same time. □

### Clause Gadgets

What remains is the definition of gadgets that translate the clauses of a MAX2SAT instance containing exactly two literals into a set of constraints of an instance of problem INTERVALCOLORING$_{\max}$, i.e. into intervals imposing certain coloring requirements. The four different types of clauses involving variables $x_i$ and $x_j$, namely $(x_i \vee x_j)$, $(\neg x_i \vee x_j)$, $(x_i \vee \neg x_j)$ and $(\neg x_i \vee \neg x_j)$, are represented by the clause gadgets shown in Figures 3.21(a)-3.21(d).

Each clause gadget is made up of two intervals $I_\alpha$ and $I_\beta$ that span the same set of vertices. Since they impose different coloring requirements, they cannot be satisfied at the same time. The intuitive idea behind the clause gadgets is that in any optimal solution to $\tau(\varphi)$ the variable gadgets will be in either TRUE or FALSE state and thus result in a vertex coloring which would satisfy exactly one of the intervals $I_\alpha$ or $I_\beta$ if and only if the corresponding clause is satisfied. This is shown in the next lemma.

**Lemma 3.7.** *Let $C$ be a clause involving variables $x_i$ and $x_j$. A coloring $\chi$ that causes variable gadgets representing $x_i$ and $x_j$ to be either in* TRUE *or* FALSE *state satisfies exactly one interval of the clause gadget corresponding to $C$ if clause $C$ is satisfied by the same boolean assignment. Otherwise none of the intervals of the clause gadget can be satisfied.*

*Proof.* The proof of the lemma follows immediately from Tables 3.22(a)-3.22(d). They show, for the four different types of clauses and all possible truth assignments, the number of vertices colored 1 and 2 in intervals $I_\alpha$ and $I_\beta$, respectively. The last column shows that only if the clause is not satisfied under the given assignment, neither $I_\alpha$ nor $I_\beta$ is satisfied. In all other cases, exactly one interval is satisfied. □

The following theorem captures the central statement of this section. For a 2CNF formula $\varphi$ we let MAX2SAT($\varphi$) denote the maximum *fraction* of clauses that is satisfied by any truth assignment. INTERVALCOLORING$_{\max}\big(\tau(\varphi)\big)$ is the maximum number of intervals in instance $\tau(\varphi)$ that can be satisfied by any vertex coloring.

**Theorem 3.7.** *There is a fixed $\epsilon > 0$ such that* INTERVALCOLORING$_{\max}$ *is $\mathcal{NP}$-hard to approximate to within a multiplicative factor $(1+\epsilon)$ unless $\mathcal{P} = \mathcal{NP}$.*

*Proof.* We show that reduction $\tau$ from MAX2SAT to INTERVALCOLORING$_{\max}$ described above is gap-preserving. From the fact that MAX2SAT is MAX-$\mathcal{SNP}$-hard [PY88] it follows that there exist $\epsilon, \delta > 0$ such that given an 2CNF $\varphi$ containing $m$ clauses, it is $\mathcal{NP}$-hard to distinguish the cases when MAX2SAT$(\varphi) \geq (1-\epsilon)m$ and when MAX2SAT$(\varphi) < (1-\epsilon-\delta)m$.

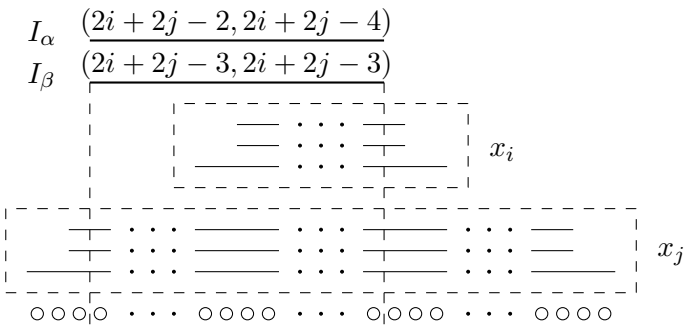
So let $\varphi$ be a 2CNF formula that contains $m$ clauses. From a truth assignment $\nu$ that satisfies $k$ clauses we construct a coloring $\chi$ for $\tau(\varphi)$ that satisfies $8m+k$ intervals as follows. We choose $\chi$ such that for all $1 \leq i \leq n$ variable gadget representing $x_i$ is in TRUE state if $\nu(x_i) = true$ and in FALSE state if $\nu(x_i) = false$ (see Lemma 3.6). From Definition 3.5 it follows that each variable gadget, including its $2m_i$ copies, contributes $2 \cdot 2m_i$ satisfied intervals. Furthermore, in each gadget corresponding to a satisfied clause in $\varphi$ exactly one interval is satisfied by $\chi$. So in total $\chi$ satisfies $4\sum_{i=1}^{n} m_i + k = 8m + k$ intervals.

Now consider a coloring for $\tau(\varphi)$ that maximizes the number of satisfied intervals. Under this coloring, each variable gadget is in either TRUE or FALSE state and thus contributes 2 satisfied intervals per copy. Otherwise, we lose for every interval in a variable gadget that we do not satisfy also its $2m_i - 1$ copies, that cannot be satisfied either in this case. Note that according to Lemma 3.5, no more than 2 intervals can be satisfied per variable gadget. At the same time, satisfying less than 2 intervals in a variable gadget for $x_i$ could lead to at most $m_i$ additional satisfied intervals in clause gadgets that involve $x_i$ (at most one interval can be satisfied per clause gadget). Thus in any optimal solution the variable gadgets and their copies contribute $8m$ satisfied intervals in total. From Lemma 3.7 it follows that $k$ satisfied intervals from clause gadgets in $\tau(\varphi)$ result in $k$ satisfied clauses in $\varphi$ under the corresponding truth assignment.

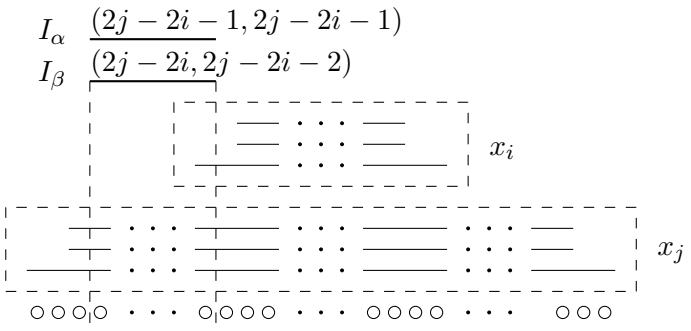
In other words, polynomial time reduction $\tau$ from MAX2SAT to INTERVALCOLORING$_{\max}$ ensures that, for every boolean formula $\varphi$:

$$\begin{aligned} \text{MAX-2SAT}(\varphi) \geq (1-\epsilon)m \quad &\Rightarrow \quad \text{INTERVALCOLORING}_{\max}\big(\tau(\varphi)\big) \geq (9-\epsilon)m \\ \text{MAX-2SAT}(\varphi) < (1-\epsilon-\delta)m \quad &\Rightarrow \quad \text{INTERVALCOLORING}_{\max}\big(\tau(\varphi)\big) < (9-\epsilon-\delta)m \end{aligned}$$

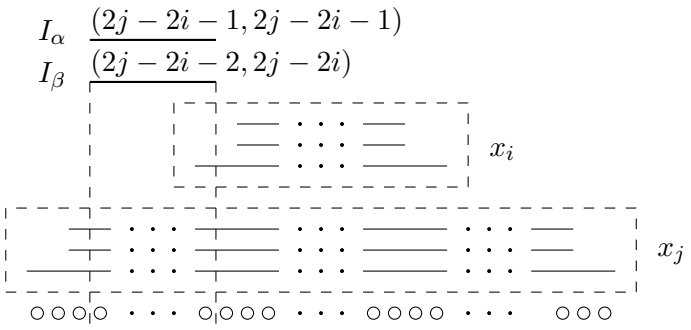
Since $\tau(\varphi)$ in general contains $\Theta(m)$ intervals the reduction is gap-preserving and the claim follows. □



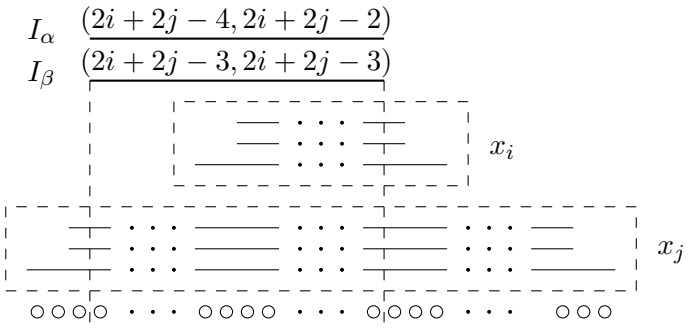

(a) Clause $(x_i \vee x_j)$




(b) Clause $(\neg x_i \vee x_j)$




(c) Clause $(x_i \vee \neg x_j)$




(d) Clause $(\neg x_i \vee \neg x_j)$

Figure 3.21: The clause gadgets encode the respective clauses.

| $x_i$ | $x_j$ | $x_i \vee x_j$ | $N_\chi(I_\alpha, 1)$ | $N_\chi(I_\alpha, 2)$ | Interval satisfied |
|---|---|---|---|---|---|
| $T$ | $T$ | $T$ | $2i + 2j - 2$ | $2i + 2j - 4$ | $I_\alpha$ |
| $T$ | $F$ | $T$ | $2i + 2j - 3$ | $2i + 2j - 3$ | $I_\beta$ |
| $F$ | $T$ | $T$ | $2i + 2j - 3$ | $2i + 2j - 3$ | $I_\beta$ |
| $F$ | $F$ | $F$ | $2i + 2j - 4$ | $2i + 2j - 2$ | – |

(a) Clause $(x_i \vee x_j)$

| $x_i$ | $x_j$ | $\neg x_i \vee \neg x_j$ | $N_\chi(I_\alpha, 1)$ | $N_\chi(I_\alpha, 2)$ | Interval satisfied |
|---|---|---|---|---|---|
| $T$ | $T$ | $F$ | $2i + 2j - 2$ | $2i + 2j - 4$ | – |
| $T$ | $F$ | $T$ | $2i + 2j - 3$ | $2i + 2j - 3$ | $I_\beta$ |
| $F$ | $T$ | $T$ | $2i + 2j - 3$ | $2i + 2j - 3$ | $I_\beta$ |
| $F$ | $F$ | $T$ | $2i + 2j - 4$ | $2i + 2j - 2$ | $I_\alpha$ |

(b) Clause $(\neg x_i \vee \neg x_j)$

| $x_i$ | $x_j$ | $\neg x_i \vee x_j$ | $N_\chi(I_\alpha, 1)$ | $N_\chi(I_\alpha, 2)$ | Interval satisfied |
|---|---|---|---|---|---|
| $T$ | $T$ | $T$ | $2j - 2i - 1$ | $2j - 2i - 1$ | $I_\alpha$ |
| $T$ | $F$ | $F$ | $2j - 2i - 2$ | $2j - 2i$ | – |
| $F$ | $T$ | $T$ | $2j - 2i$ | $2j - 2i - 2$ | $I_\beta$ |
| $F$ | $F$ | $T$ | $2j - 2i - 1$ | $2j - 2i - 1$ | $I_\alpha$ |

(c) Clause $(\neg x_i \vee x_j)$

| $x_i$ | $x_j$ | $x_i \vee \neg x_j$ | $N_\chi(I_\alpha, 1)$ | $N_\chi(I_\alpha, 2)$ | Interval satisfied |
|---|---|---|---|---|---|
| $T$ | $T$ | $T$ | $2j - 2i - 1$ | $2j - 2i - 1$ | $I_\alpha$ |
| $T$ | $F$ | $T$ | $2j - 2i - 2$ | $2j - 2i$ | $I_\beta$ |
| $F$ | $T$ | $F$ | $2j - 2i$ | $2j - 2i - 2$ | – |
| $F$ | $F$ | $T$ | $2j - 2i - 1$ | $2j - 2i - 1$ | $I_\alpha$ |

(d) Clause $(x_i \vee \neg x_j)$

Figure 3.22: Intervals satisfied in gadget corresponding to the respective clause. A $T$ in column $x_i$ ($x_j$) denotes $\nu(x_i) = true$ ($\nu(x_j) = true$), a $F$ in column $x_i$ ($x_j$) denotes $\nu(x_i) = false$ ($\nu(x_j) = false$).

## 3.10 Conclusion

We have introduced an ILP approach to assign exchange rates to individual residues based on aggregate data with respect to peptic fragments. For all instances provided by our collaborator from biochemistry, we were able to determine a coloring with minimum total error in less than 0.1 second. For the case of two different colors (exchange rates) we propose a combinatorial polynomial-time algorithm, by reducing the problem to a minimum cost network flow problem. It serves as a subroutine in a heuristic approach that improves upon the running time of the exact enumeration algorithm significantly while providing solutions, that are close to the optimum.

Furthermore, in a Lagrangian relaxation approach for the general case of an arbitrary number of colors, we dualize linking constraints in such a way, that the resulting Lagrangian subproblem becomes a set of independent two-color interval coloring instances. Exploiting the Lagrangian bounds, strengthened by the subgradient method, in a branch-and-bound algorithm yields exact (optimal) solutions to the general ($\mathcal{NP}$-complete) problem involving an arbitrary number of colors.

In our application domain, the goal usually is not to find a single solution, but to generate a number of candidate solutions and let the user choose the one that he finds most interesting or relevant for the specific application. It requires only a minor modification to the standard branch-and-bound (B&B) technique to be able to enumerate all possible solutions, independent of whether the bounds are based on LP relaxation or on Lagrangian relaxation. Experiments show however, that the efficiency of algorithms to solve a minimum cost network flow problem, compared to a general purpose algorithm for solving a linear programming relaxation, allows us to enumerate all optimal colorings considerably faster. Even the randomized rounding approach yielding a coloring that satisfies all coloring requirements within $\pm 1$ of the prescribed value is amenable to this task since there are very efficient algorithms to enumerate all the integral solutions of the assignment polytope [Uno01].

Moreover, both the B&B algorithms and the combinatorial approach can deal with noise in the experimental data; the randomized rounding method underlies the intrinsic limitation that the LP relaxation polytope must not be empty. Besides capturing the noise by additional variables and constraints in the ILP formulation, we proposed an alternative approach especially suited for the case of an empty LP relaxation polytope. Since we can show that finding the maximum number of intervals whose coloring requirements are satisfied is $\mathcal{APX}$-hard even in the two-color case, we introduced an approximation scheme that violates the requirements by a factor $(1 + \epsilon)$ and runs in quasi-polynomial time.

For a fixed number of colors greater or equal than three, the complexity of the interval constraint coloring problem remains open. The total unimodularity of the constraint matrix is destroyed, i.e. there are instances with fractional vertices. Moreover, the integrality gap is infinite, since there is an instance with strictly positive error but whose LP relaxation has optimal value 0.

Chapter 4

# Conclusion

The need for exact methods for $\mathcal{NP}$-hard problems in the biological context is often called into question. On the one hand, solving the combinatorial optimization problem is indeed not equivalent to answering the underlying biological question. It always defines a mathematical abstraction only of specific aspects of a biological process and as such captures the underlying biological problem at most to an extend, to which it is understood by biologists. The interval constraint coloring problem, for example, emphasizes on the aspect of solvent accessability to obtain information about the tertiary structure of a protein. Therefore, one might ask whether it is worth the computational effort to find a solution to highly restricted instances that is optimal only in a given model, but whose interpretation does not necessarily coincide completely with what is observed in the real world.

Nevertheless, exact methods are important in their own right. They might provide results, based on which heuristic approaches can be benchmarked, or even serve as a constituent for heuristic methods. Also, new insights into the underlying problem can lead to relevant problem instances of a smaller size. For example, the tertiary structure of a protein might be influenced only by a few key subsequences of residues. If these key subsequences would be well-preserved they might be detected by an alignment of only a small number of (short) sequences. Not to forget, that for a fixed number of sequences, the multiple alignment problem is solvable in polynomial time.

This thesis has shown, that practical relevant problem instances can be solved to optimality in reasonable time, if we exploit the specific structure of the problem. Solving a relaxation of the multiple alignment problem by a slightly modified longest path computation for each pair of sequences is much more efficient than solving a linear programming relaxation that involves an exponential number of variables and constraints by a general-purpose LP solver. Similarly, in the interval constraint coloring problem we profit from the relaxation of the integer linear program to a set of network flow problems compared to solving the LP relaxation.

Alternatively, this thesis proposed approximation algorithms (with and without perfor-

mance guarantee) for the interval constraint coloring problem. Not only the restrictions imposed by the mathematical abstraction but also the inaccuracy of the experimental data might justify a slight violation of the coloring requirements, or in some cases make it even necessary. Whether it is more appropriate to minimize (exactly or approximate) the total deviation from the requirements than to generally allow a violation of all requirements by either a factor $(1 + \epsilon)$ or by an additive error $\pm 1$, has to be evaluated by biologists and biochemists. For the multiple sequence alignment, many heuristics and approximation algorithms have been proposed in the literature.

In essence, we consider it to be of great importance to provide biologists with various alternative methods, both exact and approximate. Their interpretation of our results within a biological context can lead to new insights into the underlying biological problem, which in turn can help to improve our mathematical models. This constant interaction with biologists is crucial for the advance in this research area.

# Bibliography


[ABE+02] Ernst Althaus, Alexander Bockmayr, Matthias Elf, Thomas Kasper, Michael Jünger, and Kurt Mehlhorn. SCIL - symbolic constraints in integer linear programming. In Rolf Möhring and Rajeev Raman, editors, *Algorithms - ESA 2002 : 10th Annual European Symposium*, volume 2461 of *Lecture Notes in Computer Science*, pages 75–87, Rom, Italy, September 2002. Springer.

[AC07] Ernst Althaus and Stefan Canzar. A Lagrangian relaxation approach for the multiple sequence alignment problem. In Andreas W. M. Dress, Yinfeng Xu, and Binhai Zhu, editors, *COCOA*, volume 4616 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 2007.

[AC08a] Ernst Althaus and Stefan Canzar. A Lagrangian relaxation approach for the multiple sequence alignment problem. *Journal of Combinatorial Optimization*, 16(2), August 2008.

[AC08b] Ernst Althaus and Stefan Canzar. LASA: A tool for non-heuristic alignment of multiple sequences. In *Proceedings of the 2nd Workshop on Algorithms in Molecular Biology (ALBIO 2008)*, Vienna, Austria, July 2008.

[ACE+08a] Ernst Althaus, Stefan Canzar, Khaled M. Elbassioni, Andreas Karrenbauer, and Julián Mestre. Approximating the interval constrained coloring problem. In Joachim Gudmundsson, editor, *SWAT*, volume 5124 of *Lecture Notes in Computer Science*, pages 210–221. Springer, 2008.

[ACE+08b] Ernst Althaus, Stefan Canzar, Mark R. Emmett, Andreas Karrenbauer, Alan G. Marshall, Anke Meyer-Bäse, and Huimin Zhang. Computing h/d-exchange speeds of single residues from data of peptic fragments. In Roger L. Wainwright and Hisham Haddad, editors, *SAC*, pages 1273–1277. ACM, 2008.

[Ach04] Tobias Achterberg. SCIP - a framework to integrate constraint and mixed integer programming. Technical Report 04-19, Zuse Institute Berlin, 2004. `http://www.zib.de/Publications/abstracts/ZR-04-19/`.

[ACLR06] Ernst Althaus, Alberto Caprara, Hans-Peter Lenhof, and Knut Reinert. Aligning multiple sequences by cutting planes. *Mathematical Programming*, 105:387–425, 2006.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall Inc., Englewood Cliffs, NJ, 1993.

[BCG+03] Michael Brudno, Michael Chapman, Berthold Göttgens, Serafim Batzoglou, and Burkhard Morgenstern. Fast and sensitive multiple alignment of large genomic sequences. *BMC Bioinformatics*, 4:66, 2003.

[Bea93] John E. Beasley. Lagrangian relaxation. *Modern heuristic techniques for combinatorial problems*, pages 243–303, 1993.

[BLP97] Vineet Bafna, Eugene L. Lawler, and Pavel A. Pevzner. Approximation algorithms for multiple sequence alignment. *Theor. Comput. Sci.*, 182(1-2):233–244, 1997.

[CEGK08] Jessica Chang, Thomas Erlebach, Renars Gailis, and Samir Khuller. Broadcast scheduling: Algorithms and complexity. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2008.

[CFT99] Alberto Caprara, Matteo Fischetti, and Paolo Toth. A heuristic method for the set cover problem. *Operations Research*, 47:730–743, 1999.

[CL88] Humberto Carrillo and David Lipman. The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics*, 48(5):1073–1082, 1988.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. McGraw-Hill Science/Engineering/Math, July 2001.

[DFJ54] G.B. Dantzig, D.R. Fulkerson, and S. Johnson. Solution of a Large-Scale Traveling-Salesman Problem. *Operations Research*, 2:393–410, 1954.

[Edg04] R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

[Epp90] David Eppstein. Sequence comparison with mixed convex and concave costs. *Journal of Algorithms*, (11):85–101, 1990.

[ERRS08] Khaled Elbassioni, Rajiv Raman, Saurabh Ray, and René Sitters. On the approximability of the maximum feasible subsystem problem with 0/1 coefficients. submitted, 2008.

[ESZ07] Khaled M. Elbassioni, René Sitters, and Yan Zhang. A quasi-PTAS for profit-maximizing pricing on line graphs. In *Proceedings of the 15th Annual European Symposium on Algorithms*, pages 451–462, 2007.

[Fis04] Marshall L. Fisher. Comments on "the lagrangian relaxation method for solving integer programming problems". *Manage. Sci.*, 50(12 Supplement):1872–1874, 2004.

[GCA00] M.B. Goshe, Y.H. Chen, and V.E. Anderson. Identification of the sites of hydroxyl radical reaction with peptides by hydrogen/deuterium exchange: Pr evalence of reactions with the side chains. *Biochemistry*, 39(7):1761–1770, 2000.

[GJ79a] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[GJ79b] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[GKPS06] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *J. ACM*, 53(3):324–360, 2006.

[GKS95a] Sandeep K. Gupta, John D. Kececioglu, and Alejandro A. Schäffer. Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *Journal of Computational Biology*, 2(3):459–472, 1995.

[GKS95b] S.K. Gupta, J.D. Kececioglu, and A.A. Schaeffer. Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *J. Comput. Biol.*, 2:459–472, 1995.

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, January 1997.

[Hås97] Johan Håstad. Some optimal inapproximability results. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1997. ACM.

[HK70] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[HK71] M. Held and R.M. Karp. The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

[ILO06] ILOG, Inc. ILOG CPLEX: High-performance software for mathematical programming and optimization, 2006. `http://www.ilog.com/products/cplex/`.

[Jus01] Winfried Just. Computational complexity of multiple sequence alignment with sp-score. *Journal of Computational Biology*, 8(6):615–623, 2001.

[KHJ+06] S. Kang, A.M. Hawkridge, K.L. Johnson, D.C. Muddiman, and P.E. Prevelige. Identification of subunit-subunit interactions in bacteriophage p22 procapsids by chemical cross-linking and mass spectrometry. *Journal of Proteome Research*, 5(2):370–377, 2006.

[KiKTM05] Kazutaka Katoh, Kei ichi Kuma, Hiroyuki Toh, and Takashi Miyata. MAFFT version 5: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Research*, 33:511, 2005.

[LAK89] D.J. Lipman, S.F. Altschul, and J.D. Kececioglu. A tool for multiple sequence alignment. *Proceedings of the National Academy of Sciences of the United States of America*, 86:4412–4415, 1989.

[LC02] J.F. Leite and M. Cascio. Probing the topology of the glycine receptor by chemical modification coupled to mass spectrometry. *Biochemistry*, 41(19):6140–6148, 2002.

[Lem01] Claude Lemaréchal. Lagrangian relaxation. In Michael Jünger and Denis Naddef, editors, *Computational Combinatorial Optimization*, volume 2241 of *Lecture Notes in Computer Science*, pages 112–156. Springer, 2001.

[LGS02] Christopher Lee, Catherine Grasso, and Mark F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[LLE+02] T.T. Lam, J.K. Lanman, M.R. Emmett, C.L. Hendrickson, Marshall A.G., and P.E. Prevelige. Mapping of protein:protein contact surfaces by hydrogen/deuterium exchange, followed by on-line high-performance liquid chromatography-electrospray ionization fourier-transform ion-cyclotron-resonance mass analysis. *Journal of Chromatography A*, 982(1):85–95, 2002.

[LR00] M. Lermen and K. Reinert. The practical use of the $\mathcal{A}^*$ algorithm for exact multiple sequence alignment. *Journal of Computational Biology*, 7(5):655–673, 2000.

[LS90] L.L. Larmore and B. Schieber. Online dynamic programming with applications to the prediction of RNA secondary structure. In *Proceedings of the First Symposium on Discrete Algorithms*, pages 503–512, 1990.

[Luc92] A. Lucena. Steiner problem in graphs: Lagrangean relaxation and cutting-planes. *COAL Bulletin*, 21:2–8, 1992.

[Luc93] A. Lucena. Tight bounds for the steiner problem in graphs. Technical Report TR-21/93, Dipartimento di Informatica, Univesitá degli Studi di Pisa, Pisa, Italy, 1993.

[MN95] Kurt Mehlhorn and Stefan Näher. LEDA: a platform for combinatorial and geometric computing. *Commun. ACM*, 38(1):96–102, 1995.

[Mor04] Burkhard Morgenstern. DIALIGN: multiple DNA and protein sequence alignment at BiBiServ. *Nucl. Acids Res.*, 32(2):W33–36, 2004.

[NHH00] C. Notredame, D. G. Higgins, and J. Heringa. T-Coffee: a novel method for fast and accurate multiple sequence alignment. *J Mol Biol*, 302(1):205–217, September 2000.

[NW99] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1999.

[PY88] Christos Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 229–234, New York, NY, USA, 1988. ACM.

[Rei99] K. Reinert. *A Polyhedral Approach to Sequence Alignment Problems.* PhD thesis, Universität des Saarlandes, 1999.

[SBH04] J.S. Sharp, J.M. Becker, and R.L. Hettich. Analysis of protein solvent accessible surfaces by photochemical oxidation and mass spectrometry. *Analytical Chemistry*, 76(3):672–683, 2004.

[SEM98] J. Stoye, D. Evers, and F. Meyer. Rose: generating sequence families. *Bioinformatics*, 14:157–163, 1998.

[THG94a] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22):4673–4680, November 1994.

[THG94b] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[TPP99] J. D. Thompson, F. Plewniak, and O. Poch. BAliBASE: A benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15(1):87–88, 1999. `http://www-igbmc.u-strasbg.fr/BioInfo/BAliBASE/prog_scores.html`.

[Uno01] Takeaki Uno. A fast algorithm for enumerating bipartite perfect matchings. In *Proceedings of the 12th International Conference Algorithms and Computation*, pages 367–379, 2001.

[WGWZ92] J. D. Watson, M. Gilman, J. Witkowski, and H. Zoller. *Recombinant DNA.* Freeman NY, 1992.

[WJ94] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J. Comput. Biol.*, 1(4):337–348, 1994.

[WLW05] Ivo Van Walle, Ignace Lasters, and Lode Wyns. SABmark - a benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics*, 21(7):1267–1268, 2005.

[Wol98] Laurence A. Wolsey. *Integer programming.* Wiley-interscience series in discrete mathematics and optimization. Wiley & Sons, New York, 1998.

[Wun96] Roland Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus.* PhD thesis, Technische Universität Berlin, 1996. `http://www.zib.de/Publications/abstracts/TR-96-09/`.

[ZGM97] Z. Zhang, S. Guan, and A. G. Marshall. Enhancement of the Effective Resolution of Mass Spectra of High Mass Biomolecules by Maximum Entropy-Based Deconvolution to Eliminate the Isoto pic Natural Abundance Distribution. *Journal of American Society of Mass Spectrometry*, 8:659 – 670, 1997.

[Zui02] E.R.P. Zuiderweg. Mapping protein-protein interactions in solution by nmr spectroscopy. *Biochemistry*, 41(1):1–7, 2002.