Efficient Algorithms for Markov Random Fields, Isotonic Regression, Graph Fused Lasso, and Image Segmentation

by

Cheng Lu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Industrial Engineering and Operations Research

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Dorit S. Hochbaum, Chair
Professor Ilan Adler
Assistant Professor Aditya Guntuboyina

Summer 2017

Efficient Algorithms for Markov Random Fields, Isotonic Regression, Graph Fused Lasso, and Image Segmentation

Abstract

Efficient Algorithms for Markov Random Fields, Isotonic Regression, Graph Fused Lasso, and Image Segmentation

by

Cheng Lu

Doctor of Philosophy in Engineering - Industrial Engineering and Operations Research

University of California, Berkeley

Professor Dorit S. Hochbaum, Chair

Markov random field (MRF) is a multi-label clustering model with applications in image segmentation, image deblurring, isotonic regression, graph fused lasso, and semi-supervised learning. It is a convex optimization problem on an arbitrary graph of objective function consisting of functions defined on the nodes (called *deviation* functions) and edges (called *separation* functions) of the graph. There exists a provably fastest MRF-algorithm for arbitrary graphs and a broad class of objective functions. This MRF-algorithm uses the technique of graph minimum cut. Although MRF of this class of objective functions generalizes isotonic regression and graph fused lasso, this MRF-algorithm has lower time complexity than those specialized algorithms for isotonic regression and graph fused lasso.

Some problems in time series and gene sequence signal processing are special cases of MRF on a path graph. We present three efficient algorithms to solve MRF on a path graph for different classes of objective functions. The algorithms are the fastest algorithms for the respective classes of objective functions. The first algorithm uses graph minimum cut technique inspired by the provably fastest MRF-algorithm. The second algorithm is based on a relationship with a lot-sizing problem in production planning. The third algorithm is based on the technique of Karush-Kuhn-Tucker (KKT) optimality conditions.

MRF is used in image segmentation to identify multiple salient features in an image. The Hochbaum Normalized Cut (HNC) model is a binary clustering model that is also applicable to image segmentation, with the goal to separate the foreground from the background. We compare the empirical performance between the HNC model and the spectral method in image segmentation. Both HNC and the spectral method can be viewed as heuristics for the NP-hard normalized cut criterion, which is another binary clustering criterion often used for image segmentation. We present experimental evidence that the HNC model provides solutions which not only better approximate the optimal normalized cut solution, but also have better visual segmentation quality over those provided by the spectral method. The experimental evidence further suggests that HNC should be the preferred image segmentation criterion over normalized cut.

To Mom, Dad, and Yu.

Contents

List of Figures

List of Tables

Acknowledgments

The 6 years' PhD life at Berkeley is of pain, but more of enjoyable memories. Words alone are for from enough to express my all-hearted great gratitude to the following people. Without their generosity, patience and love, I would not have gone through this path to complete the dissertation.

It's my great fortune and honor to work with my advisor, Professor Dorit S. Hochbaum. She brought me to Berkeley and opened to me a brand new fantastic world of algorithm and optimization. Her advice always illuminates my front road in the jungle of academia. Her insight is always my trustworthy compass guiding me to the right direction and her dedication is always my source of power to clear the impeding rocks and bushes along the journey of exploration. I will not forget the days and nights of discussion with Professor Hochbaum in her purple-door office. My brain was refreshed with new momentum and inspiration after every discussion.

I am also very fortunate to have Professor Ilan Adler and Assistant Professor Aditya Guntuboyina as my committee members. Professor Adler, world renowned expert in mathematical programming and optimization, pointed to me issues in this dissertation work through his sharp questions and comments that set me the direction of improvement. Professor Aditya Guntuboyina, an expert in statistics with beautiful work in statistical estimation, expanded in front of me a grand picture of statistical problems and helped me broaden the impact of the work.

My wife, Yu Du, always stand with me along this challenging journey. I am so lucky to meet her and have her that made the life not lonely. She shares my happiness, struggle, and frustration. I could not imagine how I would accomplish this dissertation without her devotion, encouragement, and everyday emotional support.

My office mates and Berkeley friends made my life at Berkeley colorful and full of excitement. My thanks goes to Dan Bu, Haoyang Cao, Ying Cao, Jue Chen, Shiman Ding, Long He, Lingxi Huang, Wenwen Jiang, June Lai, Kevin Li, Kun Li, Xiaoya Li, Sheng Liu, Stewart Liu, Wei Qi, Xiaochuan Qin, Amber Richter, Zhaoqi Situ, Quico Spaen, Nguyen Le Truong, Luming Wang, Meng Wang, Yujia Wu, Jianbo Xie, Yang Xu, Zhiwei Xu, Nan Yang, Hongcai Zhang, Yu Zhang, Min Zhao, Mo Zhou, Tailai Zhou, Junyan Zhu. I thank my friends for accompanying me with these most precious times in my twenties.

My dearest Mom and Dad always give me the strongest support to help me accomplish this dissertation. My deepest appreciation goes to their devotion and love to me. It's their love that sustain me to today. This dissertation is dedicated to them.

Thank you all! I love you!

Chapter 1

Introduction

Markov random field (MRF) is a convex optimization problem defined on a directed graph $G = (V, A)$ where each decision variable is associated with a node from the node set V. The objective function of MRF consists of two types of penalty functions - one associated with nodes in V and the other associated arcs in A. At each node, there is a convex penalty function that gives a cost to the assigned value of the variable associated with the node. Often this function penalizes the distance between the assigned value and an observed value on the node. We call these node-associated penalty functions *deviation* cost functions or data fidelity functions. On each arc, there is a convex penalty function that gives a cost to the distance between assigned values of the two variables incident to the arc. These arc-associated penalty functions are called *separation* cost functions or regularization functions. The objective of MRF is to assign values to the variables so that the sum of the deviation cost functions over all nodes and the separation cost functions over all arcs is minimized.

One common application of MRF is *image deblurring*. Given an noise degraded image, the goal of image deblurring is to reset color values to the pixels so as to minimize the penalty for the deviation from the observed colors, and furthermore, so that the discontinuity in terms of separation of reset colors between adjacent pixels is as small as possible. To model image deblurring as MRF, we let the pixels be nodes in a graph and the pairwise neighborhood relation be indicated by bi-directed arcs between neighboring pixels in the image. The optimal solution to the MRF problem that minimizes the sum of deviation and separation penalty is used as the reset color values to the pixels.

Applications of MRF appeared in many different fields, such as operations research (e.g. in [53, 44, 2]), production and supply chain management (e.g. in [106, 73, 90, 61]), economics (e.g. in [96]), signal processing (e.g. in [84]), bioinformatics (e.g. in [12, 13, 75]), image segmentation and deblurring (e.g. in [91, 44, 48]). MRF also includes as special cases isotonic regression, graph fused lasso, and a graph-based semi-supervised learning model that arise in statistical estimation and machine learning. These problems are of increasing interests given the great upsurge on data analytics and machine learning nowadays (see e.g. [100, 101, 36]).

There exist two provably fastest MRF-algorithms for MRF of arbitrary graphs and two broad classes of deviation functions. The two provably fastest algorithms, with broader applicability, have lower run time complexities than those of most specialized algorithms for isotonic regression and graph fused lasso. The existing fastest algorithm for MRF on arbitrary graphs of arbitrary convex deviation and separation functions offers a definitive and fast complexity to solve the graph-based semi-supervised learning model [5, 17, 36], whereas only generic convex optimization solvers were used previously [17, 63].

A *path* graph is one that nodes can lie on a sequence and there are only arcs between successive nodes along the sequence. Besides the above mentioned applications of MRF on general directed graphs, there are many applications that are special cases of MRF on a path graph, such as in simple isotonic regression (e.g. in [11, 88]), time series signal processing (e.g. in [67]), bioinformatics (e.g. in [35, 66]), and computer vision (e.g. in [37]). The theoretical contribution of the dissertation is that we present three most efficient algorithms to solve MRF on a path graph for different classes of deviation and separation functions. The first algorithm uses graph minimum cut technique inspired by the two provably fastest MRF-algorithms, with adaptation to path graphs to achieve a faster running time. The second algorithm solves a related lot-sizing problem in production planning. The third algorithm efficiently solves the KKT optimality conditions for the MRF problem on a path graph.

In addition, our new algorithms have matching or faster complexities than many of the existing best or recent specialized algorithms proposed for specific applications of the MRF-on-path-graph model.

The MRF model is used for multi-label clustering that can be applied to image segmentation to separate an image into multiple salient feature regions. Another model that can be used in image segmentation is the Hochbaum Normalized Cut (HNC) model [49]. It is a binary clustering model that is used to separate the foreground from the background of an image. We compare the empirical performance between the HNC model and the spectral method in image segmentation. Both HNC and the spectral method can be viewed as heuristics for the NP-hard normalized cut (NC) optimization criterion proposed by Shi and Malik in [95], which is a binary clustering criterion often used for image segmentation. We present experimental evidence that establishes three observations. First, the HNC model is better than the spectral method at approximating the optimal normalized cut solution; Second, the HNC model is better than the spectral method for good image segmentation quality; Third, the segmentation of subjectively best visual quality rarely corresponds to the one that minimizes the objective function value of the normalized cut criterion. The third observation implies that normalized cut may not be an appropriate criterion for image segmentation. Instead, HNC model not only provides better visual segmentations, but is also solvable in polynomial time. Therefore, HNC should be the preferred image segmentation criterion for both complexity and good segmentation quality reasons. This constitutes the empirical part of contribution of the dissertation.

1.1 Outline of the dissertation

The organization of the dissertation is as follows. In the remaining part of this chapter, we first introduce the formulation of MRF and its two special forms, which are all defined on arbitrary graphs, whose deviation functions are all assumed arbitrary convex but differ in the separation functions. Then we introduce preliminaries and notations that are required to understand the technical details presented in the dissertation.

The two special forms of MRF are the ones for which provably fastest MRF-algorithms exist. In Chapter 2 and Chapter 3, we review the complexities of these two MRF-algorithms, and show that they have better complexities than most of the specialized algorithms for isotonic regression and graph fused lasso. In Chapter 4, we review the complexity of the existing fastest algorithm for MRF on arbitrary graphs of arbitrary convex deviation and separation functions, and discuss its application to the graph-based semi-supervised learning model.

In Chapter 5, Chapter 6 and Chapter 7, we present the details of the three most efficient algorithms to solve MRF on path graphs for different classes of deviation and separation functions. Chapter 8 reports the empirical comparison between the HNC model and the spectral method for image segmentation. Chapter 8 is based off of a paper by Hochbaum, Lu, and Bertelli [51]. Chapter 9 concludes the dissertation.

1.2 Problem Formulations

Given a directed graph $G = (V, A)$, the Markov random field (MRF) problem is formulated as follows [2, 48]:

$$\begin{aligned} \text{(MRF)} \ \min_{x_i : i \in V} & \sum_{i \in V} f_i(x_i) + \sum_{(i,j) \in A} h_{i,j}(x_i - x_j) \\ s.t.\ & \ell_i \leq x_i \leq u_i, \ \forall i \in V. \end{aligned} \tag{1.1}$$

Function $f_i(x_i)$ is an arbitrary convex deviation function and function $h_{i,j}(x_i - x_j)$ is an arbitrary convex separation function.

We say that the separation function $h_{i,j}(x_i - x_j)$ is "bilinear" if it is in the form $h_{i,j}(x_i - x_j) = d_{i,j} \cdot (x_i - x_j)_+$, where $d_{i,j} \geq 0$ and the notation $(x)_+$ is the positive part of x, $\max\{x, 0\}$. This special form of MRF with "bilinear" separation functions is called MRF-BL problem in this dissertation, where "BL" stands for "bilinear" [44]:

$$\begin{aligned} \text{(MRF-BL)} \ \min_{x_i : i \in V} & \sum_{i \in V} f_i(x_i) + \sum_{(i,j) \in A} d_{i,j} \cdot (x_i - x_j)_+ \\ s.t.\ & \ell_i \leq x_i \leq u_i, \ \forall i \in V. \end{aligned} \tag{1.2}$$

The "bilinear" separation function in MRF-BL includes the ℓ_1-norm separation function as a special case, because an ℓ_1-norm function $d_i|x_i - x_j|$ can be written as $d_i \cdot (x_i - x_j)_+ +$

$d_i(x_j - x_i)_+ = d_i|x_i - x_j|$. This ℓ_1-norm function corresponds, for the pair of x_i and x_j, two arcs of reverse directions (i, j) and (j, i) with equal coefficients $d_{i,j} = d_{j,i} = d_i$.

One common application of the MRF-BL model is image segmentation, see Hochbaum in [44] and [48]. Besides, applications of MRF-BL to optimization problems on graphs were also illustrated by Hochbaum in [44].

For values of $d_{i,j}$ that are sufficiently large, an optimal solution to MRF-BL satisfies the partial order constraints $x_i \leq x_j$. We call this special case of MRF-BL (thus MRF), that has no separation functions but instead has partial order constraints $x_i \leq x_j$, as the convex cost closure (CCC) problem [53]:

$$\begin{aligned} \text{(CCC)} \ \min_{x_i : i \in V} & \sum_{i \in V} f_i(x_i) \\ s.t. \ & x_i \leq x_j, \ \forall (i,j) \in A \\ & \ell_i \leq x_i \leq u_i, \ \forall i \in V. \end{aligned} \tag{1.3}$$

The convex cost closure (CCC) problem is used for partial order estimation [53]. The partial order estimation problem arises from applications where noisy observations of parameters do not satisfy preset partial order requirement. The partial order estimation problem seeks to find an adjustment of the observations that fit the partial order constraints and minimize the total deviation penalty of the fitted values from the observations. The deviation penalty is a convex function of the fitted value. The partial order estimation problem has applications in numerous fields including production [106, 73, 90, 61], signal processing [84], economics [96], bioinformatics [12, 13, 75] and statistical learning [9, 10, 11, 88, 60, 83].

The CCC problem has been discussed extensively in the literature since four decades ago; see, e.g. Veinott [106] and Barlow et al. [11]. CCC has applications in production planning and operations research, see e.g. the review by Hochbaum and Queyranne in [53].

1.3 Preliminaries and Notations

Nonlinear optimization The problems discussed are all continuous optimization problems. A fundamental issue in continuous nonlinear optimization is to define what constitutes a solution to a nonlinear optimization problem. This is because the objective functions are assumed to be general convex, one cannot bound a priori the number of digits required for the length of nonlinear programming optimal solutions (they can consist of irrational or even transcendental numbers), yet digital computers can only store numbers of finite accuracy. For a detailed discussion, we refer to the review paper by Hochbaum in [45]. We adopt the ϵ-accuracy model, with a pre-specified parameter ϵ, to specify the output of a nonlinear optimization algorithm [54, 53, 45]. A feasible solution $\boldsymbol{x}$ is an *ϵ-accurate* solution if there exists an optimal solution $\boldsymbol{x}^*$ to the respective problem such that $||\boldsymbol{x} - \boldsymbol{x}^*||_\infty < \epsilon$. The value of $\log \frac{1}{\epsilon}$ indicates the number of significant digits in the ϵ-accurate solution. As such, a solution $\boldsymbol{x}$ is specified as an integer multiple of ϵ, i.e. it lies on the so-called ϵ-grid. The

ϵ-accuracy complexity model is the only way to solve problems that involve non-linear and non-quadratic functions on digital computers. This issue is discussed in detail in [54, 53, 44], and in the review [45].

In this dissertation, the objective functions of the CCC, MRF-BL and MRF problems, and their special cases, are all convex. Unless specified otherwise, we assume no restriction on the structure of the convex functions nor the functions to be differentiable. We assume that a convex function is represented via an oracle that, for any ϵ-accurate argument, it returns the function value in $O(1)$ time.

In CCC, MRF-BL, and MRF, box constraints are presented which bound the optimal values of the variables. We denote $U = \max_i\{u_i - \ell_i\} < \infty$. The issue of boundedness of the solution vector is critical in the analysis of nonlinear optimization. This issue is discussed in detail in [54]. In some applications, even though box constraints are not present, the bounded range is implied by the practical contexts or the objective functions.

Graph definitions We follow the convention to represent a directed graph as $G = (V, A)$, where V is a set whose elements are called vertices or nodes, and $A \subseteq V \times V$ is a set of ordered pairs of vertices of the form (i, j), called arcs. If the graph is undirected, then the set of unordered pairs of vertices $[i, j]$ are called edges. We denote the edge set as E.

A directed *path* of length n is an ordered list of nodes $(v_1, \ldots, v_n)$ so that $(v_i, v_{i+1}) \in A$ for all $i = 1, \ldots, n-1$. We call a graph $G = (V, A)$ a *bi-path graph* if for $V = \{1, \ldots, n\}$ the arc set is $A = \{(i, i+1), (i+1, i)\}_{i=1,\ldots,n-1}$.

In the sequel, when we discuss about MRF-BL and MRF on a path graph, we mean a bi-path graph.

Given a directed graph $G = (V, A)$, we denote $n = |V|$ and $m = |A|$. When G is undirected, m denotes the number of edges, $|E|$.

Minimum cuts and parametric minimum cuts Given a directed graph $G = (V, A)$, let the directed *s, t-graph* $G^{st} = (V_{st}, A_{st})$ be associated with graph $G = (V, A)$ such that $V_{st} = V \cup \{s, t\}$ and $A_{st} = A \cup A_s \cup A_t$. The appended node s is called the *source node* and t is called the *sink node*. The respective sets of *source adjacent arcs* and *sink adjacent arcs* are denoted as $A_s = \{(s, i) : i \in V\}$ and $A_t = \{(i, t) : i \in V\}$. Each arc $(i, j) \in A_{st}$ has an associated nonnegative capacity $c_{i,j}$.

For any two subsets of nodes $V_1, V_2 \subseteq V_{st}$, we let $(V_1, V_2) = \{(i, j) \in A_{st} | i \in V_1, j \in V_2\}$ and $C(V_1, V_2) = \sum_{(i,j)\in(V_1,V_2)} c_{i,j}$.

An *s, t-cut* is a partition of V_{st}, $(\{s\} \cup S, T \cup \{t\})$, where $T = V \setminus S$. For simplicity, we refer to an s, t-cut partition as (S, T). We refer to S as the *source set* of the cut, excluding s.

The *capacity* of a cut (S, T) is defined as $C(\{s\} \cup S, T \cup \{t\})$. A *minimum cut* in s, t-graph G^{st} is an s, t-cut (S, T) that minimizes $C(\{s\} \cup S, T \cup \{t\})$.

A parametric s, t-graph $G^{st}(\alpha) = (V_{st}, A_{st})$ is an s, t-graph whose arc capacities are functions of a single scalar parameter α that have the following properties:

1. The capacities of arcs in A (not adjacent to source or sink) are constants.
2. The capacities of source adjacent arcs are nonincreasing functions of α.
3. The capacities of sink adjacent arcs are nondecreasing functions of α.

The minimum cuts in a parametric s,t-graph are functions of α. Those minimum cuts are called *parametric minimum cuts*. There are parametric minimum cut procedures that solve parametric minimum cuts for all values of α in a complexity of solving a single minimum s,t-cut. Procedures known to date that can be used as parametric minimum cut and have this property (that they solve the parametric problem in the complexity of a single cut) are Hochbaum's pseudoflow (HPF) algorithm [50] and the push-relabel algorithm [39].

The parametric minimum cuts in a parametric s,t-graph are nested:

Lemma 1. (Nested Cut Property [39, 44, 50]) *For any two parameter values $\alpha_1 \leq \alpha_2$, let $S^*_{\alpha_1}$ and $S^*_{\alpha_2}$ be the respective source set of the minimum cuts of $G^{st}(\alpha_1)$ and $G^{st}(\alpha_2)$, then $S^*_{\alpha_1} \supseteq S^*_{\alpha_2}$.*

The nested cut property is used in deriving the faster algorithm to solve MRF-BL on a path graph with convex piecewise linear deviation functions (presented in Chapter 5) and in the HNC algorithm to give good image segmentation results.

Convex minimum cost network flow problem Our second algorithm for MRF-BL on a path graph involves solving a convex minimum cost network flow (convex MCNF) problem [3, 46] that we introduce here. In a convex MCNF problem instance, we are given a directed network $G = (V, A)$ with a convex cost function $c_{i,j}(\cdot)$, an upper bound $u_{i,j}$, and a lower bound $\ell_{i,j}$ associated with each directed arc (i, j), and a supply/demand b_i associated with each node i. The goal is to find a flow in the network of minimum total cost such that it satisfies the capacity constraints on the arcs and the supplies/demands at the nodes. Let the flow value on an arc (i, j) be $f_{i,j}$, thus the cost of flow $f_{i,j}$ on arc (i, j) is $c_{i,j}(f_{i,j})$. The convex MCNF problem is formulated as follows:

$$\begin{aligned} \text{(Convex MCNF)} \quad \min_{f_{i,j}:(i,j)\in A} \quad & \sum_{(i,j)\in A} c_{i,j}(f_{i,j}) \\ s.t. \quad & \sum_{j:(i,j)\in A} f_{i,j} - \sum_{j:(j,i)\in A} f_{j,i} = b_i, \ \forall i \in V \\ & \ell_{i,j} \leq f_{i,j} \leq u_{i,j}, \ \forall (i,j) \in A. \end{aligned}$$

Chapter 2

Faster Algorithm for Special Cases of MRF of Convex Deviations with Partial Order

2.1 A Sketch of Fastest Algorithm for MRF of Convex Deviations with Partial Order: HQ-Algorithm

MRF of general convex deviation functions (no separation functions) with arbitrary partial order constraints, represented by a directed graph $G = (V, A)$, is the CCC problem (1.3). The existing fastest algorithm is by Hochbaum and Queyranne in [53] (HQ-algorithm), whose complexity is $O(T(n, m) + n \log \frac{U}{\epsilon})$, where $T(n, m)$ is the complexity of solving a single minimum s, t-cut on an associated s, t-graph of G, and $O(n \log \frac{U}{\epsilon})$ is the complexity of finding the minima of n convex deviation functions in the respective interval to ϵ accuracy. This complexity is provably best possible since, as shown in [53], the CCC problem is a generalization of both the graph minimum cut problem and the problem of finding the minima of n convex deviation functions over each feasible interval $[\ell_i, u_i]$ to ϵ accuracy.

The HQ-algorithm uses a parametric minimum cut procedure to solve parametric minimum cuts over a parametric s, t-graph that achieves the complexity of solving a single minimum s, t-cut, $T(n, m)$. For general graphs, this currently best complexity of $T(n, m)$ that solves parametric minimum cuts is $O(nm \log \frac{n^2}{m})$, by either Hochbaum's pseudoflow (HPF) algorithm [52] or the push-relabel algorithm [39]. In the remainder of the dissertation, we take the complexity of HQ-algorithm as $O(nm \log \frac{n^2}{m} + n \log \frac{U}{\epsilon})$ when comparing it with other algorithms.

For the special case of CCC that the partial order is a total order, the respective graph is a directed path graph. For CCC on a directed path graph, the HQ-algorithm was shown to have complexity $O(n(\log n + \log(U/\epsilon)))$ in [53].

2.2 Faster Algorithm for Isotonic Regression with HQ-Algorithm

The isotonic regression (IR) model is an alias of the CCC problem, which is more well-known in the statistics community. When the underlying graph is a directed path, i.e., the partial order of the decision variables is a total order, the respective isotonic regression model is a *simple isotonic regression* (SIR) model which is formulated as follows:

$$\begin{aligned} \text{(SIR)} \quad \min_{x_1,\ldots,x_n} \quad & \sum_{i=1}^{n} f_i(x_i) \\ s.t.\ & x_i \le x_{i+1} \ i = 1,\ldots,n-1 \\ & \ell_i \le x_i \le u_i,\ i = 1,\ldots,n. \end{aligned} \tag{2.1}$$

Function $f_i(x_i)$ is a convex deviation function, which often penalizes the distance between estimated value x_i and some observed value.

Isotonic regression has been studied since 1950s [9, 12, 13, 75, 11, 88], starting in the statistics community. Isotonic regression has wide applications in many areas, such as statistics, economics and bioinformatics. We refer readers to [4, 100, 99] and the reference therein for applications.

We review existing efficient algorithms to solve SIR and IR with different types of objective functions, and compare them with HQ-algorithm in [53].

The PAVA (Pool Adjacent Violators Algorithm) [9, 11, 42] is a well-known $O(n)$ linear algorithm to solve SIR (2.1) of convex quadratic deviation functions, $f_i(x_i) = (x_i - a_i)^2$. This complexity is faster than HQ-algorithm for SIR of convex quadratic deviation functions, of complexity $O(n \log n)$ [53]. If the objective function is sum of absolute value functions (ℓ_1 norm), the problem SIR is known as *isotonic median regression* (IMR) [87, 74, 18, 81, 4]:

$$\begin{aligned} \text{(IMR)} \quad \min_{x_1,\ldots,x_n} \quad & \sum_{i=1}^{n} \sum_{j=1}^{q_i} w_{ij}|x_i - a_{ij}| \\ s.t.\ & x_i \le x_{i+1},\ i = 1,\ldots,n-1 \end{aligned} \tag{2.2}$$

In IMR, each deviation term relates to q_i observed values $\{a_{ij}\}_{j=1,\ldots,q_i}$. Applications of IMR in statistics can be found in [85, 86, 88]. Let $q = \sum_{i=1}^{n} q_i$. Chakravarti in [18] gave an $O(qn)$ algorithm and Pardalos et al. in [81] gave an $O(q \log^2 q)$ algorithm to solve IMR (2.2). The fastest algorithm to date is the HQ-algorithm in [53] with complexity $O(n \log q)$ provided that the q observations in the input are sorted (otherwise there is an additional complexity of $O(q \log n)$ for merging n sorted lists of q_i elements each to a sorted list of q elements). The special case where $q_i = 1$ for all i (thus $q = n$), called *simple isotonic median regression* (SIMR), can be solved in time $O(n \log n)$ by [53, 4]. In Chapter 5, we will present an algorithm that solves a generalization of IMR and achieves the same complexity $O(q \log n)$ for IMR and $O(n \log n)$ for SIMR.

In addition to IMR, both HQ-algorithm in [53] and Ahuja and Orlin's algorithm in [4] are applicable to SIR of arbitrary convex objective functions. HQ-algorithm has complexity $O(n \log n + n \log \frac{U}{\epsilon})$. Ahuja and Orlin's algorithm has complexity $O(n \log \frac{U}{\epsilon})$. The two complexities are comparable.

Furthermore, recall that HQ-algorithm in [53] is applicable to IR, i.e. CCC problem, of arbitrary convex deviation functions and arbitrary directed graphs. The complexity of HQ-algorithm is $O(nm \log \frac{n^2}{m} + n \log \frac{U}{\epsilon})$, and the complexity is provably the fastest possible. If the objective is convex quadratic, HQ-algorithm has complexity $O(nm \log \frac{n^2}{m})$ [53]. After this work of Hochbaum and Queyranne that was done more than a decade ago, there are papers proposing algorithms to solve IR with specific forms of objective functions. Angelov et al. in [8] studied the case of IR with ℓ_1-norm deviation functions:

$$\begin{aligned} \min_{x_i : i \in V} \quad & \sum_{i=1}^{n} w_i |x_i - a_i| \\ s.t. \quad & x_i \leq x_j, \ \forall (i,j) \in A. \end{aligned} \tag{2.3}$$

It was claimed in [8] an algorithm that solves problem (2.3) in time $O(nm + n^2 \log n)$, but the complexity of the algorithm is *incorrect*. Stout in [100] studied a case of IR with ℓ_p-norm deviation functions for $1 < p < +\infty$:

$$\begin{aligned} \min_{x_i : i \in V} \quad & \sum_{i=1}^{n} w_i |x_i - a_i|^p \\ s.t. \quad & x_i \leq x_j, \ \forall (i,j) \in A. \end{aligned} \tag{2.4}$$

It was claimed in [100] an algorithm that solves the problem (2.4) in complexity $O((nm + n^2 \log n) \log \frac{U}{\epsilon})$ where the factor $O(nm + n^2 \log n)$ is the complexity of the algorithm in [8] as it was used as a subroutine in Stout's algorithm in [100]. As the complexity of the algorithm in [8] is incorrect, hence the complexity of Stout's algorithm in [100] is also *incorrect*. The actual complexity is much higher than the claimed one.

We also note the following three series of papers by the same group of authors studying IR at increasing generalized forms of deviation functions. Luss et al. in [70] studied the case of convex quadratic deviation function $f_i(x_i) = (x_i - a_i)^2$, giving an algorithm of complexity $O(\min\{n^4, mn^2 \log n\})$. This complexity is much higher than the HQ-algorithm for convex quadratic deviation function, whose complexity is $O(nm \log \frac{n^2}{m})$. Later Luss and Rosset in [69] studied the case where $f_i(x_i)$ is a convex and differentiable function, giving an algorithm of complexity $O(\min\{n^4, mn^2 \log n\} \log \frac{U}{\epsilon})$. Following that, Painsky and Rosset in [79] studied the case where $f_i(x_i)$ is convex yet not assumed to be differentiable, giving an algorithm of complexity $O(\min\{n^4, mn^2 \log n\} \log \frac{U}{\epsilon})$. As a comparison, HQ-algorithm does not assume differentiability of deviation functions, and has better complexity $O(nm \log \frac{n^2}{m} + n \log \frac{U}{\epsilon})$. We note that the three algorithms from the three papers are very similar, and they all rely on properties of CCC that were first shown and used by HQ-algorithm in [53].

Problem	$f_i(x_i)$	HQ-Algorithm	Specialized Algorithm(s)
SIR	$(x_i - a_i)^2$	$O(n \log n)$	$O(n)$ [9, 11, 42]
	$\sum_{j=1}^{q_i} \lvert x_i - a_{ij} \rvert$	$O(n \log q + q \log n)$	$O(qn)$ [18] $O(q \log^2 q)$ [81]
	$\lvert x_i - a_i \rvert$	$O(n \log n)$	$O(n \log n)$ [4]
	Convex	$O(n \log n + n \log \frac{U}{\epsilon})$	$O(n \log \frac{U}{\epsilon})$ [4]
IR	$(x_i - a_i)^2$	$O(nm \log \frac{n^2}{m})$	$O(\min\{n^4, mn^2 \log n\})$ [70]
	Convex	$O(nm \log \frac{n^2}{m} + n \log \frac{U}{\epsilon})$	$O(\min\{n^4, mn^2 \log n\} \log \frac{U}{\epsilon})$ [79]

Table 2.1: Comparison of HQ-algorithm in [53] with existing specialized algorithms for simple isotonic regression (SIR) and isotonic regression (IR).

We summarize the above review and comparison results in Table 2.1. The results show that HQ-algorithm in [53], for most classes of convex deviation functions, has faster or comparable complexity than the complexities of the other specialized algorithms for both SIR and IR.

Chapter 3

Faster Algorithm for Special Cases of MRF of Convex Deviations and "Bilinear" Separations

3.1 A Sketch of Fastest Algorithm for MRF of Convex Deviations and "Bilinear" Separations: H01-Algorithm

MRF of general convex deviation functions and "bilinear" separation functions is the MRF-BL problem (1.2). The existing fastest algorithm is by Hochbaum in [44] (H01-algorithm), whose complexity is the same as the HQ-algorithm for CCC, $O(T(n,m) + n\log\frac{U}{\epsilon})$, where $T(n,m)$ is the complexity of a parametric minimum cut procedure that equals to the complexity of a single minimum s,t-cut. This complexity is also provably the fastest possible because MRF-BL also generalizes both the graph minimum cut problem and the problem of finding the minima of n convex deviation functions over each feasible interval $[\ell_i, u_i]$ to ϵ-accuracy. We also take the value $T(n,m)$ as $O(nm\log\frac{n^2}{m})$ and thus the complexity of H01-algorithm is $O(nm\log\frac{n^2}{m} + n\log\frac{U}{\epsilon})$ when comparing it with other algorithms.

3.2 Faster Algorithm for Graph Fused Lasso with H01-Algorithm

Graph fused lasso (GFL) is a model used for data smoothing where the each data point is associated with a node of an undirected graph $G = (V, E)$. Let the noisy observation at node $i \in V$ be $a_i \in \mathbb{R}$. The problem is that the observed values do not satisfy the local smoothness assumption in the graph. The goal of the graph fused lasso model is to adjust the observed values so that on one hand they are "close" to the values of their neighbors along the edges (fused lasso regularization or separation), and on the other hand do not deviate too much from the observations (deviation aka fidelity). Let the adjusted value of node i to be computed be x_i. Graph fused lasso is the following optimization problem whose optimal solution is used as the adjusted values:

$$\begin{aligned}\text{(GFL)} \ \min_{x_i:i\in V} \ &\sum_{i\in V} f_i(x_i;a_i) + \lambda \sum_{[i,j]\in E} |x_i - x_j| \\ &s.t.\ \ell_i \leq x_i \leq u_i,\ i \in V.\end{aligned} \tag{3.1}$$

In the formulation of GFL, parameter $\lambda \geq 0$ is a tuning parameter that measures the relative importance between the deviation function costs and the separation function costs.

The special case of GFL on a 2-dimensional grid graph has applications in the field of image deblurring and segmentation, see e.g. [41, 14, 40, 65, 16, 58, 44, 48]. The GFL model is also known as a *total variation* model on a graph that was first proposed by Rudin et al. in [91].

The GFL problem is a special case of MRF-BL (1.2) on an associated graph where an edge $[i,j] \in E$ is split as two directed arcs $(i,j),(j,i) \in A$ and the respective separation functions have same coefficient $d_{i,j} = d_{j,i} = \lambda$. As a result, the H01-algorithm in [44] is applicable to GFL (3.1). Recall that H01-algorithm has provable complexity of $O(nm\log\frac{n^2}{m} + n\log\frac{U}{\epsilon})$. When the deviation functions are convex quadratic, H01-algorithm reduces to complexity of $O(nm\log\frac{n^2}{m})$ [44].

After the H01-algorithm, specialized algorithms were derived for solving GFL with convex quadratic deviation functions and arbitrary graphs. Hoefling in [55] gave an algorithm for this special case that finds an approximate solution. The complexity of his algorithm is $O(T_{\text{max\_flow}}(n,m)T)$, where $T_{\text{max\_flow}}(n,m)$ is the complexity of computing a maximum flow [3] on a graph of n nodes and m edges, and T is the number of iterations of the algorithm. Hoefling did not give in [55] a definitive bound on T. If we quote the currently best complexity for solving maximum flow that depends only on n and m, we have $T_{\text{max\_flow}}(n,m)$ is $O(nm)$ given by Orlin in [78]. Thus Hoefling's algorithm in [55] has complexity $O(nmT)$. Comparing with the H01-algorithm in [44], we see that, unless $T = O(\log\frac{n^2}{m})$, the H01-algorithm has faster running time. Another algorithm was given by Tibshirani and Taylor in [103] for GFL of convex quadratic deviation functions and arbitrary graphs. Tibshirani and Taylor showed that their algorithm has run time complexity $O(mn^2+Tm^2)$ when $m \leq n$, and $O(m^2n+Tn^2)$ when $m > n$, where T is the number of iterations of their algorithm and $T \geq m$ [103]. Hence their algorithm has higher complexity than the H01-algorithm.

Chapter 4

Fast Algorithm for Special Cases of MRF of Convex Deviations and Convex Separations

4.1 A Sketch of Fast Algorithm for MRF of Convex Deviations and Convex Separations: AHO-Algorithm

For MRF of general convex deviation and general convex separation functions, the existing fastest algorithm is by Ahuja et al. in [2] (AHO-algorithm), whose complexity is $O(nm \log \frac{n^2}{m} \log \frac{nU}{\epsilon})$. Note that although the term $O(nm \log \frac{n^2}{m})$ is the same as what we discuss above on parametric minimum cuts, the AHO-algorithm has nothing to do with parametric minimum cuts. It uses a different technique that is based on solving a convex MCNF problem related to MRF.

4.2 Fast Algorithm for a Graph-based Semi-Supervised Learning Model with AHO-Algorithm

Semi-supervised learning is to infer the labels of a large set of unlabeled data from a small set of labeled data. A challenge in some machine learning problems is that the size of labeled data is far smaller than the size of unlabeled data. This is because obtaining labeled data is often very time consuming and expensive, as they require the efforts of human annotators, who must often be quite skilled [114]. One approach is the following semi-supervised learning model that construct a graph representing the data and their similarities. This approach first solves a real-valued regression problem on a weighted undirected graph that computes continuous "label" values to the data, and then thresholds the continuous values to generate discrete labels to the unlabeled data [114]. The graph regression problem is formulated as follows. Suppose we have a data set of size n, and the data are indexed from 1 to n. Let $\boldsymbol{a}_i \in \mathbb{R}^d$ be the feature vector of the ith datum. Suppose the first l data are labeled with labels $t_1, \dots, t_l$, and the remaining $u = n - l$ data are unlabeled. Usually we have $l \ll u$. Let $L = \{1, 2, \dots, l\}$ be the set of labeled data and $U = \{l+1, l+2, \dots, l+u = n\}$ be the set of unlabeled data. Consider a connected weighted undirected graph $G = (V = L \cup U, E)$ with nodes in V corresponding to the n data. The weight $w_{i,j}$ of each edge $[i,j] \in E$ is a function of the feature vectors $\boldsymbol{a}_i$ and $\boldsymbol{a}_j$ that measures the similarity between the ith and jth data, either labeled or unlabeled. For example, one can use the following Gaussian-based similarity measure:

$$w_{i,j} = \exp\left(-\sum_{k=1}^{d} \frac{(a_{ik} - a_{jk})^2}{\sigma_k^2}\right),$$

where a_{ik} is the k-th component of feature vector $\boldsymbol{a}_i$ and σ_k is a length scale hyperparameter for each dimension [114]. Let $x_i \in \mathbb{R}$ be the estimated continuous label of node i ($i = 1, \dots, l, l+1, \dots, l+u = n$) to be solved, including both labeled and unlabeled data. The graph regression model has a noiseless and a noisy version. In the noiseless version, the labels of the labeled data are assumed to be noise free. Thus the optimization problem to solve x_is is a constrained problem that minimizes the total ℓ_q-norm separation between pairs of estimated continuous label values of nodes along the edges, weighted by their similarity. The problem is referred as the (discrete) q-Laplacian of the graph G [36]:

$$\begin{aligned} (q\text{-Laplacian(noiseless)}) \quad \min_{x_i : i \in V = L \cup U} \; & \sum_{[i,j] \in E} w_{i,j} |x_i - x_j|^q \\ s.t. \; & x_i = t_i, \; i = 1, \dots, l. \end{aligned} \tag{4.1}$$

In the noisy version, the observed labels are not completely truthful. Hence we allow the estimated values on the labeled data to be deviated from the observed labels, but the deviation is penalized by a deviation cost function in the objective:

$$(q\text{-Laplacian(noisy)}) \quad \min_{x_i : i \in V = L \cup U} \sum_{i=1}^{l} f_i(x_i; t_i) + \sum_{[i,j] \in E} w_{i,j} |x_i - x_j|^q. \tag{4.2}$$

The noiseless version is a special case of the noisy version because one can give a sufficiently large deviation penalty in the noisy version to prevent the estimated value from deviating from the observed label on the labeled data. The modeling assumption of graph q-Laplacian is that data of similar features should have similar labels [114, 36]. The graph q-Laplacian model for semi-supervised learning has been studied extensively, see [114, 76, 5, 17, 36] and the references therein.

The graph q-Laplacian model for semi-supervised learning was first proposed by Zhu et al. in [114] for $q = 2$:

$$\text{(2-Laplacian(noiseless))} \quad \min_{x_i : i \in V = L \cup U} \sum_{[i,j] \in E} w_{i,j}(x_i - x_j)^2 \qquad \text{s.t. } x_i = t_i,\ i = 1, \ldots, l. \tag{4.3}$$

As the separation functions are quadratic and the constraints are equality constraints, Zhu et al. in [114] showed that solving 2-Laplacian(noiseless) (4.3) is equivalent to solving a system of linear equations, and thus the optimal solution has a closed-form matrix representation. Despite its nice computational and representation property, the 2-Laplacian(noiseless) model may be ill-posed for some semi-supervised learning tasks. Nadler et al. in [76] studied this model in the setting where the number of labeled data l is fixed yet the number of unlabeled data u goes to infinity on a d-dimensional geometric random graph. They showed that as $u \to \infty$, the optimal solution to problem (4.3) is degenerated in that it converges to a solution that is constant everywhere except for the labeled data that are forced to be equal to the observed labels by the equality constraints. This phenomenon also exists for the noisy counterpart of 2-Laplacian(noiseless) (4.3), as every feasible solution to the noiseless version (4.1) is feasible to the noisy version (4.2).

Following the result of Nadler et al. in [76], higher orders of q in the graph q-Laplacian model are investigated to circumvent the degeneracy issue associated with the 2-Laplacian case [5, 17, 36]. A "phase transition" property was shown for this model: When q is small, the optimal solution is degenerated; however, there is a "phase transition" value such that when q is greater than this specific value, the resulting graph q-Laplacian model gives non-degenerated optimal solution, which is meaningful for the semi-supervised learning task. The most recent result is by El Alaoui et al. in [36]. In this paper they studied the phase transition property of graph q-Laplacian model on a d-dimensional geometric random graph. They showed that the optimal solution is degenerated for $q \leq d$, whereas for $q \geq d + 1$, the optimal solution is smooth and non-degenerated. Hence they recommend $q = d + 1$ as an optimal choice of the power index, yielding a solution that is smooth, non-degenerate, and remaining maximally sensitive to the structure of the data [36].

Despite the above active study on statistical properties of the graph q-Laplacian model for semi-supervised learning, there is no specialized efficient algorithm derived to solve this model. For the q-Laplacian(noisy) model (4.2) with $q > 2$, only generic convex optimization solvers were used in [17, 63]. However, there is no definitive complexity results of generic convex optimization solvers for the q-Laplacian(noisy) model. One reason is that theoretical convergence rate analyses of many generic convex optimization solvers for unconstrained convex optimization problems, like decent-type algorithms, does not apply to the q-Laplacian(noisy) model because those analyses require the gradients of objective function to be Lipschitz continuous [15, 68], which does not hold for the q-Laplacian(noisy) model of arbitrary convex deviation functions $f_i(x_i; t_i)$ and ℓ_q-norm separation functions. Furthermore, regardless of the Lipschitz continuity condition, many generic solvers converge at rate

$O(poly(1/\epsilon))$ to an optimal objective value of ϵ-accuracy [68], but $1/\epsilon$ is not a polynomial quantity because one only needs $O(\log(1/\epsilon))$ bits to achieve ϵ-accuracy.

On the other hand, note that the q-Laplacian(noisy) model (4.2) is a special case of MRF (1.1). Hence algorithms for MRF are immediately applicable. Recall that the fastest algorithm for MRF is AHO-algorithm in [2] with definitive complexity $O(nm \log \frac{n^2}{m} \log \frac{nU}{\epsilon})$, where $U = O(\max_{i \in L}\{t_i\} - \min_{i \in L}\{t_i\})$ for the q-Laplacian(noisy) model (One can easily check that the optimal values of all variables of q-Laplacian(noisy) are in the range of $[\min_{i \in L}\{t_i\}, \max_{i \in L}\{t_i\}]$). Hence the AHO-algorithm has a definitive and faster run time complexity over generic convex optimization solvers for the q-Laplacian(noisy) model (4.2).

Chapter 5

Min-Cut-based Algorithm for MRF on Path: Speed-up of Nearly-Isotonic Median Regression and Generalizations

The problem studied in this chapter is MRF-BL on a path graph of convex piecewise linear deviation functions. This special case of MRF-BL is a generalization of several well-known problems including the Isotonic Median Regression (IMR). In this chapter we call this special case problem of MRF-BL *Generalized Isotonic Median Regression* (GIMR). The GIMR problem is formulated as:

$$\begin{aligned} &\min_{x_1,\ldots,x_n} \sum_{i=1}^{n} f_i^{pl}(x_i;\{a_{i,j}\}_{j=1}^{q_i}) + \sum_{i=1}^{n-1} d_{i,i+1}(x_i - x_{i+1})_+ + \sum_{i=1}^{n-1} d_{i+1,i}(x_{i+1} - x_i)_+ \qquad (5.1)\\ &\text{(GIMR)} \quad s.t.\ \ell_i \le x_i \le u_i, \quad i = 1,\ldots,n, \end{aligned}$$

where each deviation function $f_i^{pl}(x_i)$ is a convex piecewise linear function with q_i breakpoints $a_{i,1} < a_{i,2} < \ldots < a_{i,q_i}$ (the superscript "pl" stands for "piecewise linear"). The separation terms involve the nonnegative coefficients $d_{i,i+1}$ and $d_{i+1,i}$ for positive and negative separation penalties, and the notation $(x)_+$ is the positive part of x, $\max\{x, 0\}$. For values of $d_{i,i+1}$ that are sufficiently large, an optimal solution to GIMR satisfies the total rank order $x_1 \le x_2 \le \ldots \le x_n$. The set of feasible values for each x_i are contained in the interval $[\ell_i, u_i]$.

5.1 Special Cases and Applications

Models with Deviation Terms Only

A commonly used special case of convex piecewise linear deviation functions, in the context of isotonic regression, is the ℓ_1 norm. An ℓ_1 deviation function is a sum of (weighted) absolute values of the differences between the estimated values and the respective observation values. There are a number of advantages for the use of the ℓ_1 deviation function: This function gives the exact maximum likelihood estimate if the noises in the observation values follow the Laplacian distribution [74, 18]; it is in general robust to heavy-tailed noises and to the presence of outliers [77, 107, 98]; it provides better preservation of the contrast and the invariance to global contrast changes [20, 98]. The ℓ_1 deviation function, in weighted or unweighted form, was used in a number of models and applications. Isotonic median regression (IMR) was studied in [87, 74, 18, 81], and its applications in statistics can be found in [85, 86, 88]. Additional areas in which IMR on partial order has been applied include chromosomal microarray analysis (CMA) [8] in bioinformatics and ordinal classification with monotonicity constraints [29]. We note that the algorithm of [8] is incorrect, and the complexity of the corrected version is considerably worse than the complexity of the algorithm of [53] for IMR.

The formulation of IMR with weights w_{ij} is:

$$\begin{aligned} &\min_{x_1,\ldots,x_n} \sum_{i=1}^{n}\sum_{j=1}^{q_i} w_{ij}|x_i - a_{ij}| \\ \text{(IMR)}\quad & s.t.\ x_i \le x_{i+1}, \quad i = 1,\ldots,n-1. \end{aligned} \tag{5.2}$$

When each variable x_i is associated with only one observation a_i, i.e. $q_i = 1$, IMR is then referred to as the Simple Isotonic Median Regression (SIMR):

$$\begin{aligned} &\min_{x_1,\ldots,x_n} \sum_{i=1}^{n} w_i|x_i - a_i| \\ \text{(SIMR)}\quad & s.t.\ x_i \le x_{i+1}, \quad i = 1,\ldots,n-1. \end{aligned} \tag{5.3}$$

SIMR (5.3) was studied extensively, e.g. in [53, 4].

Some applications use deviation functions that are not ℓ_1 functions, but use only two pieces in the piecewise linear deviation function. One example is the *Quantile deviation* function which preserves the robustness property of the ℓ_1 deviation function. The quantile deviation function of estimated variable x_i from observation a_i for parameter $\tau \in [0,1]$ is:

$$\rho_\tau(x_i; a_i) = \begin{cases} \tau(x_i - a_i) & \text{if } x_i - a_i \ge 0 \\ -(1-\tau)(x_i - a_i) & \text{if } x_i - a_i < 0. \end{cases} \tag{5.4}$$

Here the parameter τ represents the quantile of the observations of interest. For $\tau = \frac{1}{2}$ (half-quantile), the quantile deviation function is identical to the absolute value function. Quantile deviations have been used in array-based comparative genomic hybridization (array-CGH) analysis in bioinformatics [35, 66].

Another special case of the convex piecewise linear function is the ϵ-*insensitive deviation* function, defined as follows [79]:

$$d_\epsilon(x_i; a_i) = \begin{cases} x_i - a_i - \epsilon \quad \text{if } x_i - a_i > \epsilon \\ 0 \quad \text{if } |x_i - a_i| \leq \epsilon \\ -x_i + a_i - \epsilon \quad \text{if } x_i - a_i < -\epsilon. \end{cases} \tag{5.5}$$

The absolute value function is a special case of the 0-*insensitive deviation* function.

The use of isotonic regression for medical prognosis was discussed by Ruy et al. [92]. They considered convex piecewise linear deviation functions (each with 3 pieces) and a partial order that is derived from medical knowledge on the relative prognosis prospects for pairs of feature vectors.

Models that Include Separation/Regularization Terms

There are variant models of total order estimation that include penalty terms in the objective on the violation of total order constraints, instead of imposing those constraints. These penalty functions are often referred to as separation or regularization functions. We present here four such models that were presented for specific contexts.

Tibshirani et al., [102], studied a "nearly-isotonic" model on total order for the purpose of fitting global warming data on annual temperature anomalies. Here a_i is the observation of the temperature anomaly value at the ith year of the dataset:

$$\text{(Nearly-isotonic)} \quad \min_{x_1,\ldots,x_n} \frac{1}{2}\sum_{i=1}^{n}(x_i - a_i)^2 + \lambda\sum_{i=1}^{n-1}(x_i - x_{i+1})_+. \tag{5.6}$$

In this model (5.6), the tuning parameter $\lambda \geq 0$ measures the relative importance between the deviation terms and the separation terms. As $\lambda \to \infty$ the problem is equivalent to imposing the total order constraints of IMR (5.2) and SIMR (5.3). In model (5.6) the quadratic deviation terms are based on the Gaussian noise assumption on the observations. We note that the GIMR model differs from (5.6) in that instead of convex quadratic deviation functions it has convex piecewise linear deviation functions. This convex piecewise linear class of functions includes ℓ_1 deviation functions that are considered to be more appropriate as a model for Laplacian noises, or heavy-tailed noises.

The separation term in the nearly-isotonic model (5.6), $\lambda(x_i - x_{i+1})_+$, is "one-sided" in that it only penalizes the surplus of x_i over x_{i+1}. A more general separation term also penalizes the surplus of x_{i+1} over x_i, in the form of $\lambda'(x_{i+1} - x_i)_+$, leading to a "two-sided" separation penalty. If the two-sided penalty is symmetric ($\lambda = \lambda'$), it can be presented as the absolute value (ℓ_1) separation term $\lambda|x_i - x_{i+1}|$. This ℓ_1 separation penalty is known as *fused lasso* [104]. An example of the use of fused lasso model is in array-based comparative genomic hybridization (array-CGH) analysis, [35, 66]. It is to estimate the ratio of gene copying numbers at each position in DNA sequences between tumor and normal cell samples, based on the biological knowledge that the ratios between adjacent positions in the DNA sequences are similar. Eilers and de Menezes, [35], proposed the following quantile fused lasso (Q-FL) model to identify the estimated log-ratio x_i, based on the observed log-ratio a_i at the ith position:

$$\text{(Q-FL)} \quad \min_{x_1,\ldots,x_n} \sum_{i=1}^{n} \rho_\tau(x_i; a_i) + \lambda \sum_{i=1}^{n-1} |x_i - x_{i+1}|. \tag{5.7}$$

The deviation terms are the quantile deviation functions (5.4) and the ℓ_1 separation functions $|x_i - x_{i+1}|$ drive the log-ratios of adjacent positions to be similar. Later, Li and Zhu in [66] extended the above model to the following quantile weighted fused lasso (Q-wFL) model by considering the distances between adjacent positions, which is claimed to help improve the estimation [66]:

$$\text{(Q-wFL)} \quad \min_{x_1,\ldots,x_n} \sum_{i=1}^{n} \rho_\tau(x_i; a_i) + \lambda \sum_{i=1}^{n-1} \frac{1}{d_{i,i+1}} |x_i - x_{i+1}|, \tag{5.8}$$

where $d_{i,i+1} \in \mathbb{R}$ is the distance between the ith and the $(i+1)$th positions. Thus the closer the adjacent positions, the larger penalty on the log-ratio difference. The quantile deviation functions are also claimed in [35, 66] to be advantageous over the standard quadratic deviation functions (least square mean regression).

In signal processing, Storath et al., [98], considered a fused lasso model with ℓ_1 deviation functions:

$$(\ell_1\text{-FL}) \quad \min_{x_1,\ldots,x_n} \sum_{i=1}^{n} w_i |x_i - a_i| + \lambda \sum_{i=1}^{n-1} |x_i - x_{i+1}|, \tag{5.9}$$

where w_is are nonnegative weights and $\lambda > 0$ is the model tuning parameter.

Recently, Kolmogorov et al., [62], studied a weighted fused lasso problem, which generalizes Q-FL (5.7), Q-wFL (5.8) and ℓ_1-FL (5.9), by allowing deviation functions to be general convex piecewise linear functions with $O(1)$ breakpoints each, and different weights on the ℓ_1 separation terms $|x_i - x_{i+1}|$ in place of the uniform coefficient λ. This problem is denoted by PL-wFL-$O(1)$, where PL stands for "piecewise linear", and $O(1)$ indicates that each convex piecewise linear deviation function has $O(1)$ breakpoints.

5.2 Existing Best Algorithms for Total Order Estimation

Total order estimation with deviation terms only. For the special case that the partial order is a total order, the respective graph is a directed *path* graph. For the directed path graph HQ-algorithm was shown to have complexity $O(n(\log n + \log(U/\epsilon)))$ in [53] and the algorithm given by Ahuja and Orlin in [4] has complexity $O(n \log(U/\epsilon))$. (It is important to note that the complexity term of $\log(U/\epsilon)$ is provably unavoidable when minimizing a non-linear and non-quadratic convex function, as discussed in [53] and is based on the impossibility of strongly polynomial algorithms for non-linear non-quadratic optimization proved in [47]). For quadratic deviation functions, the run time of HQ-algorithm is $O(n \log n)$.

Let $q = \sum_{i=1}^{n} q_i$ be the total number of breakpoints of the n convex piecewise linear deviation functions, $\{f_i^{pl}(x_i)\}_{i=1,\dots,n}$, in GIMR. For the GIMR problem with deviation terms only, (or very large d parameters), if the q breakpoints of the convex piecewise linear deviation functions are sorted, the complexity of HQ-algorithm is $O(n(\log n + \log q)) = O(n \log q)$ $(q = \Omega(n))$, and otherwise the sorting complexity of $O(q \log n)$ is added.

Other known algorithms for IMR (5.2) include an algorithm in [18] of complexity $O(qn)$ and an algorithm of complexity $O(q \log^2 q)$ by [81]. GIMR-algorithm shown here solves IMR (5.2) in time $O(q \log n)$, which improves the complexity of [18] and [81], and matches the complexity of HQ-algorithm in [53]. For SIMR (5.3), the fastest algorithms to date, by [53, 4], have complexity $O(n \log n)$. GIMR-algorithm solves SIMR (5.3) in the same complexity.

Total order estimation with separation terms. GIMR is a special case of MRF-BL (1.2) where the graph G is a bi-path graph with node set $V = \{1, \dots, n\}$, arc set $A = \{(i, i+1), (i+1, i)\}_{i=1,\dots,n-1}$, and the deviation functions are convex piecewise linear. Therefore, any algorithm that solves MRF-BL can solve GIMR. A direct application of H01-algorithm for MRF-BL to GIMR (5.1) (total order graph) is of complexity $O(n^2 \log n + n \log q)$ since $m = O(n)$ and the deviation functions are convex piecewise linear, so finding n times the minima of such functions requires at most binary search on the set of breakpoints. Our main contribution here can be viewed as an algorithm that speeds up H01-algorithm for MRF-BL by efficiently generating the respective minimum s,t-cuts for a path graph.

Other algorithms were devised for various special cases of total order estimation with separation terms: Eilers and de Menezes in [35], and Li and Zhu in [66], derived algorithms to solve Q-FL (5.7) and Q-wFL (5.8) respectively, both based on linear programming. They did not state concrete complexity results. For the ℓ_1-FL problem (5.9) Storath et al. in [98] proposed an algorithm of complexity $O(n^2)$. GIMR algorithm solves all these problems in $O(n \log n)$ complexity.

For problem PL-wFL-$O(1)$, Kolmogorov et al., [62], derived an algorithm of complexity $O(n \log n)$, which, like the algorithm presented here, uses a method for efficiently generating the respective minimum s,t-cuts for H01-algorithm and achieves the same complexity as ours (using a different methodology). Since PL-wFL-$O(1)$ generalizes Q-FL (5.7), Q-wFL (5.8) and ℓ_1-FL (5.9), this is also an alternative fastest algorithm for these problems. Note that Kolmogorov et al. in [62] also claimed an $O(n \log \log n)$ algorithm for the PL-wFL-$O(1)$ problem. However, the divide-and-conquer technique used in the algorithm requires the sorting of the breakpoints, which adds to the complexity $O(n \log n)$.

5.3 Summary of Results

In Table 5.1 we provide a comparison of the complexity of our algorithm for GIMR and its special cases as compared to the best and recent algorithms' complexities known to date.

Table 5.1: Summary of comparison of complexities. Here LP stands for the complexity of solving a linear programming problem of size $O(n)$.

Problem	Deviation	Separation	Algorithm here	Algorithms to-date
GIMR	Convex piecewise linear	$d_{i,i+1}(x_i - x_{i+1})_+ + d_{i+1,i}(x_{i+1} - x_i)_+$	$O(q \log n)$	$O(n^2 \log n + n \log q)$ [44]
IMR	$\sum_{j=1}^{q_i} \lvert x_i - a_{ij} \rvert$		$O(q \log n)$	$O(n \log q + q \log n)$ [53]
SIMR	$\lvert x_i - a_i \rvert$		$O(n \log n)$	$O(n \log n)$ [53, 4]
Nearly-isotonic	Convex piecewise linear	$\lambda(x_i - x_{i+1})_+$	$O(q \log n)$	$O(n^2 \log n + n \log q)$ [44]
Q-FL	$\rho_\tau(x_i; a_i)$	$\lambda \lvert x_i - x_{i+1} \rvert$	$O(n \log n)$	LP [35]
Q-wFL	$\rho_\tau(x_i; a_i)$	$\lambda \frac{1}{d_{i,i+1}} \lvert x_i - x_{i+1} \rvert$	$O(n \log n)$	LP [66]
ℓ_1-FL	$w_i \lvert x_i - a_i \rvert$	$\lambda \lvert x_i - x_{i+1} \rvert$	$O(n \log n)$	$O(n^2)$ [98]
PL-wFL-$O(1)$	Convex $O(1)$-piecewise linear	$d_{i,i+1} \lvert x_i - x_{i+1} \rvert$	$O(n \log n)$	$O(n \log n)$ [62]

As can be seen, our GIMR-algorithm's complexity either matches the complexity of the best algorithms to date, or improves on them. And furthermore, GIMR-algorithm is one unified algorithm for all these special cases whereas up till now specialized algorithms were devised for each category of the special cases.

We assess the empirical performance of GIMR-algorithm by comparing our software implementation with Gurobi, a commercial linear programming solver, on simulated data sets of various sizes. The experimental results demonstrate that GIMR-algorithm runs faster than Gurobi on the collection of simulated data sets, by approximately a factor of 10.

Overview

The rest of this chapter is organized as follows: We first provide a brief review of H01-algorithm [44] for MRF-BL (1.2) in Section §5.4. Then we give an overview of GIMR-algorithm in Section §5.5, including additional notations that will help to present the algorithm and its analysis. Then we present GIMR-algorithm in details in two steps: Firstly, in Section §5.6 we present GIMR-algorithm for GIMR with ℓ_1 deviation functions, namely ℓ_1-GIMR-algorithm; then, in Section §5.7, ℓ_1-GIMR-algorithm is generalized to solving GIMR with arbitrary convex piecewise linear deviation functions, namely GIMR-algorithm. Experimental study that assess the performance of GIMR-algorithm is discussed in Section §5.8. Concluding remarks are provided in Section §5.9. The pseudo-codes for the various subroutines used in ℓ_1-GIMR-algorithm and GIMR-algorithm are given in Appendix A and Appendix B.

5.4 Review of H01-Algorithm

Recall that for MRF-BL (1.2), the partial order is represented by a directed graph $G = (V, A)$. H01-algorithm constructs a *parametric graph* $G^{st}(\alpha) = (V_{st}, A_{st})$ associated with $G = (V, A)$, for any scalar value α in union of the ranges of the decision variables, $\bigcup_i [\ell_i, u_i]$. The capacity of arc $(i, j) \in A$ is $c_{i,j} = d_{i,j}$. Each arc in $A_s = \{(s, i)\}_{i \in V}$ has capacity $c_{s,i} = \max\{0, -f_i'(\alpha)\}$ and each arc in $A_t = \{(i, t)\}_{i \in V}$ has capacity $c_{i,t} = \max\{0, f_i'(\alpha)\}$, where $f_i'(\alpha)$ is the right sub-gradient of function $f_i(\cdot)$ at argument α. (One can select instead the left sub-gradient.) Note that for any given value of α, either $c_{s,i} = 0$ or $c_{i,t} = 0$. H01-algorithm finds the minimum cuts in the parametric graph $G^{st}(\alpha)$, for all values of α, in the complexity of a single minimum s, t-cut. The key idea of H01-algorithm is the *threshold theorem* which links the optimal solution of MRF-BL (1.2) with the minimum cut partitions in $G^{st}(\alpha)$:

Theorem 2. (Threshold Theorem, Hochbaum [44]) *For any given α, let S^* be the maximal source set of the minimum cut in graph $G^{st}(\alpha)$. Then there is an optimal solution $\boldsymbol{x}^*$ to MRF-BL (1.2) satisfying $x_i^* \geq \alpha$ if $i \in S^*$ and $x_i^* < \alpha$ if $i \in T^*$.*

An important property of $G^{st}(\alpha)$ is that the capacities of source adjacent arcs are non-increasing functions of α, the capacities of sink adjacent arcs are nondecreasing functions of α, and the capacities of all the other arcs are constants. This implies the following *nested cut property*:

Lemma 3. (Nested Cut Property [39, 44, 50]) *For any two parameter values $\alpha_1 \leq \alpha_2$, let $S^*_{\alpha_1}$ and $S^*_{\alpha_2}$ be the respective maximal source set of the minimum cuts of $G^{st}(\alpha_1)$ and $G^{st}(\alpha_2)$, then $S^*_{\alpha_1} \supseteq S^*_{\alpha_2}$.*

The threshold theorem is used to find an optimal solution to MRF-BL (1.2): For each variable x_i, the largest value of α in $[\ell_i, u_i]$ for which the corresponding node is still in the maximal source set is the optimal value of x_i. This can either be done with binary search, or as in H01-algorithm, all cut partitions for all values of α are identified with the use of a parametric cut procedure.

5.5 Overview of GIMR-Algorithm

The key idea used in our GIMR-algorithm is to adapt the cut-derived threshold theorem of Hochbaum's MRF-BL algorithm. But, instead of using a parametric cut procedure as in the MRF-BL algorithm, our algorithm uses certain properties of the cut function, for the special case of a bi-path graph and convex piecewise linear functions, which lead to a more efficient procedure for computing all relevant cuts. This adaptation runs an order of magnitude faster than the direct application of H01-algorithm to GIMR, as explained in Section §5.2.

The parametric graph $G^{st}(\alpha)$ associated with a bi-path graph G has the capacities of arcs $(i, i+1), (i+1, i) \in A$ as $c_{i,i+1} = d_{i,i+1}$ and $c_{i+1,i} = d_{i+1,i}$ respectively, and the capacities of the source and sink adjacent arcs as $c_{s,i} = \max\{0, -(f_i^{pl})'(\alpha)\}$ and $c_{i,t} = \max\{0, (f_i^{pl})'(\alpha)\}$

respectively. Based on the Threshold Theorem, Theorem 2, it is sufficient to solve the minimum cuts in the parametric graph $G^{st}(\alpha)$ for all values of α, in order to solve GIMR (5.1). We next show that the values of α to be considered can be restricted to the set of breakpoints of the n convex piecewise linear functions, $\{f_i^{pl}(x_i)\}_{i=1,\ldots,n}$. This is proved in the following lemma:

Lemma 4. *The minimum cuts in $G^{st}(\alpha)$ remain unchanged for α assuming any value between any two consecutive breakpoints in the sorted list of breakpoints of all the n convex piecewise linear functions, $\{f_i^{pl}(x_i)\}_{i=1,\ldots,n}$.*

Proof. Recall that only the capacities of the source and sink adjacent arcs depend on the values of α. Since each $f_i^{pl}(x_i)$ is convex piecewise linear in GIMR, thus the source and sink adjacent arc capacities remain constant for α between any two consecutive breakpoint values in the sorted list of breakpoints over all the n convex piecewise linear functions. Therefore the minimum cuts in $G^{st}(\alpha)$ remain unchanged as capacities of all the arcs in the parametric graph are unchanged. □

GIMR-algorithm efficiently computes the minimum cuts of $G^{st}(\alpha)$ for subsequent values of α in the ascending list of breakpoints of all the n convex piecewise linear functions, $\{f_i^{pl}(x_i)\}_{i=1,\ldots,n}$.

Remark 5. *For graphs such as bi-path graphs, a simple, linear complexity, dynamic programming algorithm solves the minimum cut in $G^{st}(\alpha)$ for a fixed value of α. A naive application of this dynamic programming algorithm with setting α equal to each value of the q breakpoints, would render the running time of $O(qn)$ for solving GIMR. Our GIMR-algorithm solves GIMR in $O(q \log n)$ complexity which is significantly faster.*

Additional Notations

We introduce additional notations to facilitate the presentation of GIMR-algorithm.

A convex piecewise linear function $f_i^{pl}(x_i)$ is specified by its ascending list of q_i breakpoints, $a_{i,1} < a_{i,2} < \ldots < a_{i,q_i}$, and the slopes of the $q_i + 1$ linear pieces between every two consecutive breakpoints, denoted by $w_{i,0} < w_{i,1} < \ldots < w_{i,q_i}$. We assume that the n sets of breakpoints are disjoint and that the total number of breakpoints in the union is $q = \sum_{i=1}^{n} q_i$. Note that we make the disjoint breakpoint assumption only for convenience of presenting the algorithm. Our algorithm works in the same way even when a breakpoint is shared by multiple functions, for details see Remark 16 in Section §5.7.

Let the sorted list of the union of q breakpoints of all the n convex piecewise linear functions be $a_{i_1,j_1} < a_{i_2,j_2} < \ldots < a_{i_q,j_q}$, where a_{i_k,j_k}, the kth breakpoint in the sorted list, is the breakpoint between the $(j_k - 1)$th and the j_kth linear pieces of function $f_{i_k}^{pl}(x_{i_k})$.

For $f_i^{pl}(x_i) = w_i|x_i - a_i|$, each function has one breakpoint a_i and two pieces of slopes $-w_i$ and w_i. Thus for this case the sorted list of all the n breakpoints is $a_{i_1} < a_{i_2} < \ldots < a_{i_n}$, where a_{i_k} is the single breakpoint of function $f_{i_k}^{pl}(x_{i_k})$.

Let *interval* $[i, j]$ in G for $i \leq j$ be the subset of V, $\{i, i+1, \ldots, j-1, j\}$. If $i = j$, the interval $[i, i]$ is the singleton i. The notation $[i, j)$ and $(i, j]$ indicate the intervals $[i, j-1]$ and $[i+1, j]$ respectively. Let $[i, j] = \emptyset$ if $i > j$.

For an (S, T) cut, an *s-interval* is the maximal interval containing only s-nodes:

Definition 6. *An interval $[i_\ell, i_r]$ of s-nodes in $G^{st}(\alpha)$ is said to be an* s-interval *if it is not strictly contained in another interval of only s-nodes.*

Node $i_\ell(i_r)$ is said to be the left(right) endpoint of the s-interval $[i_\ell, i_r]$. The definition of s-interval implies that for an s-interval $[i_\ell, i_r]$, if $i_\ell > 1$, then $i_\ell - 1$ is a t-node; if $i_r < n$, then $i_r + 1$ is a t-node.

Let G_k, for $k \geq 1$, be the parametric graph $G^{st}(\alpha)$ for α equal to a_{i_k,j_k}, i.e. $G_k = G^{st}(a_{i_k,j_k})$ ($G_k = G^{st}(a_{i_k})$ for the ℓ_1 special case). For ease of presentation, we introduce $G_0 = G^{st}(a_{i_1,j_1} - \epsilon)$ (or $G_0 = G^{st}(a_{i_1} - \epsilon)$ in the ℓ_1 special case) for a small value of $\epsilon > 0$ (all values of $\epsilon > 0$ generate the same graph G_0). Let (S_k, T_k) be the minimum cut in G_k, for $k \geq 0$. Recall that S_k is the maximal source set.

5.6 ℓ_1-GIMR-Algorithm

This section describes an $O(n \log n)$ algorithm for the GIMR problem with ℓ_1 deviation functions, $f_i^{pl}(x_i) = w_i|x_i - a_i|$ with nonnegative coefficients w_i. This problem is referred to as ℓ_1-GIMR:

$$(\ell_1\text{-GIMR}) \quad \min_{x_1,\ldots,x_n} \sum_{i=1}^{n} w_i|x_i - a_i| + \sum_{i=1}^{n-1} d_{i,i+1}(x_i - x_{i+1})_+ + \sum_{i=1}^{n-1} d_{i+1,i}(x_{i+1} - x_i)_+. \tag{5.10}$$

The algorithm generates the respective minimum cuts of graphs G_k in increasing order of k. Based on the Threshold Theorem, Theorem 2, and the nested cut property, Lemma 3, we know that for each node $j = 1, \ldots, n$, $x_j^* = a_{i_k}$ for the index k such that $j \in S_{k-1}$ and $j \in T_k$.

In G_0, illustrated in Figure 5.1, $c_{s,i} = w_i$ and $c_{i,t} = 0$ for all $i = 1, \ldots, n$. The minimum cut in G_0 is $(\{s\} \cup V, \{t\})$.

For $k \geq 1$ graph G_k is obtained from graph G_{k-1} as follows: Capacity c_{s,i_k} is modified from w_{i_k} to 0 and capacity $c_{i_k,t}$ is modified from 0 to w_{i_k}. Other arcs' capacities remain unchanged. An illustration of G_k for $k \geq 1$ is provided in Figure 5.2.

The key idea of the algorithm is to use a property, proved later, that the minimum cut in G_k is derived by updating the minimum cut in G_{k-1} on an interval of nodes that change their status from s to t. It is then shown that this interval of s-nodes can be found in amortized time $O(\log n)$. With this result, the total running time to solve ℓ_1-GIMR (5.10) is $O(n \log n)$. The remainder of this section is a proof of this main result, stated as Theorem 7:

Theorem 7. *Given the minimum cut (S_{k-1}, T_{k-1}) in G_{k-1}, there is an amortized $O(\log n)$ algorithm for computing the minimum cut (S_k, T_k) in G_k.*

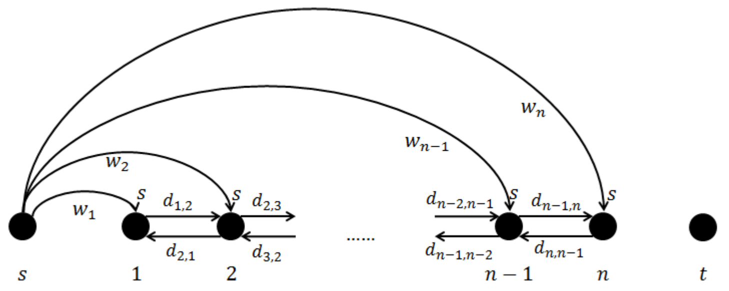


Figure 5.1: The structure of graph G_0. Arcs of capacity 0 are not displayed. Nodes 1 to n are labeled s on top as they are all in the maximal source set S_0 of the minimum cut in G_0.

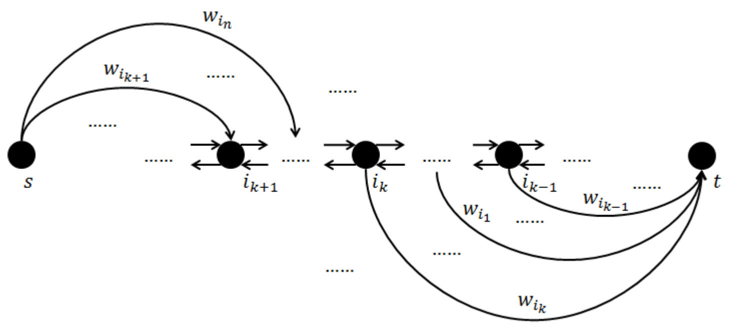


Figure 5.2: The structure of graph G_k. Arcs of capacity 0 are not displayed. Here node $i_k - 1$ appears to the right of i_k and $i_k + 1$ appears to the left, to illustrate that the order of the nodes in the graph, $(1, 2, \dots, n)$, does not necessarily correspond to the order of the nodes according to the subscripts of the sorted breakpoints, $(i_1, i_2, \dots, i_n)$.

Note that the update of the graph from G_{k-1} to G_k involves only a change in the capacities of the source and sink adjacent arcs of i_k. The algorithm proceeds from G_0 to G_n by inspecting in order the nodes $i_1, i_2, \dots, i_n$, the order of which is determined by the sorted list of breakpoints $a_{i_1} < a_{i_2} < \dots < a_{i_n}$. Next we evaluate certain properties of node i_k.

For any node i, if $i \in T_{k-1}$, the nested cut property, Lemma 3, implies that i remains in the sink set for all subsequent cuts (i.e., $status(i) = t$ remains unchanged), and in particular $i \in T_k$. Hence an update of the minimum cut in G_k from the minimum cut in G_{k-1} can only involve shifting some nodes from source set S_{k-1} to sink set T_k (i.e., changing some nodes from s-nodes in G_{k-1} to t-nodes in G_k).

We first demonstrate that if node $i_k \in T_{k-1}$, then (S_{k-1}, T_{k-1}), the minimum cut in G_{k-1}, is also the minimum cut in G_k. This is proved in Lemma 8:

Lemma 8. *If $i_k \in T_{k-1}$ then $(S_k, T_k) = (S_{k-1}, T_{k-1})$.*

Proof. Since $i_k \in T_{k-1}$, by the nested cut property, Lemma 3, $i_k \in T_k$.

The minimum cut in G_{k-1} satisfies:

$$\begin{aligned}
C(\{s\} \cup S_{k-1}, T_{k-1} \cup \{t\}) &= \min_{\emptyset \subseteq S \subseteq V} C(\{s\} \cup S, T \cup \{t\}) \\
= \min_{\emptyset \subseteq S \subseteq V} \Big\{ & C(\{s\}, T \cap ([1, i_k) \cup (i_k, n])) + C(S \cap ([1, i_k) \cup (i_k, n]), \{t\}) \\
& + C(S \cap [1, n], T \cap [1, n]) \Big\} + w_{i_k},
\end{aligned}$$

and the minimum cut in G_k satisfies:

$$\begin{aligned}
C(\{s\} \cup S_k, T_k \cup \{t\}) &= \min_{\emptyset \subseteq S \subseteq V} C(\{s\} \cup S, T \cup \{t\}) \\
= \min_{\emptyset \subseteq S \subseteq V} \Big\{ & C(\{s\}, T \cap ([1, i_k) \cup (i_k, n])) + C(S \cap ([1, i_k) \cup (i_k, n]), \{t\}) \\
& + C(S \cap [1, n], T \cap [1, n]) \Big\} + 0.
\end{aligned}$$

Since the expressions inside the curly brackets are the same for both graphs, it follows that $C(\{s\} \cup S_{k-1}, T_{k-1} \cup \{t\}) - C(\{s\} \cup S_k, T_k \cup \{t\}) = w_{i_k}$, a constant. Therefore the total cut capacities in the two graphs differ by a constant. Recall that the minimum cut is unique as it is the maximal source set minimum cut. Hence the minimizer set S is the same for both G_{k-1} and G_k. □

We conclude that there is no update to the minimum cut in G_k from the minimum cut in G_{k-1} when $i_k \notin S_{k-1}$ (i.e., $i_k \in T_{k-1}$). As proved next, there is still no change to the minimum cut in G_k from the minimum cut in G_{k-1} when i_k is an s-node that does not change its status from G_{k-1} to G_k:

Lemma 9. *If $i_k \in S_{k-1}$ and $i_k \in S_k$, then $(S_k, T_k) = (S_{k-1}, T_{k-1})$.*

Proof. The minimum cut in G_{k-1} satisfies:

$$\begin{aligned}
C(\{s\} \cup S_{k-1}, T_{k-1} \cup \{t\}) &= \min_{\emptyset \subseteq S \subseteq V} C(\{s\} \cup S, T \cup \{t\}) \\
= \min_{\emptyset \subseteq S \subseteq V} \Big\{ & C(\{s\}, T \cap ([1, i_k) \cup (i_k, n])) + C(S \cap ([1, i_k) \cup (i_k, n]), \{t\}) \\
& + C(S \cap [1, n], T \cap [1, n]) \Big\} + 0.
\end{aligned}$$

And the minimum cut in G_k satisfies:

$$C(\{s\}\cup S_k, T_k\cup\{t\}) = \min_{\emptyset\subseteq S\subseteq V} C(\{s\}\cup S, T\cup\{t\})$$

$$= \min_{\emptyset\subseteq S\subseteq V}\Big\{C(\{s\}, T\cap([1,i_k)\cup(i_k,n])) + C(S\cap([1,i_k)\cup(i_k,n]),\{t\}) + C(S\cap[1,n], T\cap[1,n])\Big\} + w_{i_k}.$$

Therefore, $C(\{s\}\cup S_{k-1}, T_{k-1}\cup\{t\}) - C(\{s\}\cup S_k, T_k\cup\{t\}) = -w_{i_k}$, and hence the minimizer set S is the same for both G_{k-1} and G_k. □

These two lemmas imply that the only case that will involve an update to the minimum cut in G_k is when $i_k\in S_{k-1}$ yet $i_k\in T_k$, i.e., when node i_k changes its status from an s-node in G_{k-1} to a t-node in G_k. It is shown next that in this case, if there is any node $j<i_k$ (on the left of i_k) that does not change its status from G_{k-1} to G_k (i.e., either j is an s-node in both G_{k-1} and G_k or j is a t-node in both G_{k-1} and G_k), then all nodes in the interval $[1,j]$ do not change their status from G_{k-1} to G_k; similarly, if there is any node $j'>i_k$ (on the right of i_k) that does not change its status from G_{k-1} to G_k, then all nodes in the interval $[j',n]$ do not change their status from G_{k-1} to G_k. This is proved formally in Lemma 10:

Lemma 10. *Suppose that $i_k\in S_{k-1}$ and $i_k\in T_k$.*

(a) If there is a node $j<i_k$ that does not change its status from G_{k-1} to G_k (i.e., either j is an s-node in both G_{k-1} and G_k or j is a t-node in both G_{k-1} and G_k), then all nodes in $[1,j]$ do not change their status from G_{k-1} to G_k;

(b) If there is a node $j'>i_k$ that does not change its status from G_{k-1} to G_k, then all nodes in $[j',n]$ do not change their status from G_{k-1} to G_k.

Proof. (a) The minimum cut in G_{k-1} satisfies:

$$C(\{s\}\cup S_{k-1}, T_{k-1}\cup\{t\}) = \min_{\emptyset\subseteq S\subseteq V} C(\{s\}\cup S, T\cup\{t\})$$

$$= \min_{\emptyset\subseteq S\subseteq V}\Big\{C(\{s\}\cup(S\cap[1,j]), (T\cap[1,j])\cup\{t\}) + C(S\cap[j,n], T\cap[j,n]) + C(\{s\}, T\cap(j,n]) + C(S\cap(j,n],\{t\})\Big\}$$

$$= \min_{S\cap[1,j]} C(\{s\}\cup(S\cap[1,j]), (T\cap[1,j])\cup\{t\}) + \min_{S\cap[j,n]}\Big\{C(S\cap[j,n], T\cap[j,n]) + C(\{s\}, T\cap(j,n]) + C(S\cap(j,n],\{t\})\Big\}.$$

And the minimum cut in G_k satisfies:

$$C(\{s\} \cup S_k, T_k \cup \{t\}) = \min_{\emptyset \subseteq S \subseteq V} C(\{s\} \cup S, T \cup \{t\})$$

$$= \min_{\emptyset \subseteq S \subseteq V} \Big\{ C(\{s\} \cup (S \cap [1,j]), (T \cap [1,j]) \cup \{t\})$$

$$+ C(S \cap [j,n], T \cap [j,n]) + C(\{s\}, T \cap (j,n]) + C(S \cap (j,n], \{t\}) \Big\}$$

$$= \min_{S \cap [1,j]} C(\{s\} \cup (S \cap [1,j]), (T \cap [1,j]) \cup \{t\})$$

$$+ \min_{S \cap [j,n]} \Big\{ C(S \cap [j,n], T \cap [j,n]) + C(\{s\}, T \cap (j,n]) + C(S \cap (j,n], \{t\}) \Big\}.$$

Since the arc capacities other than (s, i_k) and (i_k, t) in these two cut capacities are respectively the same, and the status of j does not change from G_{k-1} to G_k, the expressions $C(\{s\} \cup (S \cap [1,j]), (T \cap [1,j]) \cup \{t\})$ are of the same value for both graphs. As a result, the minimizer set $S \cap [1,j]$ is the same for both G_{k-1} and G_k.

(b) The minimum cut in G_{k-1} satisfies:

$$C(\{s\} \cup S_{k-1}, T_{k-1} \cup \{t\}) = \min_{\emptyset \subseteq S \subseteq V} C(\{s\} \cup S, T \cup \{t\})$$

$$= \min_{\emptyset \subseteq S \subseteq V} \Big\{ C(\{s\}, T \cap [1,j')) + C(S \cap [1,j'), \{t\}) + C(S \cap [1,j'], T \cap [1,j'])$$

$$+ C(\{s\} \cup (S \cap [j',n]), (T \cap [j',n]) \cup \{t\}) \Big\}$$

$$= \min_{S \cap [1,j']} \Big\{ C(\{s\}, T \cap [1,j')) + C(S \cap [1,j'), \{t\}) + C(S \cap [1,j'], T \cap [1,j']) \Big\}$$

$$+ \min_{S \cap [j',n]} C(\{s\} \cup (S \cap [j',n]), (T \cap [j',n]) \cup \{t\}).$$

And the minimum cut in G_k satisfies:

$$C(\{s\} \cup S_k, T_k \cup \{t\}) = \min_{\emptyset \subseteq S \subseteq V} C(\{s\} \cup S, T \cup \{t\})$$

$$= \min_{\emptyset \subseteq S \subseteq V} \Big\{ C(\{s\}, T \cap [1,j')) + C(S \cap [1,j'), \{t\}) + C(S \cap [1,j'], T \cap [1,j'])$$

$$+ C(\{s\} \cup (S \cap [j',n]), (T \cap [j',n]) \cup \{t\}) \Big\}$$

$$= \min_{S \cap [1,j']} \Big\{ C(\{s\}, T \cap [1,j')) + C(S \cap [1,j'), \{t\}) + C(S \cap [1,j'], T \cap [1,j']) \Big\}$$

$$+ \min_{S \cap [j',n]} C(\{s\} \cup (S \cap [j',n]), (T \cap [j',n]) \cup \{t\}).$$

Since the arc capacities other than (s, i_k) and (i_k, t) in these two cut capacities are respectively the same, and the status of j' does not change from G_{k-1} to G_k, the expressions $C(\{s\} \cup (S \cap [j', n]), (T \cap [j', n]) \cup \{t\})$ are of the same value for both graphs. As a result, the minimizer set $S \cap [j', n]$ is the same for both G_{k-1} and G_k. □

Lemma 10 is illustrated in Figure 5.3.

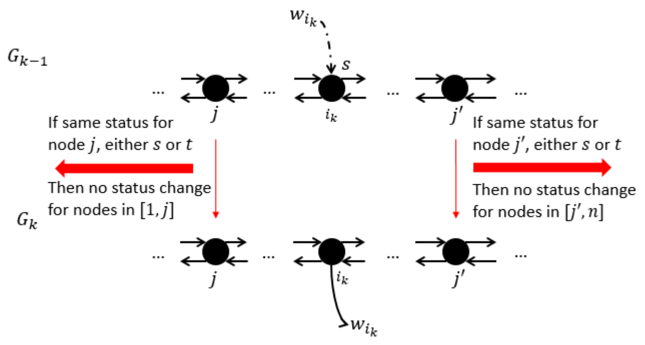

w_{i_k}
G_{k-1}
s
j
i_k
j'
If same status for node j, either s or t
Then no status change for nodes in $[1, j]$
If same status for node j', either s or t
Then no status change for nodes in $[j', n]$
G_k
j
i_k
j'
w_{i_k}

Figure 5.3: Illustration of Lemma 10. $i_k \in S_{k-1}$ (labeled s on top). If there is a node $j < i_k$ that does not change its status from G_{k-1} to G_k (i.e., either j is an s-node in both G_{k-1} and G_k or j is a t-node in both G_{k-1} and G_k), then all nodes in $[1, j]$ do not change their status from G_{k-1} to G_k; if there is a node $j' > i_k$ that does not change its status from G_{k-1} to G_k, then all nodes in $[j', n]$ do not change their status from G_{k-1} to G_k.

If there is a non-empty set of nodes that change their status from s in G_{k-1} to t in G_k, it must include i_k (otherwise by Lemma 8 and Lemma 9, none of the nodes changes its status, which is a contradiction). Among all the nodes in $V = [1, n]$ that change from s in G_{k-1} to t in G_k, we denote the smallest node index as i^*_{k1} and the largest node index as i^*_{k2}, thus $i^*_{k1} \leq i_k \leq i^*_{k2}$. All nodes in the interval $[i^*_{k1}, i^*_{k2}]$ must change their status from s in G_{k-1} to t in G_k, because if there is a node $j \in [i^*_{k1}, i^*_{k2}] \setminus \{i_k\}$ whose status does not change, then Lemma 10 implies that either the status of i^*_{k1} does not change (when $j < i_k$) or the status of i^*_{k2} does not change (when $j > i_k$), which contradicts the choice of these nodes as nodes that do change their status. We conclude that if i_k changes its status, then all nodes that change their status form an interval of s-nodes containing i_k. This is stated in the following corollary:

Corollary 11. *If $i_k \in S_{k-1}$, then all the nodes that change their status from s in G_{k-1} to t in G_k must form a (possibly empty) interval of s-nodes containing i_k in G_{k-1}.*

Corollary 11 is illustrated in Figure 5.4. Note that $[i^*_{k1}, i^*_{k2}]$ is a sub-interval of the s-interval w.r.t. node i_k in G_{k-1}, $[i_{k\ell}, i_{kr}]$. Using Corollary 11 the problem of computing the minimum cut in G_k from the minimum cut in G_{k-1} is reduced to the problem of identifying the interval $[i^*_{k1}, i^*_{k2}]$.

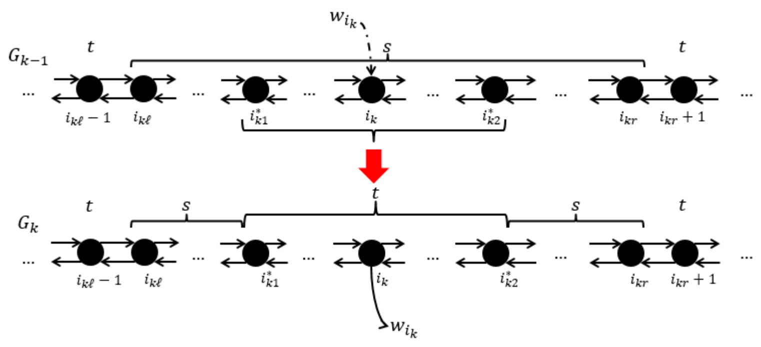


Figure 5.4: Illustration of Corollary 11. $i_k \in S_{k-1}$. Nodes are labeled on top s if they are s-nodes in G_{k-1} or G_k. Nodes are labeled on top t if they are t-nodes in G_{k-1} or G_k. All s-nodes in $[i^*_{k1}, i^*_{k2}]$ (possibly empty) in G_{k-1}, containing i_k, change to t-nodes in G_k. Note that $[i^*_{k1}, i^*_{k2}]$ is a sub-interval of the s-interval w.r.t. node i_k in G_{k-1}, $[i_{k\ell}, i_{kr}]$.

Finding Node Status Change Interval

To identify the node status change interval $[i^*_{k1}, i^*_{k2}]$, we have the following lemma:

Lemma 12. *The node status change interval $[i^*_{k1}, i^*_{k2}]$ is the optimal solution to the following optimization problem for G_k:*

$$\begin{aligned} \min_{[i_{k1},i_{k2}]} \; & C(\{s\}, [i_{k1}, i_{k2}]) + C([i_{k\ell}, i_{kr}] \setminus [i_{k1}, i_{k2}], \{t\}) \\ & + C([i_{k\ell}, i_{kr}] \setminus [i_{k1}, i_{k2}], [i_{k1}, i_{k2}] \cup \{i_{k\ell}-1, i_{kr}+1\}) \\ s.t. \; & [i_{k1}, i_{k2}] = \emptyset \text{ or } i_k \in [i_{k1}, i_{k2}] \subseteq [i_{k\ell}, i_{kr}] \end{aligned} \tag{5.11}$$

Proof. The minimum cut (S_k, T_k) in G_k satisfies:

$$\begin{aligned} C(\{s\} \cup S_k, T_k \cup \{t\}) &= \min_{(S,T)} C(\{s\} \cup S, T \cup \{t\}) \\ = \min_{(S,T)} \Big\{ & C(\{s\} \cup (S \cap [1, i_{k\ell}-1]), (T \cap [1, i_{k\ell}-1]) \cup \{t\}) \\ & + C(\{s\}, T \cap [i_{k\ell}, i_{kr}]) + C(S \cap [i_{k\ell}, i_{kr}], \{t\}) \\ & + C(S \cap [i_{k\ell}-1, i_{kr}+1], T \cap [i_{k\ell}-1, i_{kr}+1]) \\ & + C(\{s\} \cup (S \cap [i_{kr}+1, n]), (T \cap [i_{kr}+1, n]) \cup \{t\}) \Big\} \end{aligned}$$

This minimization problem can be written as the sum of three minimization problems:

$$= \min_{S\cap[1,i_{k\ell}-1]} C(\{s\}\cup(S\cap[1,i_{k\ell}-1]),(T\cap[1,i_{k\ell}-1])\cup\{t\}) \tag{5.12}$$

$$+ \min_{T\cap[i_{k\ell},i_{kr}]} \Big\{ C(\{s\},T\cap[i_{k\ell},i_{kr}]) + C(S\cap[i_{k\ell},i_{kr}],\{t\}) \tag{5.13}$$

$$+ C(S\cap[i_{k\ell}-1,i_{kr}+1],T\cap[i_{k\ell}-1,i_{kr}+1]) \Big\}$$

$$+ \min_{S\cap[i_{kr}+1,n]} C(\{s\}\cup(S\cap[i_{kr}+1,n]),(T\cap[i_{kr}+1,n])\cup\{t\}). \tag{5.14}$$

By Corollary 11, the minimizer set $S\cap[1,i_{k\ell}-1]$ in subproblem (5.12) is $S_{k-1}\cap[1,i_{k\ell}-1]$ and the minimizer set $S\cap[i_{kr}+1,n]$ in subproblem (5.14) is $S_{k-1}\cap[i_{kr}+1,n]$.

It remains to solve subproblem (5.13). Recall that since $i_{k\ell}-1$ and $i_{kr}+1$ are outside an s-interval, they are both t-nodes in G_k, therefore, $S\cap[i_{k\ell}-1,i_{kr}+1] = S\cap[i_{k\ell},i_{kr}]$. For $[i_{k1},i_{k2}]\subseteq[i_{k\ell},i_{kr}]$ the interval of nodes that change their status from s to t, that must contain i_k if non-empty (Corollary 11, $S\cap[i_{k\ell},i_{kr}] = [i_{k\ell},i_{kr}]\setminus[i_{k1},i_{k2}]$, and $T\cap[i_{k\ell},i_{kr}] = [i_{k1},i_{k2}]$). An interval $[i_{k1},i_{k2}]\subseteq[i_{k\ell},i_{kr}]$ is said to be *feasible* if it is either empty, $[i_{k1},i_{k2}]=\emptyset$, or else it contains i_k, $i_k\in[i_{k1},i_{k2}]$. The interval $[i^*_{k1},i^*_{k2}]$ is optimal if it is the feasible interval that minimizes the objective value of subproblem (5.13).

For any feasible interval $[i_{k1},i_{k2}]\subseteq[i_{k\ell},i_{kr}]$, we can re-write the terms in subproblem (5.13) as:

$$S\cap[i_{k\ell}-1,i_{kr}+1] = S\cap[i_{k\ell},i_{kr}] = [i_{k\ell},i_{kr}]\setminus[i_{k1},i_{k2}], \quad \text{and}$$
$$T\cap[i_{k\ell}-1,i_{kr}+1] = (T\cap[i_{k\ell},i_{kr}])\cup\{i_{k\ell}-1,i_{kr}+1\} = [i_{k1},i_{k2}]\cup\{i_{k\ell}-1,i_{kr}+1\}.$$

Substituting for these expressions in subproblem (5.13), it is rewritten as:

$$\begin{aligned} \min_{[i_{k1},i_{k2}]} \; & C(\{s\},[i_{k1},i_{k2}]) + C([i_{k\ell},i_{kr}]\setminus[i_{k1},i_{k2}],\{t\}) \\ & + C([i_{k\ell},i_{kr}]\setminus[i_{k1},i_{k2}],[i_{k1},i_{k2}]\cup\{i_{k\ell}-1,i_{kr}+1\}) \\ s.t.\; & [i_{k1},i_{k2}]=\emptyset \text{ or } i_k\in[i_{k1},i_{k2}]\subseteq[i_{k\ell},i_{kr}] \end{aligned}$$

This completes the proof. □

Next, we discuss how to solve the optimization problem (5.11). In the following equations, let $d_{0,1}=d_{1,0}=d_{n,n+1}=d_{n+1,n}=0$.

We evaluate the objective value of problem (5.11) for an empty interval solution, and compare it to the objective value of the optimal nonempty interval solution. The one that gives smaller objective value is the optimal solution to problem (5.11). For $[i_{k1},i_{k2}]=\emptyset$, the objective value of problem (5.11) is:

$$Z(\emptyset) \triangleq C([i_{k\ell},i_{kr}],\{i_{k\ell}-1,i_{kr}+1,t\}) = \sum_{i=i_{k\ell}}^{i_{kr}} c_{i,t} + d_{i_{k\ell},i_{k\ell}-1} + d_{i_{kr},i_{kr}+1}. \tag{5.15}$$

Let $[\hat{i}_{k1}, \hat{i}_{k2}]$ be the optimal solution to the problem (5.11) restricted to nonempty intervals. Let the value of the objective function of problem (5.11) for $[\hat{i}_{k1}, \hat{i}_{k2}]$ be $Z([\hat{i}_{k1}, \hat{i}_{k2}])$. If $Z([\hat{i}_{k1}, \hat{i}_{k2}]) < Z(\emptyset)$ then the optimal solution is $[i^*_{k1}, i^*_{k2}] = [\hat{i}_{k1}, \hat{i}_{k2}]$, otherwise $[i^*_{k1}, i^*_{k2}] = \emptyset$.

We next demonstrate that $\hat{i}_{k1}$ and $\hat{i}_{k2}$ can be found by solving two independent optimization problems:

Lemma 13. *The optimal nonempty interval solution, $[\hat{i}_{k1}, \hat{i}_{k2}]$, can be found by solving two independent minimization problems for i_{k1} and i_{k2} respectively.*

Proof. Problem (5.11) that restricted to nonempty intervals is equivalent to

$$\begin{aligned}
&\min_{\substack{i_{k1}: i_{k\ell} \le i_{k1} \le i_k \\ i_{k2}: i_k \le i_{k2} \le i_{kr}}} \Big\{ C(\{s\}, [i_{k1}, i_k]) + C([i_{k\ell}, i_{k1}), \{t\}) + C([i_{k\ell}, i_{k1}), [i_{k1}, i_k] \cup \{i_{k\ell} - 1\}) \\
&\qquad + C(\{s\}, (i_k, i_{k2}]) + C((i_{k2}, i_{kr}], \{t\}) + C((i_{k2}, i_{kr}], [i_k, i_{k2}] \cup \{i_{kr} + 1\}) \Big\} \\
&= \min_{i_{k1}: i_{k\ell} \le i_{k1} \le i_k} \Big\{ C(\{s\}, [i_{k1}, i_k]) + C([i_{k\ell}, i_{k1}), \{t\}) + C([i_{k\ell}, i_{k1}), [i_{k1}, i_k] \cup \{i_{k\ell} - 1\}) \Big\} \\
&+ \min_{i_{k2}: i_k \le i_{k2} \le i_{kr}} \Big\{ C(\{s\}, (i_k, i_{k2}]) + C((i_{k2}, i_{kr}], \{t\}) + C((i_{k2}, i_{kr}], [i_k, i_{k2}] \cup \{i_{kr} + 1\}) \Big\} \\
&\triangleq \min_{i_{k1}: i_{k\ell} \le i_{k1} \le i_k} \big\{ f_1(i_{k1}) \big\} + \min_{i_{k2}: i_k \le i_{k2} \le i_{kr}} \big\{ f_2(i_{k2}) \big\}.
\end{aligned}$$

Hence $\hat{i}_{k1}$ is found by solving the optimization $\min_{i_{k1}: i_{k\ell} \le i_{k1} \le i_k} \{f_1(i_{k1})\}$ and $\hat{i}_{k2}$ is found by solving the optimization problem $\min_{i_{k2}: i_k \le i_{k2} \le i_{kr}} \{f_2(i_{k2})\}$. □

We first show how to solve problem $\min_{i_{k1}: i_{k\ell} \le i_{k1} \le i_k} \{f_1(i_{k1})\}$. We evaluate the objective value $f_1(i_{k1})$ for $i_{k1} = i_{k\ell}$, and compare it to the optimal objective value for $i_{k1} \in [i_{k\ell} + 1, i_k]$. The one that gives smaller objective value is the optimal solution $\hat{i}_{k1}$. For $i_{k1} = i_{k\ell}$, the objective value $f_1(i_{k\ell})$ is:

$$f_1(i_{k\ell}) = \sum_{i=i_{k\ell}}^{i_k} c_{s,i}. \tag{5.16}$$

For $i_{k1} \in [i_{k\ell} + 1, i_k]$, we have the following equation to express the objective value $f_1(i_{k1})$:

$$f_1(i_{k1}) = \sum_{i=i_{k\ell}}^{i_{k1}-1} c_{i,t} + \sum_{i=i_{k1}}^{i_k} c_{s,i} + d_{i_{k\ell}, i_{k\ell}-1} + d_{i_{k1}-1, i_{k1}}. \tag{5.17}$$

Let $\tilde{i}_{k1}$ be the minimizer of $f_1(i_{k1})$ for $i_{k1} \in [i_{k\ell} + 1, i_k]$. If there are multiple minima, $\tilde{i}_{k1}$ takes the largest index, due to the maximal source set requirement. To summarize,

$$\hat{i}_{k1} = \begin{cases} \tilde{i}_{k1}, & \text{if } i_{k\ell} + 1 \le i_k \text{ and } f_1(\tilde{i}_{k1}) \le f_1(i_{k\ell}) \\ i_{k\ell}, & \text{otherwise.} \end{cases}$$

We next show how to solve problem $\min_{i_{k2}:i_k \leq i_{k2} \leq i_{kr}}\{f_2(i_{k2})\}$. We evaluate the objective value $f_2(i_{k2})$ for $i_{k2} = i_{kr}$, and compare it to the optimal objective value for $i_{k2} \in [i_k, i_{kr}-1]$. The one that gives smaller objective value is the optimal solution $\hat{i}_{k2}$. For $i_{k2} = i_{kr}$, the objective value $f_2(i_{kr})$ is

$$f_2(i_{kr}) = \sum_{i=i_k+1}^{i_{kr}} c_{s,i}. \tag{5.18}$$

For $i_{k2} \in [i_k, i_{kr} - 1]$, we have the following equation to express the objective value $f_2(i_{k2})$:

$$f_2(i_{k2}) = \sum_{i=i_k+1}^{i_{k2}} c_{s,i} + \sum_{i=i_{k2}+1}^{i_{kr}} c_{i,t} + d_{i_{kr},i_{kr}+1} + d_{i_{k2}+1,i_{k2}}. \tag{5.19}$$

Let $\tilde{i}_{k2}$ be the minimizer of $f_2(i_{k2})$ for $i_{k2} \in [i_k, i_{kr} - 1]$. If there are multiple minima, $\tilde{i}_{k2}$ takes the smallest index, due to the maximal source set requirement. To summarize,

$$\hat{i}_{k2} = \begin{cases} \tilde{i}_{k2}, & \text{if } i_k \leq i_{kr} - 1 \text{ and } f_2(\tilde{i}_{k2}) \leq f_2(i_{kr}) \\ i_{kr}, & \text{otherwise.} \end{cases}$$

Finally the value of the objective function of problem (5.11) for $[\hat{i}_{k1}, \hat{i}_{k2}]$ is

$$Z([\hat{i}_{k1}, \hat{i}_{k2}]) = f_1(\hat{i}_{k1}) + f_2(\hat{i}_{k2}).$$

Data Structure to Find Node Status Change Interval Efficiently

We observe that Equations (5.15) to (5.19) share two operations, one is sum of capacities of source adjacent arcs of nodes in an interval $[i, j]$, $\sum_{i'=i}^{j} c_{s,i'}$, and the other is sum of capacities of sink adjacent arcs of nodes in an interval $[i, j]$, $\sum_{i'=i}^{j} c_{i',t}$. It will be convenient to rewrite these two sums of capacities as:

$$\sum_{i'=i}^{j} c_{s,i'} = \sum_{i'=1}^{j} c_{s,i'} - \sum_{i'=1}^{i-1} c_{s,i'},$$
$$\sum_{i'=i}^{j} c_{i',t} = \sum_{i'=1}^{j} c_{i',t} - \sum_{i'=1}^{i-1} c_{i',t}.$$

To derive these sums easily, we maintain two arrays, $(sa(i))_{i=0,1,\ldots,n}$ and $(ta(i))_{i=0,1,\ldots,n}$. $sa(i)$ is the sum of capacities of source adjacent arcs of nodes in $[1, i]$ and $ta(i)$ is the sum of capacities of sink adjacent arcs of nodes in $[1, i]$. Formally:

$$sa(0) = 0; sa(i) = C(\{s\}, [1, i]) = \sum_{j=1}^{i} c_{s,j} \ (i = 1, \ldots, n); \tag{5.20}$$

$$ta(0) = 0; ta(i) = C([1, i], \{t\}) = \sum_{j=1}^{i} c_{j,t} \ (i = 1, \ldots, n). \tag{5.21}$$

Note that both arrays can also be defined recursively as

$$\begin{aligned} &sa(0) = 0; sa(i) = sa(i-1) + c_{s,i} \ (i = 1, \ldots, n); \\ &ta(0) = 0; ta(i) = ta(i-1) + c_{i,t} \ (i = 1, \ldots, n). \end{aligned}$$

The two arrays, along with two others to be introduced, will be used throughout the algorithm.

Equations (5.15), (5.16) and (5.18) in terms of the two arrays result in:

$$\begin{aligned} Z(\emptyset) &= \sum_{i=i_{k\ell}}^{i_{kr}} c_{i,t} + d_{i_{k\ell},i_{k\ell}-1} + d_{i_{kr},i_{kr}+1} = ta(i_{kr}) - ta(i_{k\ell}-1) + d_{i_{k\ell},i_{k\ell}-1} + d_{i_{kr},i_{kr}+1} \\ f_1(i_{k\ell}) &= \sum_{i=i_{k\ell}}^{i_k} c_{s,i} = sa(i_k) - sa(i_{k\ell}-1). \\ f_2(i_{kr}) &= \sum_{i=i_k+1}^{i_{kr}} c_{s,i} = sa(i_{kr}) - sa(i_k). \end{aligned}$$

Equation (5.17) can be rewritten as:

$$\begin{aligned} f_1(i_{k1}) &= \sum_{i=i_{k\ell}}^{i_{k1}-1} c_{i,t} + \sum_{i=i_{k1}}^{i_k} c_{s,i} + d_{i_{k\ell},i_{k\ell}-1} + d_{i_{k1}-1,i_{k1}} \\ &= \big(ta(i_{k1}-1) - ta(i_{k\ell}-1)\big) + \big(sa(i_k) - sa(i_{k1}-1)\big) + d_{i_{k\ell},i_{k\ell}-1} + d_{i_{k1}-1,i_{k1}}. \end{aligned}$$

Next we introduce a third array $(tms(i))_{i=0,1,\ldots,n}$ defined as:

$$tms(i) = ta(i) - sa(i) + d_{i,i+1} \ (i = 0, 1, \ldots, n). \tag{5.22}$$

With these arrays Equation (5.17) can be simplified to:

$$f_1(i_{k1}) = tms(i_{k1}-1) - ta(i_{k\ell}-1) + sa(i_k) + d_{i_{k\ell},i_{k\ell}-1}.$$

Recall that the above equation is the expression of $f_1(i_{k1})$ for $i_{k1} \in [i_{k\ell}+1, i_k]$, and we want to find the minimizer of $f_1(i_{k1})$ for $i_{k1} \in [i_{k\ell}+1, i_k]$. The only term in $f_1(i_{k1})$ that depends on i_{k1} is $tms(i_{k1}-1)$, thus the minimizer $\tilde{i}_{k1}$ of $f_1(i_{k1})$, is also the minimizer of $tms(i_{k1}-1)$, for $i_{k1} \in [i_{k\ell}+1, i_k]$:

$$\tilde{i}_{k1} = \text{argmin}_{i_{k1}: i_{k\ell}+1 \leq i_{k1} \leq i_k} \big\{ f_1(i_{k1}) \big\} = \text{argmin}_{i_{k1}: i_{k\ell}+1 \leq i_{k1} \leq i_k} \big\{ tms(i_{k1}-1) \big\}. \tag{5.23}$$

Similarly, for $f_2(i_{k2})$ ($i_{k2} \in [i_k, i_{kr}-1]$), Equation (5.19) can be rewritten as:

$$\begin{aligned} f_2(i_{k2}) &= \sum_{i=i_k+1}^{i_{k2}} c_{s,i} + \sum_{i=i_{k2}+1}^{i_{kr}} c_{i,t} + d_{i_{kr},i_{kr}+1} + d_{i_{k2}+1,i_{k2}} \\ &= \big(sa(i_{k2}) - sa(i_k)\big) + \big(ta(i_{kr}) - ta(i_{k2})\big) + d_{i_{kr},i_{kr}+1} + d_{i_{k2}+1,i_{k2}}. \end{aligned}$$

A final, fourth, array $(smt(i))_{i=0,1,\ldots,n}$ is:

$$smt(i) = sa(i) - ta(i) + d_{i+1,i} \ (i = 0, 1, \ldots, n). \tag{5.24}$$

Then Equation (5.19) can be further simplified to:

$$f_2(i_{k2}) = smt(i_{k2}) - sa(i_k) + ta(i_{kr}) + d_{i_{kr},i_{kr}+1}.$$

Recall that the above equation is the expression of $f_2(i_{k2})$ for $i_{k2} \in [i_k, i_{kr} - 1]$, and we want to find the minimizer of $f_2(i_{k2})$ for $i_{k2} \in [i_k, i_{kr} - 1]$. The only term in $f_2(i_{k2})$ that depends on i_{k2} is $smt(i_{k2})$, thus the minimizer $\tilde{i}_{k2}$ of $f_2(i_{k2})$, is also the minimizer of $smt(i_{k2})$, for $i_{k2} \in [i_k, i_{kr} - 1]$:

$$\tilde{i}_{k2} = \text{argmin}_{i_{k2}:i_k \leq i_{k2} \leq i_{kr}-1}\{f_2(i_{k2})\} = \text{argmin}_{i_{k2}:i_k \leq i_{k2} \leq i_{kr}-1}\{smt(i_{k2})\}. \tag{5.25}$$

To summarize, we introduced here four arrays: $(sa(i))_{i=0,1,\ldots,n}$ in (5.20), $(ta(i))_{i=0,1,\ldots,n}$ in (5.21), $(tms(i))_{i=0,1,\ldots,n}$ in (5.22) and $(smt(i))_{i=0,1,\ldots,n}$ in (5.24). With the four arrays, $\tilde{i}_{k1}$ is identified by finding the minimum value of a subarray of array $(tms(i))_{i=0,1,\ldots,n}$ according to (5.23), $\tilde{i}_{k2}$ is identified by finding the minimum value of a subarray of array $(smt(i))_{i=0,1,\ldots,n}$ according to (5.25). After we identify $\tilde{i}_{k1}$ and $\tilde{i}_{k2}$, we evaluate and compare $f_1(i_{k\ell})$ to $f_1(\tilde{i}_{k1})$ to identify $\hat{i}_{k1}$, and evaluate and compare $f_2(i_{kr})$ to $f_2(\tilde{i}_{k2})$ to identify $\hat{i}_{k2}$. The process also gives us the objective value of $Z([\hat{i}_{k1}, \hat{i}_{k2}])$. Finally, we evaluate $Z(\emptyset)$ and compare it to $Z([\hat{i}_{k1}, \hat{i}_{k2}])$ to identify $[i^*_{k1}, i^*_{k2}]$. Evaluating all the above objective values involves querying different specific elements of the four arrays.

In our algorithm, we implement the four arrays using a data structure introduced in Appendix B. Using the data structure, the operations of identifying the minimum value of any subarray of an array; querying a specific element of an array; updating the arrays from graph G_{k-1} to G_k, can all be done efficiently, in complexity of $O(\log n)$ per operation. Therefore the node status change interval $[i^*_{k1}, i^*_{k2}]$ and the array updates can be computed efficiently.

The Complete ℓ_1-GIMR-Algorithm

To summarize, the algorithm proceeds from G_{k-1} to G_k, by checking the status of node i_k. If this node is an s-node then it is possible that the minimum cut in G_{k-1} is changed when the graph is updated to G_k. In that case, the algorithm identifies the node status change interval with respect to i_k, $[i^*_{k1}, i^*_{k2}]$. If this interval is not empty then the nodes in the interval change their status from s to t. This triggers a change in the set of s-intervals of G_{k-1} by decomposing one s-interval to up to two new s-intervals in G_k. Once the s-intervals have been updated, the iteration for k terminates.

We next present the pseudo-code for ℓ_1-GIMR-algorithm, followed by explanation of the subroutines used:

```
$\ell_1$-GIMR-Algorithm
input: $\{w_i, a_i\}_{i=1,\ldots,n}$ and $\{d_{i,i+1}, d_{i+1,i}\}_{i=1,\ldots,n-1}$ in $\ell_1$-GIMR (5.10).
output: An optimal solution $\{x_i^*\}_{i=1,\ldots,n}$.
begin
1     Sort the $a_i$s as $a_{i_1} < a_{i_2} < \ldots < a_{i_n}$;
2     initialization();
3     for $k := 1, \ldots, n$:
4         {Update graph}update_arrays($i_k, -w_{i_k}, w_{i_k}$);
5         if $status(i_k) = s$ then
6             $[i_{k\ell}, i_{kr}]$ := get_s_interval($i_k$);
7             $[i^*_{k1}, i^*_{k2}]$ := find_status_change_interval($i_{k\ell}, i_k, i_{kr}$);
8             if $[i^*_{k1}, i^*_{k2}] \neq \emptyset$ then
9                 for $i \in [i^*_{k1}, i^*_{k2}]$: $x_i^* := a_{i_k}$, $status(i) := t$;
10                update_s_interval($i_{k\ell}, i^*_{k1}, i^*_{k2}, i_{kr}$);
11            end if
12        end if
13    end for
14    return $\{x_i^*\}_{i=1,\ldots,n}$;
end
```

At line 2, We use initialization() to initialize all the data structures for G_0 that are needed in the algorithm, including the set of s-intervals, the four arrays, and the status of all the nodes. The data structure for the set of s-intervals is introduced in Appendix A. G_0 contains a single s-interval $[1, n]$. Appendix A.1 shows that initializing the set of s-intervals containing a single s-interval $[1, n]$ is done in $O(1)$ time using the data structure. The data structure for the four arrays is introduced in Appendix B. Appendix B.1 shows that initializing the four arrays for G_0 is done in time $O(n \log n)$ using the data structure. We implement the status of all the nodes as a simple boolean array such that $status(i)$ is the status of node i for $i = 1, \ldots, n$. As all nodes are in the maximal source set in the minimum cut in G_0, thus initially $status(i) = s$ for all $i = 1, \ldots, n$, which is initialized in $O(n)$ time. Hence the complexity of initialization() is $O(n \log n)$.

At line 3, the **for** loop computes, in the kth iteration, the minimum cut in G_k from the minimum cut in G_{k-1}. At line 4 we first call subroutine update_arrays($i_k, -w_{i_k}, w_{i_k}$) to update the values of the four arrays from G_{k-1} to G_k. Recall that graph G_k is obtained from graph G_{k-1} by changing only the capacities c_{s,i_k} and $c_{i_k,t}$. Thus the values of the four arrays are updated as follows:

$$\begin{aligned}
&\forall i \in [i_k, n] : \\
&\quad sa(i) := sa(i) - w_{i_k}; \\
&\quad ta(i) := ta(i) + w_{i_k}; \\
&\quad tms(i) := tms(i) + 2w_{i_k}; \\
&\quad smt(i) := smt(i) - 2w_{i_k}.
\end{aligned}$$

Note that the above operations are all to add a same constant to a subarray. We show in Appendix B.2 that adding a same constant to a subarray of size $O(n)$ can be done in complexity $O(\log n)$ using the data structure for the array. Hence the complexity of update_arrays is $O(\log n)$, with a pseudo-code provided in Appendix B.2.

At line 5 we check whether i_k is an s-node in G_{k-1}, that is, whether $i_k \in S_{k-1}$, and if so, there is a potential change of status of nodes. If i_k is an s-node, we proceed to the **if** statement to identify the node status change interval $[i^*_{k1}, i^*_{k2}]$. We first find the s-interval $[i_{k\ell}, i_{kr}]$ w.r.t. node i_k in G_{k-1}. This is implemented in subroutine $[i_{k\ell}, i_{kr}] :=$ get_s_interval(i_k) at line 6. The data structure for the (disjoint) s-intervals maintains them sorted in increasing order of their left endpoints. This allows to identify $[i_{k\ell}, i_{kr}]$ with binary search in $O(\log n)$ time. The pseudo-code of get_s_interval is given in Appendix A.2. With the values of i_k and $[i_{k\ell}, i_{kr}]$, the algorithm proceeds to subroutine $[i^*_{k1}, i^*_{k2}] :=$ find_status_change_interval$(i_{k\ell}, i_k, i_{kr})$ at line 7 to identify $[i^*_{k1}, i^*_{k2}]$ by solving the optimization problem (5.11) according to the procedure shown in Section §5.6. Appendix B shows that using the data structure, it takes $O(\log n)$ time to identify the minimum value of any subarray of an array and $O(\log n)$ time to query a specific element of an array. Hence the complexity of find_status_change_interval is $O(\log n)$. The pseudo-code for find_status_change_interval is in Appendix B.3.

If $[i^*_{k1}, i^*_{k2}]$ is nonempty, checked at line 8, we proceed to line 9 to record the optimal values of all x_i for all node i in the node status change interval $[i^*_{k1}, i^*_{k2}]$, as $x^*_i = a_{i_k}$, and update the status of node i from s in G_{k-1} to t in G_k. The s-interval $[i_{k\ell}, i_{kr}]$ in G_{k-1} is then decomposed into at most two new nonempty s-intervals in G_k, $[i_{k\ell}, i^*_{k1} - 1]$ (if $i^*_{k1} > i_{k\ell}$) and $[i^*_{k2} + 1, i_{kr}]$ (if $i^*_{k2} < i_{kr}$), while all the other s-intervals in G_{k-1} remain unchanged in G_k. This is achieved by subroutine update_s_interval$(i_{k\ell}, i^*_{k1}, i^*_{k2}, i_{kr})$ at line 10 by removing the s-interval $[i_{k\ell}, i_{kr}]$ from the data structure and inserting the decomposed nonempty s-intervals $[i_{k\ell}, i^*_{k1} - 1]$ and $[i^*_{k2} + 1, i_{kr}]$ into the data structure. Appendix A.3 shows that the data structure can complete the above operations, while keeping the s-intervals sorted, in $O(\log n)$ time. Hence the complexity of update_s_interval is $O(\log n)$. The pseudo-code of update_s_interval is in Appendix A.3. The optimal solution is returned at line 14.

It takes $O(n \log n)$ time to sort the breakpoints a_is. In each iteration of the **for** loop starting at line 3, each of the four subroutines called takes $O(\log n)$ time. For each node $i \in V = [1, n]$, its corresponding decision variable gets optimal value assigned exactly once and it changes status from s to t exactly once over the n iterations. Thus the amortized complexity of line 9 is $O(1)$ in each iteration. Therefore each iteration takes amortized time $O(\log n)$. This completes the proof of Theorem 7.

In addition, reading the input data and output the optimal solution take $O(n)$ time in total. Thus the total complexity of ℓ_1-GIMR-Algorithm is $O(n \log n)$. We therefore conclude that,

Theorem 14. *ℓ_1-GIMR-Algorithm solves problem ℓ_1-GIMR (5.10) in $O(n \log n)$ time.*

5.7 Extending ℓ_1-GIMR-Algorithm to GIMR-Algorithm

The key ideas used in ℓ_1-GIMR-algorithm are extended here for GIMR (5.1) of general convex piecewise linear deviation functions. The adjustments required are described below.

First we note that without loss of generality, any convex piecewise linear deviation function $f_i^{pl}(x_i)$ with box constraints for the variable $\ell_i \leq x_i \leq u_i$, is equivalent to a convex piecewise linear function without the box constraints:

$$\tilde{f}_i^{pl}(x_i) = \begin{cases} f_i^{pl}(\ell_i) - M(x_i - \ell_i) & \text{for } x_i < \ell_i, \\ f_i^{pl}(x_i) & \text{for } \ell_i \leq x_i \leq u_i, \\ f_i^{pl}(u_i) + M(x_i - u_i) & \text{for } x_i > u_i, \end{cases}$$

for M sufficiently large. Therefore GIMR is unconstrained, without loss of generality, with the first piece of each convex piecewise linear function having negative (non-positive) slope ($w_{i,0} = -M$) and the last piece of each convex piecewise linear function having positive (non-negative) slope ($w_{i,q_i} = M$).

The running time of GIMR-algorithm is proved in Theorem 15:

Theorem 15. *GIMR (5.1) is solved in $O(q \log n)$ time, where q is the total number of breakpoints of the n arbitrary convex piecewise linear deviation functions.*

Proof. For GIMR (5.1), the structure of G_0 remains as in Figure 5.1 as for ℓ_1-GIMR (5.10), with $c_{s,i} = -w_{i,0} > 0$ and $c_{i,t} = 0$ for all $i = 1, \ldots, n$. Thus the minimum cut in G_0 is $(\{s\} \cup V, \{t\})$. Hence subroutine initialization() is still valid for GIMR (5.1) in the same complexity.

For GIMR (5.1), as for ℓ_1-GIMR (5.10), all arc capacities other than c_{s,i_k} and $c_{i_k,t}$ are the same for both G_{k-1} and G_k. But the construction of G_k from G_{k-1} is more complicated than that in ℓ_1-GIMR. Recall that from G_{k-1} to G_k, the right sub-gradient of $f_{i_k}^{pl}$ changes from w_{i_k,j_k-1} to w_{i_k,j_k}. Thus depending on the signs of w_{i_k,j_k-1} and w_{i_k,j_k}, we have the following three possible cases:

Case 1. $\underline{w_{i_k,j_k-1} \leq 0,\ w_{i_k,j_k} \leq 0}$: c_{s,i_k} is changed from $-w_{i_k,j_k-1}$ to $-w_{i_k,j_k}$.

Case 2. $\underline{w_{i_k,j_k-1} \leq 0,\ w_{i_k,j_k} \geq 0}$: c_{s,i_k} is changed from $-w_{i_k,j_k-1}$ to 0 and $c_{i_k,t}$ is changed from 0 to w_{i_k,j_k}.

Case 3. $\underline{w_{i_k,j_k-1} \geq 0,\ w_{i_k,j_k} \geq 0}$: $c_{i_k,t}$ is changed from w_{i_k,j_k-1} to w_{i_k,j_k}.

The capacities of other arcs do not change.

Accordingly, the four arrays are updated for each one of these three cases as follows:

Case 1. $\underline{w_{i_k,j_k-1} \leq 0,\, w_{i_k,j_k} \leq 0}$:

$$\begin{aligned}
&\forall i \in [i_k, n]:\\
&\quad sa(i) := sa(i) - (w_{i_k,j_k} - w_{i_k,j_k-1});\\
&\quad tms(i) := tms(i) + (w_{i_k,j_k} - w_{i_k,j_k-1});\\
&\quad smt(i) := smt(i) - (w_{i_k,j_k} - w_{i_k,j_k-1}).
\end{aligned}$$

Case 2. $\underline{w_{i_k,j_k-1} \leq 0,\, w_{i_k,j_k} \geq 0}$:

$$\begin{aligned}
&\forall i \in [i_k, n]:\\
&\quad sa(i) := sa(i) + w_{i_k,j_k-1};\\
&\quad ta(i) := ta(i) + w_{i_k,j_k};\\
&\quad tms(i) := tms(i) + (w_{i_k,j_k} - w_{i_k,j_k-1});\\
&\quad smt(i) := smt(i) - (w_{i_k,j_k} - w_{i_k,j_k-1}).
\end{aligned}$$

Case 3. $\underline{w_{i_k,j_k-1} \geq 0,\, w_{i_k,j_k} \geq 0}$:

$$\begin{aligned}
&\forall i \in [i_k, n]:\\
&\quad ta(i) := ta(i) + (w_{i_k,j_k} - w_{i_k,j_k-1});\\
&\quad tms(i) := tms(i) + (w_{i_k,j_k} - w_{i_k,j_k-1});\\
&\quad smt(i) := smt(i) - (w_{i_k,j_k} - w_{i_k,j_k-1}).
\end{aligned}$$

Although seemingly more complicated, all the above operations amount to adding a constant to a subarray, which can be done efficiently using the data structure for the four arrays. The above update is done by calling the subroutine update_arrays($i_k, w_{i_k,j_k-1}, w_{i_k,j_k}$) in complexity $O(\log n)$ (see Appendix B.2 for pseudo-code). Note that the ℓ_1 deviation function in ℓ_1-GIMR (5.10) is a special case of Case 2 above where $w_{i_k,j_k-1} < 0$, $w_{i_k,j_k} > 0$ and $-w_{i_k,j_k-1} = w_{i_k,j_k}$.

On the other hand, since all arc capacities other than c_{s,i_k} and $c_{i_k,t}$ are the same for both G_{k-1} and G_k, Lemma 8, Lemma 9, Lemma 10 and Corollary 11 for ℓ_1-GIMR (5.10) hold true for GIMR (5.1). As a result, the procedure to identify the node status change interval $[i^*_{k1}, i^*_{k2}]$ in graph G_k for ℓ_1-GIMR, shown in Section §5.6 and §5.6, also applies to GIMR. This implies that subroutines get_s_interval, find_status_change_interval and update_s_interval are still valid for GIMR in the same complexity respectively as for ℓ_1-GIMR. Thus Theorem 7 holds for GIMR.

The complete GIMR-algorithm is:

GIMR-Algorithm
input: $\{\{a_{i,1}, \ldots, a_{i,q_i}\}, \{w_{i,0}, \ldots, w_{i,q_i}\}\}_{i=1,\ldots,n}$ and $\{d_{i,i+1}, d_{i+1,i}\}_{i=1,\ldots,n-1}$ in GIMR (5.1).
output: An optimal solution $\{x^*_i\}_{i=1,\ldots,n}$.
begin

```
    Sort the breakpoints of all the $n$ convex piecewise linear functions as $a_{i_1,j_1} < a_{i_2,j_2} < \ldots < a_{i_q,j_q}$;

    initialization();

    for $k := 1, \ldots, q$:

        $\{$Update graph$\}$update_arrays($i_k$, $w_{i_k,j_k-1}$, $w_{i_k,j_k}$);
        if $status(i_k) = s$ then
            $[i_{k\ell}, i_{kr}]$ := get_s_interval($i_k$);
            $[i^*_{k1}, i^*_{k2}]$ := find_status_change_interval($i_{k\ell}, i_k, i_{kr}$);
            if $[i^*_{k1}, i^*_{k2}] \neq \emptyset$ then
                for $i \in [i^*_{k1}, i^*_{k2}]$: $x^*_i := a_{i_k,j_k}$, $status(i) := t$;
                update_s_interval($i_{k\ell}, i^*_{k1}, i^*_{k2}, i_{kr}$);
            end if
        end if

    end for

    return $\{x^*_i\}_{i=1,\ldots,n}$;

end
```

Reading the input data takes $O(q)$ time. Sorting the q breakpoints from n ordered lists takes $O(q \log n)$ time [27]. The amortized complexity of each iteration in the **for** loop remains $O(\log n)$, thus for the q iterations the total complexity is $O(q \log n)$. Finally it takes $O(n)$ time to output the optimal solution. Therefore the total complexity of GIMR-Algorithm is $O(q \log n)$. □

Remark 16. *The discussion above and algorithms' presentation assume that all breakpoints are distinct. However, when this is not the case and a breakpoint is shared by more than one function, the algorithms still work in the same way, by breaking ties arbitrarily: A breakpoint is associated with a function or a variable, or with multiple functions and variables. The ordering of the variables that correspond to the same breakpoint can be selected arbitrary. To see that, consider a "perturbed" problem, in which small perturbations are applied to the original shared breakpoints so as to break the ties. The values that the optimal variables would then assume are either values of the breakpoints or the perturbed breakpoints. Since the perturbations can be made arbitrarily small, it follows that the optimal variables values will be among the "un-perturbed" breakpoints.*

Remark 17. *GIMR-Algorithm also works for GIMR (5.1) on integer valued variables. In [44] it is proved in the Threshold Theorem (Theorem 3.1) that the integer optimal solution can only change when at least one integer sub-gradient, $f'(x) = f(x+1) - f(x)$, changes.*

This implies that, for the piecewise linear functions studied here, the only possible optimal values of the variables are the breakpoints rounded up or down. In contrast to Lemma 4 for the continuous case that states that the optimal values reside at the breakpoints, here the optimal values can only be the breakpoints rounded up or down (follows from [44]). Therefore the running time of GIMR-Algorithm for the integer version of GIMR is the same as for the continuous version, $O(q \log n)$.

5.8 Experimental Study

We implement GIMR-Algorithm in C++ in Microsoft Visual Studio 2015. We use the "set" data structure object in C++ standard template library (STL) to maintain the set of s-intervals in the algorithm. The dynamic path data structure has been implemented according to [97]. In order to assess the performance of GIMR-Algorithm in practice, we compare our software implementation with Gurobi (version 6.5.2), a commercial linear programming solver, on 30 simulated data sets of various sizes. Both algorithms are run on the same laptop with Intel(R) Core i7-6820HQ CPU at 2.70GHz, 32GB RAM, and 64-bit Windows 10 operating system.

The GIMR problem can be formulated as a linear programming problem: Let $b_{i,j} = f_i^{pl}(a_{i,j})$ for $i = 1, \ldots, n; j = 1, \ldots, q_i$. It is easy to see that the following linear programming problem has the same optimal solution as GIMR (5.1), where u_i is the upper envelope of the q_i lines, $\{w_{i,0}(x_i - a_{i,1}) + b_{i,1}, \{w_{i,j}(x_i - a_{i,j}) + b_{i,j}\}_{j=1,\ldots,q_i}\}$, which correspond to the q_i linear pieces of function $f_i^{pl}(x_i)$:

$$\begin{aligned}
\min_{\substack{\{u_i, x_i\}_{i=1,\ldots,n} \\ \{z_{i,i+1}, z_{i+1,i}\}_{i=1,\ldots,n-1}}} \quad & \sum_{i=1}^{n} u_i + \sum_{i=1}^{n-1} d_{i,i+1} z_{i,i+1} + \sum_{i=1}^{n-1} d_{i+1,i} z_{i+1,i} \\
s.t. \quad & u_i \geq w_{i,0}(x_i - a_{i,1}) + b_{i,1}, \ i = 1, \ldots, n \\
& u_i \geq w_{i,j}(x_i - a_{i,j}) + b_{i,j}, \ i = 1, \ldots, n; j = 1, \ldots, q_i \\
& x_i - x_{i+1} \leq z_{i,i+1}, \ i = 1, \ldots, n-1 \\
& x_{i+1} - x_i \leq z_{i+1,i}, \ i = 1, \ldots, n-1 \\
& \ell_i \leq x_i \leq u_i, \ i = 1, \ldots, n \\
& z_{i,i+1}, z_{i+1,i} \geq 0, \ i = 1, \ldots, n-1.
\end{aligned}$$

The simulated data sets. In the generated data sets there are no box constraints. For each convex piecewise linear deviation function, we let the slope of the first linear piece be negative and the slope of the last linear piece be positive. That guarantees that the problem has an optimal solution in a bounded interval. In the separation terms, we set $d_{i,i+1} = d_{i+1,i} = d_i$, thus $d_{i,i+1}(x_i - x_{i+1})_+ + d_{i+1,i}(x_{i+1} - x_i)_+ = d_i|x_i - x_{i+1}|$. We set the number of breakpoints of each piecewise linear function $f_i^{pl}(x_i)$, q_i, to be all equal to a common value $\bar{q}$. Thus the total number of breakpoints of the n convex piecewise linear functions is $q = n\bar{q}$. For each pair of $(n, \bar{q})$, we generate 5 random problem instances, by randomly generating 5 groups of $\bar{q}+1$ slope values, $\bar{q}$ breakpoints (for each convex piecewise linear deviation function) and d_i coefficients for the separation terms. The slopes of each $f_i^{pl}(x_i)$ are randomly generated in increasing order as follows: We first sample the value of $w_{i,0}$ from a uniform distribution on $(-\bar{q}, 0)$. Each subsequent breakpoint $w_{i,j}$ $(j = 1, \ldots, q_i)$ is generated by adding a uniformly sampled random real value from $(0, 100)$ to $w_{i,j-1}$. The breakpoints of $f_i^{pl}(x_i)$ are generated in increasing order as follows: A first value, denoted as $a_{i,0}$, is sampled with uniform distribution from $(-\bar{q}, 0)$. This value is not a breakpoint. Each subsequent breakpoint $a_{i,j}$ $(j = 1, \ldots, q_i)$ is generated by adding a uniformly sampled real value from $(0, 100)$ to $a_{i,j-1}$. This guarantees that the generated slopes and breakpoints are strictly increasing in each convex piecewise linear function. Each d_i value is sampled uniformly from the interval $(0, \bar{q})$.

We compare the average running times of GIMR-Algorithm and Gurobi for the 5 random instances of GIMR for each pair of $(n, \bar{q})$. We use default parameters in calling the Gurobi linear programming solver. We report the average running times(standard deviations) for all six families of problem instances in Table 5.2.

Table 5.2: Running time (in seconds) comparison between GIMR-Algorithm and Gurobi for solving GIMR (5.1). The numbers reported are the average running times(standard deviations).

	Time (in seconds)	
$(n, \bar{q})$	GIMR-Algorithm	Gurobi
$(100, 100)$	0.17(0.013)	1.64(0.034)
$(100, 1000)$	1.47(0.079)	15.59(0.29)
$(1000, 100)$	1.97(0.035)	16.37(0.50)
$(1000, 1000)$	16.33(0.12)	148.46(0.40)
$(1000, 10000)$	174.06(9.86)	1608.36(61.66)
$(10000, 1000)$	190.53(3.62)	1559.07(63.41)

From Table 5.2 one can see that GIMR-Algorithm is approximately 10 times faster than Gurobi for each problem size with a smaller standard deviation.

5.9 Concluding Remarks

We describe here an efficient algorithm that solves GIMR (5.1), generalizing isotonic median regression and a class of fused lasso problems with wide applications in signal processing, bioinformatics and statistical learning. The algorithm proposed here is the first known unified, and most efficient in terms of complexity to date, for isotonic median regression (IMR), simple isotonic median regression (SIMR), and fused lasso problems with convex piecewise linear deviation functions. The latter includes the quantile fused lasso (Q-FL) and the quantile weighted fused lasso (Q-wFL) problems. For all these problems our algorithm improves or matches on previous complexities of a collection of specialized algorithms and offers a unified framework for all these problems. The unified framework here is also amenable to extensions to other generalized versions of GIMR, that include generalized isotonic median regression on simple structure graphs, such as directed trees or cycles. The algorithm devised here is also shown to work well in practice, as demonstrated in an empirical study.

Chapter 6

A Lot-sizing-linked Algorithm for MRF on Path

Recall that the formulation of MRF-BL on a path graph of general convex deviation functions is:

$$\begin{aligned}\text{(MRF-BL-PATH)} \quad \min_{x_1,\ldots,x_n} \ & \sum_{i=1}^{n} f_i(x_i) + \sum_{i=1}^{n-1} d_{i,i+1}(x_i - x_{i+1})_+ + \sum_{i=1}^{n-1} d_{i+1,i}(x_{i+1} - x_i)_+ \\ & s.t.\ \ell_i \leq x_i \leq u_i,\ i = 1, \ldots, n.\end{aligned} \tag{6.1}$$

In this chapter, we devise a fast polynomial time algorithm to solve MRF-BL-PATH of convex L-Lipschitz deviation functions $f_i(x_i)$. Recall that a function $f(x)$ $(x \in \mathbb{R})$ is L-Lipschitz continous if there exists an $L > 0$ such that for any $x, x' \in \mathbb{R}$, $|f(x) - f(x')| \leq L|x - x'|$.

6.1 Comparison with Existing Best Algorithms

We give an algorithm to solve MRF-BL-PATH (6.1), of convex L-Lipschitz deviation functions and arbitrary coefficients $d_{i,i+1}, d_{i+1,i} \geq 0$, in time complexity of $O(n(\log n + \log \frac{U}{\epsilon}) \log(\frac{nDU}{\epsilon}))$, where $D = \max\{L, \max_i\{d_{i,i+1}, d_{i+1,i}\}\}$.

Prior to this work, the fastest algorithm for MRF-BL-PATH of convex L-Lipschitz deviation functions is by direct application of H01-algorithm in [44] for MRF-BL from arbitrary graphs to path graphs. Recall that H01-algorithm has run time $O(nm \log \frac{n^2}{m} + n \log \frac{U}{\epsilon})$. When the graph is a path graph, $m = O(n)$, hence the complexity of H01-algorithm is $O(n^2 \log n + n \log \frac{U}{\epsilon})$. The algorithm presented here improves over this prior best complexity in terms of parameter n by a factor of $O(n)$.

There exist faster specialized algorithms for MRF-BL-PATH of specific forms of deviation and separation functions. For convex quadratic deviation functions and absolute value (ℓ_1-

norm) separation functions of same coefficient $\lambda > 0$:

$$\min_{x_1,\ldots,x_n} \frac{1}{2}\sum_{i=1}^{n}(x_i - a_i)^2 + \lambda \sum_{i=1}^{n-1} |x_i - x_{i+1}|, \tag{6.2}$$

Johnson in [59] gave an efficient $O(n)$ algorithm by dynamic programming. Hoefling in [55] gave an efficient algorithm that finds the solution for all values of $\lambda \geq 0$ of problem (6.2) in a total time complexity of $O(n \log n)$. Tibshirani et al. in [102] studied a "nearly-isotonic regression" model which is a special case of MRF-BL-PATH with convex quadratic deviation functions and separation functions of "one-sided" penalty:

$$\text{(Nearly-isotonic regression)} \quad \min_{x_1,\ldots,x_n} \frac{1}{2}\sum_{i=1}^{n}(x_i - a_i)^2 + \lambda \sum_{i=1}^{n-1} (x_i - x_{i+1})_+. \tag{6.3}$$

They gave an efficient algorithm that solves the solution for all values of $\lambda \geq 0$ of nearly-isotonic regression (6.3) in time $O(n \log n)$. Comparing with the above faster specialized algorithms for the case of convex quadratic deviation functions, the algorithm presented here is one unified algorithm applicable to a broader class of arbitrary convex deviation functions at an additional cost to the complexity of only some logarithmic factors.

6.2 A Lot-sizing-linked Algorithm

Our algorithm works on the following equivalent formulation of MRF-BL-PATH:

$$\begin{aligned}
\text{(MRF-BL-PATH)} \quad \min_{\substack{\{x_i\}_{i=1,\ldots,n} \\ \{z_{i,i+1}, z_{i+1,i}\}_{i=1,\ldots,n-1}}} \quad & \sum_{i=1}^{n} f_i(x_i) + \sum_{i=1}^{n-1} d_{i,i+1} z_{i,i+1} + \sum_{i=1}^{n-1} d_{i+1,i} z_{i+1,i} \\
s.t.\ & x_i - x_{i+1} \leq z_{i,i+1},\ i = 1,\ldots,n-1 \\
& x_{i+1} - x_i \leq z_{i+1,i},\ i = 1,\ldots,n-1 \\
& \ell_i \leq x_i \leq u_i,\ i = 1,\ldots,n \\
& z_{i,i+1}, z_{i+1,i} \geq 0,\ i = 1,\ldots,n-1.
\end{aligned} \tag{6.4}$$

Our algorithm leverages two existing results in [2] and [1]. We consider the dual problem of MRF-BL-PATH (6.4), which was discussed in [2] for the more general MRF problem (1.1) on arbitrary directed graphs. Following the primal-dual transformation in [2], we show that the dual problem is a convex minimum cost network flow problem defined on a simple structure graph (almost like a path graph). It was shown in [2] that the optimal objective values of the primal MRF-BL-PATH problem and the dual convex minimum cost network flow problem are equal. Ahuja and Hochbaum in [1] studied this specific convex minimum cost network flow problem as a lot-sizing problem in production planning and gave an efficient algorithm to solve this problem. Finally, we can compute an optimal solution to the primal MRF-BL-PATH problem from the optimal solution to the dual convex minimum cost network flow problem by a shortest path computation on the almost-path graph in the dual network flow problem, which was discussed in [2]. The shortest path computation can be done efficiently since the underlying graph has simple structure.

As in most places we directly adapt the results in [2] and [1] to our specific problem, we only highlight the key steps in the adaptation. Note that our notation is significantly different from that in [2] and [1].

We first remove the box constraints $\ell_i \leq x_i \leq u_i$, without loss of generality, by modifying each deviation function $f_i(x_i)$ as in [2]:

$$\tilde{f}_i(x_i) = \begin{cases} f_i(\ell_i) - M(x_i - \ell_i), & \text{for } x_i < \ell_i, \\ f_i(x_i), & \text{for } \ell_i \leq x_i \leq u_i, \\ f_i(u_i) + M(x_i - u_i), & \text{for } x_i > u_i, \end{cases}$$

for a sufficiently large M. As is shown in [2], a sufficient value of M is equal to an upper bound to the objective function value of MRF-BL-PATH (6.4) minus a lower bound to the objective function value, both ignoring the constraints $x_i - x_{i+1} \leq z_{i,i+1}$ and $x_{i+1} - x_i \leq z_{i+1,i}$. As the deviation functions are L-Lipschitz continuous and the separation functions are linear, the difference between an upper bound and a lower bound of the objective function value is bounded by $O(nDU)$, where $D = \max\{L, \max_i\{d_{i,i+1}, d_{i+1,i}\}\}$. As we solve MRF-BL-PATH (6.4) on an ϵ-grid, a minimum violation to the box constrain $\ell_i \leq x_i \leq u_i$ is ϵ, thus we scale the bound by $\frac{1}{\epsilon}$. Hence $M = O(\frac{nDU}{\epsilon})$.

After removing the box constraints, we construct the dual problem of MRF-BL-PATH (6.4). We introduce a dual variable $\mu_{i,i+1}$ for each constraint $x_i - x_{i+1} \leq z_{i,i+1}$ and a dual variable $\mu_{i+1,i}$ for each constraint $x_{i+1} - x_i \leq z_{i+1,i}$. In addition, we introduce a dual variable $\mu_{0,i}$ (unrestricted in sign) for each deviation function $f_i(x_i)$. For details of constructing the dual problem, we refer readers to [2]. The dual problem is formulated as follows:

$$\begin{aligned} \min_{\substack{\{\mu_{0,i}\}_{i=1,\ldots,n} \\ \{\mu_{i,i+1},\mu_{i+1,i}\}_{i=1,\ldots,n-1}}} \quad & \sum_{i=1}^{n} C_{0,i}(\mu_{0,i}) \\ s.t.\ & \mu_{1,2} - (\mu_{2,1} + \mu_{0,1}) = 0, \\ & \mu_{i,i+1} + \mu_{i,i-1} - (\mu_{i+1,i} + \mu_{i-1,i} + \mu_{0,i}) = 0,\ i = 2, \ldots, n-1 \\ & \mu_{n,n-1} - (\mu_{n-1,n} + \mu_{0,n}) = 0, \\ & -M \leq \mu_{0,i} \leq M,\ i = 1, \ldots, n \\ & 0 \leq \mu_{i,i+1} \leq d_{i,i+1},\ i = 1, \ldots, n-1 \\ & 0 \leq \mu_{i+1,i} \leq d_{i+1,i},\ i = 1, \ldots, n-1. \end{aligned} \tag{6.5}$$

Each function $C_{0,i}(\mu_{0,i})$ is defined as $C_{0,i}(\mu_{0,i}) = -\min_{x_i}\{f_i(x_i) - \mu_{0,i}x_i : \ell_i \leq x_i \leq u_i\}$ [2]. It was shown in [2] that $C_{0,i}(\mu_{0,i})$ is a convex function of $\mu_{0,i}$. Furthermore, as x_i takes integer multiple values of ϵ, it was shown in [2] that $C_{0,i}(\mu_{0,i})$ is a piecewise linear function whose slopes of the linear pieces are the integer multiple values of ϵ. Hence the number of linear pieces in $C_{0,i}(\mu_{0,i})$ is $O(\frac{u_i-\ell_i}{\epsilon})$.

In dual problem formulation (6.5), we re-define $\mu_{0,i}$ as $\mu_{0,i}+M$. Then the equivalent formulation (6.5) becomes:

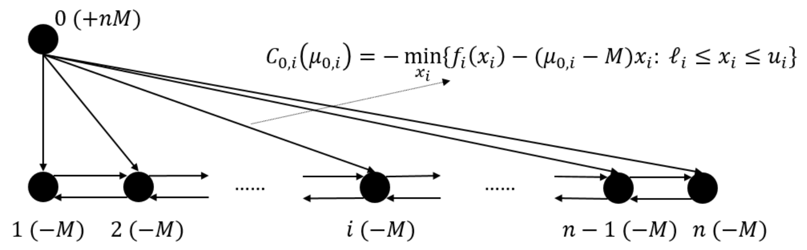
$$
\begin{aligned}
\min_{\substack{\{\mu_{0,i}\}_{i=1,\dots,n}\\ \{\mu_{i,i+1},\mu_{i+1,i}\}_{i=1,\dots,n-1}}} \quad & \sum_{i=1}^{n} C_{0,i}(\mu_{0,i}) \\
s.t.\ & \mu_{1,2}-(\mu_{2,1}+\mu_{0,1})=-M, \\
& \mu_{i,i+1}+\mu_{i,i-1}-(\mu_{i+1,i}+\mu_{i-1,i}+\mu_{0,i})=-M,\ i=2,\dots,n-1 \\
& \mu_{n,n-1}-(\mu_{n-1,n}+\mu_{0,n})=-M, \\
& 0\le\mu_{0,i}\le 2M,\ i=1,\dots,n \\
& 0\le\mu_{i,i+1}\le d_{i,i+1},\ i=1,\dots,n-1 \\
& 0\le\mu_{i+1,i}\le d_{i+1,i},\ i=1,\dots,n-1.
\end{aligned}
\tag{6.6}
$$

And function $C_{0,i}(\mu_{0,i})$ is updated to $C_{0,i}(\mu_{0,i})=-\min_{x_i}\{f_i(x_i)-(\mu_{0,i}-M)x_i:\ell_i\le x_i\le u_i\}$. This shift of variable $\mu_{0,i}$ does not affect the properties of function $C_{0,i}(\mu_{0,i})$ stated above.

The dual problem (6.6) is a convex minimum cost network flow problem. To see this, consider the graph $G=(V,A)$, where $V=\{0,1,2,\dots,n\}$, and $A=\{\{(0,i)\}_{i=1,\dots,n},\{(i,i+1),(i+1,i)\}_{i=1,\dots,n-1}\}$. The flow value on an arc $(0,i)$ is $\mu_{0,i}$, the flow value on an arc $(i,i+1)$ is $\mu_{i,i+1}$, and the flow value on an arc $(i+1,i)$ is $\mu_{i+1,i}$. The supply of node 0 is $+nM$, and the demand of node i $(i=1,\dots,n)$ is $-M$. The cost of flow $\mu_{0,i}$ on arc $(0,i)$ is $C_{0,i}(\mu_{0,i})$ $(i=1,\dots,n)$. The costs of flows $\mu_{i,i+1}$ and $\mu_{i+1,i}$ on the respective arc $(i,i+1)$ and $(i+1,i)$ are all 0. The capacity of arc $(0,i)$ is $2M$, the capacity of arc $(i,i+1)$ is $d_{i,i+1}$, and the capacity of arc $(i+1,i)$ is $d_{i+1,i}$. The convex minimum cost network flow problem is a capacitated dynamic lot-sizing problem with back orders, where node 0 is a production node and the other nodes 1 to n are demand nodes for period 1 to n. The flows on the arcs are flows of items through production, inventory forwarding, or back ordering [1]. The convex minimum cost network flow problem is illustrated in Figure 6.1.

The convex minimum cost network flow problem (6.6) (shown in Figure 6.1) is defined on a simple structure graph (a path plus an additional node 0), and it has convex costs only on arcs $(0,i)$, while the costs on arcs $(i,i+1)$ and $(i+1,i)$ are all linear (0 costs in our case). Ahuja and Hochbaum in [1] studied this capacitated dynamic lot-sizing problem in production planning and gave an efficient algorithm to solve this problem. Their algorithm solves our problem (6.6) in time $O\big(n(\log n+\log\frac{U}{\epsilon})\log M\big)=O\big(n(\log n+\log\frac{U}{\epsilon})\log(\frac{nDU}{\epsilon})\big)$ [1] (see Remark 19).

Solving the convex minimum cost network flow problem gives an optimal flow solution $\{\{\mu^*_{0,i}\}_{i=1,\dots,n},\ \{\mu^*_{i,i+1},\mu^*_{i+1,i}\}_{i=1,\dots,n-1}\}$. An optimal solution to the primal MRF-BL-PATH problem (6.4) can be obtained by solving a shortest path problem from node 0 to all other nodes on an associated graph defined by the original graph $G=(V,A)$ (defining the network flow problem) and the optimal flow values [2]. This "shortest path" graph is very similar to



0 (+nM)
$C_{0,i}(\mu_{0,i}) = -\min_{x_i}\{f_i(x_i) - (\mu_{0,i} - M)x_i: \ell_i \le x_i \le u_i\}$
1 (−M)
2 (−M)
i (−M)
n − 1 (−M)
n (−M)

Figure 6.1: Illustration of the lot-sizing problem, which is a convex minimum cost network flow problem (6.6). The numbers in parentheses are the supply/demands of respective nodes, where $M = O(\frac{nDU}{\epsilon})$, $D = \max\{L, \max_i\{d_{i,i+1}, d_{i+1,i}\}\}$. Flow variable along each arc (i, j) is $\mu_{i,j}$. The cost on an arc $(0, i)$ is a convex cost $C_{0,i}(\mu_{0,i})$ (shown in the figure). The costs on arcs $(i, i+1)$ and $(i+1, i)$ are all 0. The capacity of arc $(0, i)$ is $2M$, the capacity of arc $(i, i+1)$ is $d_{i,i+1}$, and the capacity of arc $(i+1, i)$ is $d_{i+1,i}$. The capacities of the arcs are not shown in the figure.

the original graph G in that there can only be arcs, in either direction, between node 0 and node i ($i = 1, \dots, n$), and between node i and node $i + 1$ ($i = 1, \dots, n - 1$). In the "shortest path" graph, the cost of an arc between node 0 and node i is the slope, or its negative, of the linear piece covering $\mu^*_{0,i}$ in function $C_{0,i}(\mu_{0,i})$; the cost of an arc between node i and node $i + 1$ is 0. Let $d(i)$ be the shortest path distance from node 0 to node i in the "shortest path" graph. Since all costs in the "shortest path" graph are integer multiples of ϵ, so do all the $d(i)$ distances. It was shown in [2] that $x^*_i = d_i$ ($i = 1, \dots, n$) is an optimal solution to the primal MRF-BL-PATH problem (6.4). Constructing the "shortest path" graph takes $O(n \log \frac{U}{\epsilon})$, where each $O(\log \frac{U}{\epsilon})$ term corresponds to finding the linear piece of $C_{0,i}(\mu_{0,i})$ containing $\mu^*_{0,i}$. Since the "shortest path" graph has a simple structure, we can solve all the shortest path distances $\{d(i)\}_{i=1,\dots,n}$ in $O(n \log n)$ time using the Dijkstra's algorithm and a binary heap data structure [27]. Hence it takes a total of $O\big(n(\log n + \log \frac{U}{\epsilon})\big)$ time to compute an optimal solution to MRF-BL-PATH (6.4) from an optimal flow solution to the dual problem.

Combining the complexities of the two steps, we conclude that the total complexity of the algorithm is $O\big(n(\log n + \log \frac{U}{\epsilon}) \log \frac{nDU}{\epsilon}\big)$. Thus we have:

Theorem 18. *The MRF-BL-PATH problem (6.1) of convex L-Lipschitz deviation functions is solved in time $O\big(n(\log n + \log \frac{U}{\epsilon}) \log \frac{nDU}{\epsilon}\big)$, where $D = \max\{L, \max_i\{d_{i,i+1}, d_{i+1,i}\}\}$.*

Remark 19. *In Ahuja and Hochbaum's algorithm in [1], at each iteration one needs to evaluate the convex cost $C_{0,i}(\mu_{0,i})$ on a value of argument $\mu_{0,i}$ on arc $(0, i)$. In their set-up, they assume the time complexity to evaluate the convex function $C_{0,i}(\mu_{0,i})$ for any argument is $O(1)$ time, so they state the complexity as $O\big(n(\log n + O(1)) \log d_{tot}\big) = O(n \log n \log d_{tot})$, where d_{tot} is the total demand quantity from node 1 to n, which is $O(nM) = O(n^2DU/\epsilon)$ in* *our case. In our set-up, however, we only assume that one can evaluate in* $O(1)$ *time the value of the deviation functions* $f_i(x_i)$*, not for the functions* $C_{0,i}(\mu_{0,i})$*. Hence for any value of argument* $\mu_{0,i}$*, we do a binary search over the linear pieces of* $C_{0,i}(\mu_{0,i})$ *to evaluate the function value of* $C_{0,i}(\mu_{0,i})$*, which incur an additional complexity of* $O(\log \frac{U}{\epsilon})$*.*

Chapter 7

KKT-based Algorithms for MRF on Path

The problem studied in this chapter is MRF on a path graph, which is formulated as follows:

$$\text{(MRF-PATH)} \min_{x_1,\ldots,x_n} \sum_{i=1}^{n} f_i(x_i) + \sum_{i=1}^{n-1} h_i(x_i - x_{i+1}) \tag{7.1}$$

We assume that the optimal values of the variables in MRF-PATH are all in a bounded range $[-U, U]$ for some $U < \infty$, which is implied by the objective functions.

Applications of MRF-PATH arise in time-series data analysis, in smoothing function fitting to data, and in genetic data smoothing. In many practical contexts of time-series and genetic signal processing, the signals are assumed to be smooth over time and genetic sequences (e.g. see [67] and the references therein). One problem is that the observed data do not satisfy this signal smoothness constraint. The MRF-PATH problem [91, 72, 34, 67] seeks to tackle the problem by adjusting the observed data values such that the adjusted values are smooth over the time or genetic sequences, while resemble the given observations. Suppose we observe data at n discrete positions along a sequence. We model the n positions as n nodes, $1, \ldots, n$, on a path graph where node i, except nodes 1 and n, has an edge with node $i-1$ and node i. Let $a_i \in \mathbb{R}$ be the observed value at node i, and $x_i \in \mathbb{R}$ be the adjusted value from a_i. For each node i, we introduce a convex deviation function $f_i(x_i; a_i)$ that penalizes the distance between x_i and a_i. It is a data fidelity term that fits the adjusted value to the respective observation. In addition, for each edge $[i, i+1]$, we introduce a convex separation function $h_i(x_i - x_{i+1})$ that penalizes the distance between x_i and x_{i+1}. It is a regularization term that promotes signal smoothness along the path graph. Hence we have the formulation of MRF-PATH in (7.1), for which the optimal solution is used as the adjusted values from the observed values.

In many applications, such as in [67, 110, 108], it is often the case that $f_i(x_i; a_i)$ has minimum value of 0 when $x_i = a_i$, and the function value grows with the distance between x_i and a_i enlarges, and $h_i(x_i - x_{i+1})$ has minimum value of 0 when $x_i = x_{i+1}$ and the function value grows with the distance between x_i and x_{i+1} enlarges. Yet the algorithms presented here for MRF-PATH do not require these properties to the objective functions, with only convexity assumed. Thus in the formulation (7.1) we remove the dependency on data a_i in the deviation functions.

MRF-PATH is a special case of MRF where the underlying graph is bi-path graph of arc set $A = \{(i, i+1), (i+1, i)\}_{i=1,\ldots,n-1}$. In the formulation of MRF-PATH (7.1), we combine two separation functions $h_{i,i+1}(x_i - x_{i+1})$ and $h_{i+1,i}(x_{i+1} - x_i)$ into a single separation function $h_i(x_i - x_{i+1})$.

We will discuss algorithms to solve MRF-PATH of different (restricted) sub-classes of convex deviation and separation objective functions. For each sub-class, we will denote it with a "(deviation/separation)" notation where the "deviation" placeholder specifies the additional assumptions, beyond convexity, to the deviation functions, and the "separation" placeholder specifies the additional assumptions, beyond convexity, to the separation function. We denote the broadest class of general convex deviation and separation functions as (convex/convex).

In this chapter, we present efficient algorithms to solve MRF-PATH of two different classes of deviation and separation objective functions. For the class of differentiable convex deviation functions and strictly convex separation functions (differentiable/strict), we give an algorithm that solves MRF-PATH in time $O(n \log^2 \frac{U}{\epsilon})$. In the rest of the chapter, we call this algorithm as (differentiable/strict)-algorithm. If the separation functions are convex quadratic (ℓ_2-norm), (differentiable/strict)-algorithm has complexity $O(n \log \frac{U}{\epsilon})$.

When the objective functions are from a broader class of general convex deviation and separation functions (convex/convex), we give another algorithm that solves MRF-PATH in time $O(n^2 \log^2 \frac{U}{\epsilon})$. We call this algorithm as (convex/convex)-algorithm hereafter. If the separation functions are convex piecewise linear with constant number of pieces, which include the absolute value (ℓ_1-norm) separation functions, or the separation functions are convex quadratic (ℓ_2-norm), (convex/convex)-algorithm has complexity $O(n^2 \log \frac{U}{\epsilon})$.

Both (differentiable/strict)-algorithm and (convex/convex)-algorithm directly solve the KKT optimality conditions of MRF-PATH. We believe that our KKT-based algorithms are the first to achieve fastest complexities for MRF problems using this technique.

In the following we compare our algorithms with the best algorithms to date for MRF-PATH and some of its special cases to illustrate the advantage of our algorithms.

7.1 Comparison with Existing Best Algorithms

(convex/convex) and (differentiable/strict):
The best algorithm for MRF-PATH of the two classes of objective functions, (convex/convex) and (differentiable/strict), is by direct application of the AHO-algorithm in [2] for MRF on arbitrary directed graphs.

Recall (in Chapter 1) that the AHO-algorithm finds an ϵ-accurate solution of MRF of general convex deviation and separation functions in time $O(nm \log \frac{n^2}{m} \log \frac{nU}{\epsilon})$. In MRF-

PATH, the graph is a path graph, thus $m = O(n)$, as a result a direct application of this algorithm leads to an algorithm to solve MRF-PATH of general convex deviation and separation functions in time $O(n^2 \log n \log \frac{nU}{\epsilon})$. The (convex/convex)-algorithm presented here, of complexity $O(n^2 \log^2 \frac{U}{\epsilon})$, is also applicable to general convex deviation and separation functions. The (convex/convex)-algorithm improves over this existing fastest algorithm in terms of parameter n by a factor of $O(\log n)$.

When the deviation functions are differentiable convex and the separation functions are strictly convex, the (differentiable/strict)-algorithm presented here, of complexity $O(n \log^2 \frac{U}{\epsilon})$, is applicable. In this case, the improvement of the (differentiable/strict)-algorithm over the AHO-algorithm in terms of parameter n is $O(n \log n)$, which is more significant.

(convex/"bilinear"):
When the separation functions are "bilinear", i.e. $h_i(x_i - x_{i+1}) = h_{i,i+1}(x_i - x_{i+1}) + h_{i+1,i}(x_{i+1} - x_i) = d_{i,i+1} \cdot (x_i - x_{i+1})_+ + d_{i+1,i} \cdot (x_{i+1} - x_i)_+$, the MRF-PATH problem becomes a MRF-BL-PATH problem (6.1). The existing fastest algorithm for MRF-BL-PATH of general convex deviation functions is by direct application of H01-algorithm in [44], of complexity $O(n^2 \log n + n \log \frac{U}{\epsilon})$. Since "bilinear" separation functions are not strictly convex, only the (convex/convex)-algorithm is applicable. In this case, the complexity of the (convex/convex)-algorithm, $O(n^2 \log \frac{U}{\epsilon})$, is comparable to the above complexity of $O(n^2 \log n + n \log \frac{U}{\epsilon})$.

($\boldsymbol{\ell}_p/\boldsymbol{\ell}_q$):
One special case of MRF-PATH, that was proposed in literature [91, 72, 104, 34, 55, 67] for signal smoothing, has ℓ_p-norm deviation functions and ℓ_q-norm separation functions (for fixed $p, q \geq 1$), with the separation functions multiplied by a uniform fixed parameter $\lambda > 0$:

$$\min_{x_1,\ldots,x_n} \sum_{i=1}^{n} |x_i - a_i|^p + \lambda \sum_{i=1}^{n-1} |x_i - x_{i+1}|^q \tag{7.2}$$

We use two exponents p and q to specify that deviation and separation functions can have different norms. From the objective functions, one can easily see that the optimal values of all variables are in the range of $[\min_i\{a_i\}, \max_i\{a_i\}]$. Thus here $U = O(\max_i\{a_i\} - \min_i\{a_i\})$.

Many algorithms were proposed for problem (7.2) for different values of p and q. The fastest algorithm for the case of $p = q = 1$ (ℓ_1-norm deviation and separation functions) has run time $O(n \log n)$ presented in Chapter 5. The fastest algorithm for the case of $p = 2, q = 1$ (ℓ_2-norm deviation functions and ℓ_1-norm separation functions) has run time $O(n)$ by Johnson in [59]. The fastest algorithm for the case of $p = q = 2$ (ℓ_2-norm deviation and separation functions) was given by Hohm et al. in [56] with complexity $O(n^2)$. Weinmann and Storath in [110] studied the case of $p = 2$ and $q > 2$. They used the algorithm by Chambolle and Pock in [19] which was designed to solve a family of more general convex programming problems. But they did not discuss the complexity of solving this problem (7.2) of $p = 2$ and $q > 2$ with Chambolle and Pock's algorithm. We derive this complexity as $O(n \log \frac{U}{\epsilon} \log \frac{1}{\epsilon})$. This is the fastest algorithm found in the literatures.

Weinmann et al. in [108] proposed an iterative minimization algorithm for problem (7.2) of arbitrary fixed values of $p, q \geq 1$ and $\lambda > 0$. The algorithm was based on the earlier work of the same authors in [109]. It was shown in [109] that the iterative minimization algorithm converges to an optimal solution of problem (7.2). However, they did not prove the convergence rate of the iterative algorithm.

Consider the (differentiable/strict)-algorithm and (convex/convex)-algorithm presented here for problem (7.2). When $p, q \geq 2$, the deviation functions are differentiable and the separation functions are strictly convex, hence the (differentiable/strict)-algorithm is applicable, with complexity $O(n \log^2 \frac{U}{\epsilon})$. In addition, if the separation functions are quadratic ($q = 2$), the complexity of the (differentiable/strict)-algorithm becomes $O(n \log \frac{U}{\epsilon})$. Our results are faster than the $O(n^2)$ algorithm for $p = q = 2$ by Hohm et al. in [56] and comparable to the $O(n \log \frac{U}{\epsilon} \log \frac{1}{\epsilon})$ result for $p = 2$ and $q > 2$ by Weinmann and Storath in [110].

(convex/$\boldsymbol{\ell}_q$):
In computer vision, Felzenszwalb and Zabih in [37] considered a sequential element labeling problem that is related to MRF-PATH. In the problem, a label from a discrete set is to be assigned to each element in the sequence. The problem arises, for example, in pixel labeling for a pixel along a scanline of video frames. Let the label of the ith element to be assigned as x_i. Suppose x_i takes label from set $\mathcal{L}_i$. Felzenszwalb and Zabih introduced a $D_i(x_i)$ cost for assigning a specific label x_i to the ith element of the sequence, and a $V_i(x_i, x_{i+1})$ cost for assigning specific labels $x_i \in \mathcal{L}_i$ and $x_{i+1} \in \mathcal{L}_{i+1}$ to the ith and $(i+1)$th elements respectively. The labels of the elements are attained via minimizing the sum of the above two types of costs over all elements in the sequence [37]:

$$\begin{aligned} \min_{x_1,\ldots,x_n} \ & \sum_{i=1}^{n} D_i(x_i) + \sum_{i=1}^{n-1} V_i(x_i, x_{i+1}) \\ s.t.\ & x_i \in \mathcal{L}_i,\ i = 1, \ldots, n. \end{aligned} \tag{7.3}$$

In problem (7.3), the function $V_i(x_i, x_{i+1})$ generalizes the separation function $h_i(x_i - x_{i+1})$ in that the cost is not necessarily a function of the difference between x_i and x_{i+1}, $x_i - x_{i+1}$. Felzenszwalb and Zabih in [37] used dynamic programming to solve the problem. Let $k_i = |\mathcal{L}_i|$, and $k = \max_i\{k_i\}$. They showed that the dynamic programming algorithm has run time complexity of $O(nk^2)$. In addition, they showed that, if $V_i(x_i, x_{i+1}) = |x_i - x_{i+1}|^q$ for a given value of $q \geq 1$, the dynamic programming algorithm can be sped up to $O(nk)$ complexity. In their set-up, the value of k is part of the input, so their dynamic programming algorithm has polynomial run time.

Applying the dynamic programming algorithm of Felzenszwalb and Zabih to MRF-PATH yields a pseudo-polynomial algorithm, since the parameter k is not a polynomial quantity for MRF-PATH. This is because, in order to find an ϵ-accurate solution in MRF-PATH, the set of values of each variable x_i to be considered in $[-U, U]$ are integer multiples of ϵ. The number of such values in $[-U, U]$ is $O(\frac{U}{\epsilon})$. Thus $k = \max_i\{k_i\} = O(\frac{U}{\epsilon})$. Plugging this value of k into the complexity expression of the dynamic programming algorithm, one attains an

$O(n(\frac{U}{\epsilon})^2)$ algorithm for MRF-PATH for general convex deviation and separation functions. In addition, if the separation functions are of the form $h_i(x_i - x_{i+1}) = |x_i - x_{i+1}|^q$, the speed-up version of the dynamic programming algorithm leads to an algorithm of complexity $O(n(\frac{U}{\epsilon}))$ for MRF-PATH. However, the quantity of U is exponential to the input size of MRF-PATH, as the value U is given as input with $O(\log U)$ bits.

We summarize the above comparison between our algorithms and existing algorithms in Table 7.1 for the (differentiable/strict)-algorithm and Table 7.2 for the (convex/convex)-algorithm respectively.

$f_i(x_i)$	$h_i(x_i - x_{i+1})$	Algorithm here	Algorithm(s) to-date
Convex and differentiable	Strictly Convex	$O(n \log^2 \frac{U}{\epsilon})$	$O(n^2 \log n \log(nU/\epsilon))$[2] $O(n(U/\epsilon)^2)$[37]
	$\|x_i - x_{i+1}\|^2$	$O(n \log \frac{U}{\epsilon})$	$O(n^2 \log n \log(nU/\epsilon))$[2] $O(n(U/\epsilon))$[37]
	$\|x_i - x_{i+1}\|^q$ $(q > 2)$	$O(n \log^2 \frac{U}{\epsilon})$	
$\|x_i - a_i\|^p$ $(p \geq 2)$	$\|x_i - x_{i+1}\|^2$	$O(n \log \frac{U}{\epsilon})$	$O(n^2)(p = q = 2)$[56]
	$\|x_i - x_{i+1}\|^q$ $(q > 2)$	$O(n \log^2 \frac{U}{\epsilon})$	$O(n \log(U/\epsilon) \log(1/\epsilon))(p = 2, q > 2)$[110] Iterative algorithm$(p, q \geq 2)$[108]

Table 7.1: Summary of complexity comparison between the (differentiable/strict)-algorithm and the recent/best-to-date algorithms.

$f_i(x_i)$	$h_i(x_i - x_{i+1})$	Algorithm here	Algorithm(s) to-date
Convex	Convex	$O(n^2 \log^2 \frac{U}{\epsilon})$	$O(n^2 \log n \log(nU/\epsilon))$[2] $O(n(U/\epsilon)^2)$[37]
	$\|x_i - x_{i+1}\|$	$O(n^2 \log \frac{U}{\epsilon})$	$O(n^2 \log n + n \log(U/\epsilon))$[44] $O(n(U/\epsilon))$[37]
	$\|x_i - x_{i+1}\|^2$		$O(n^2 \log n \log(nU/\epsilon))$[2] $O(n(U/\epsilon))$[37]
	$\|x_i - x_{i+1}\|^q$ $(q > 2)$	$O(n^2 \log^2 \frac{U}{\epsilon})$	

Table 7.2: Summary of complexity comparison between the (convex/convex)-algorithm and the recent/best-to-date algorithms. Since the (convex/convex)-algorithm applies to a broader class of objective functions, it can also be applied to the class of problems shown in Table 7.1.

The chapter is organized as follows. Chapter-specialized notations are introduced in Section §7.2. Then we present our algorithms for MRF-PATH in Section §7.3. We first present the (differentiable/strict)-algorithm. This algorithm conveys all key ideas that are shared by the (convex/convex)-algorithm. Then we show how to apply these ideas to derive the (convex/convex)-algorithm. Concluding remarks are discussed in Section §7.4.

7.2 Additional Notations

In this chapter, the case of convex and differentiable deviation functions includes the assumption that the gradient, or derivative, of function f is available via an oracle such that, for any ϵ-accurate argument x, it returns the derivative value $f'(x)$ in $O(1)$ time.

When a convex function f is not differentiable, $f'(x)$ denotes a sub-gradient of function f at argument x. The interval of sub-gradients of a convex function f at input x is denoted as $\partial f(x) = [f'_L(x), f'_R(x)]$, where $f'_L(x)$ is the left sub-gradient of f, and $f'_R(x)$ is the right sub-gradient of f. On the ϵ-grid, the left and right sub-gradient values are:

$$\begin{aligned} f'_L(x) &= (f(x) - f(x-\epsilon))/\epsilon, \\ f'_R(x) &= (f(x+\epsilon) - f(x))/\epsilon. \end{aligned} \tag{7.4}$$

By equations (7.4), the intervals of sub-gradients of function f over input on the ϵ-grid are continuous in that $f'_R(x) = f'_L(x+\epsilon)$. For simplicity, we let $f'(-\infty) = f'_L(-\infty) = f'_R(-\infty) = -\infty$ and $f'(+\infty) = f'_L(+\infty) = f'_R(+\infty) = +\infty$.

By the convexity of function f, for any $x_1 < x_2$, we have

$$f'_L(x_1) \le f'_R(x_1) \le f'_L(x_2) \le f'_R(x_2). \tag{7.5}$$

A function $f(x)$ is *strictly convex* if for any $x_1 < x_2$ and $\lambda \in (0, 1)$, we have the following strict inequality holds:

$$f(\lambda x_1 + (1-\lambda)x_2) < \lambda f(x_1) + (1-\lambda) f(x_2).$$

Given a strictly convex function $f(x)$, for any $x_1 < x_2$, the first and third inequalities of (7.5) hold strictly:

$$f'_L(x_1) < f'_R(x_1) \le f'_L(x_2) < f'_R(x_2).$$

We introduce an inverse operation that, for a convex function h and a given sub-gradient value g, computes the maximal interval of argument z on the ϵ-grid, $[z_L, z_R]$, such that for any $z \in [z_L, z_R]$, $g \in \partial h(z)$, i.e. $h'_L(z) \le g \le h'_R(z)$. The maximality of the interval implies that z_L satisfies $g \in \partial h(z_L)$ and $\forall z \le z_L - \epsilon$, $h'_R(z) < g$; similarly, z_R satisfies that $g \in \partial h(z_R)$ and $\forall z \ge z_R + \epsilon$, $h'_L(z) > g$. We denote the operation to compute z_L as $z_L := (\partial h)^{-1}_L(g)$, and the operation to compute z_R as $z_R := (\partial h)^{-1}_R(g)$.

In the above inverse operations, if the given sub-gradient value g is greater than the range of sub-gradients of function h, we define $(\partial h)^{-1}_L(g) = (\partial h)^{-1}_R(g) = +\infty$; if the given sub-gradient value g is smaller than the range of sub-gradients of function h, we define $(\partial h)_L^{-1}(g) = (\partial h)_R^{-1}(g) = -\infty$.

By the convexity of function h, for any two sub-gradient values $g_1 < g_2$, we have

$$(\partial h)_L^{-1}(g_1) \leq (\partial h)_R^{-1}(g_1) \leq (\partial h)_L^{-1}(g_2) \leq (\partial h)_R^{-1}(g_2).$$

In addition, if function h is strictly convex, for any sub-gradient value g, we have that $|(\partial h)_L^{-1}(g) - (\partial h)_R^{-1}(g)| \leq \epsilon$. In the (differentiable/strict)-algorithm for strictly convex separation functions, it is sufficient to always use either $(\partial h)_L^{-1}(g)$ or $(\partial h)_R^{-1}(g)$, because both values maintain ϵ-accuracy of solution. Hence for strictly convex functions, we define operation $(\partial h)^{-1}(g) = (\partial h)_L^{-1}(g)$.

For a given value of g, one can do a binary search to find the values of $(\partial h)_L^{-1}(g)$ and $(\partial h)_R^{-1}(g)$ on an interval of length $O(U)$. This is because function h is convex, thus the operations reduce to finding zeros of monotone sub-gradient functions. The complexity is $O(\log \frac{U}{\epsilon})$ for either operation. If function h is convex quadratic (ℓ_2-norm) or convex piecewise linear with constant number of pieces, including ℓ_1-norm, then one can compute the sub-gradient inverses in $O(1)$ time each.

An Equivalent Formulation of MRF-PATH

Both algorithms presented here work on an equivalent formulation of MRF-PATH that we introduce here. Since each x_i is bounded in $[-U, U]$, we have $x_i - x_{i+1}$ bounded in $[-2U, 2U]$. For each separation function h_i, we find its ϵ-accurate minimizer, c_i, in the interval $[-2U, 2U]$. We assume that $0 \in \partial h_i(c_i)$. This can be done by binary search over the interval in time $O(\log \frac{U}{\epsilon})$ for each separation function h_i, with a total $O(n \log \frac{U}{\epsilon})$ complexity for all the $n-1$ separation functions. Then for each separation function $h_i(x_i - x_{i+1})$, we split it into two convex separation functions, $h_{i,i+1}(z_{i,i+1})$ and $h_{i+1,i}(z_{i+1,i})$, with additional two decision variables $z_{i,i+1}$ and $z_{i+1,i}$:

$$\begin{aligned} h_{i,i+1}(z_{i,i+1}) &= h_i(z_{i,i+1} + c_i) - h_i(c_i), \\ h_{i+1,i}(z_{i+1,i}) &= h_i(c_i - z_{i+1,i}) - h_i(c_i). \end{aligned}$$

The split separation functions $h_{i,i+1}$ and $h_{i+1,i}$ have the following properties:

Proposition 20. *The (strict) convexity of function h_i implies the (strict) convexity of functions $h_{i,i+1}$ and $h_{i+1,i}$. Both functions $h_{i,i+1}(z_{i,i+1})$ and $h_{i+1,i}(z_{i+1,i})$ are nondecreasing for $z_{i,i+1}, z_{i+1,i} \geq 0$, and $h'_{i,i+1;R}(0), h'_{i+1,i;R}(0) \geq 0$.*

Proof. We prove the properties for $h_{i,i+1}$. The case of $h_{i+1,i}$ can be proved similarly.

If function h_i is (strictly) convex, so is function $h_i(z_{i,i+1} + c_i) - h_i(c_i)$, hence by definition, $h_{i,i+1}(z_{i,i+1})$ is also (strictly) convex.

By definition, we have $h'_{i,i+1}(z_{i,i+1}) = h'_i(z_{i,i+1} + c_i)$. Since $0 \in \partial h_i(c_i)$, by definition we have $h'_{i;L}(c_i) \leq 0 \leq h'_{i;R}(c_i)$. As a result, plugging $z_{i,i+1} = 0$ into $h'_{i,i+1}(z_{i,i+1})$, we have

$h'_{i,i+1;R}(0) = h'_{i;R}(c_i) \geq 0$. Combining the convexity of $h_{i,i+1}(z_{i,i+1})$ and $h'_{i,i+1;R}(0) \geq 0$, we have that for any $z_{i,i+1} > 0$, $h'_{i,i+1;R}(z_{i,i+1}) \geq h'_{i,i+1;L}(z_{i,i+1}) \geq h'_{i,i+1;R}(0) \geq 0$. This implies that $h_{i,i+1}(z_{i,i+1})$ is nondecreasing for $z_{i,i+1} \geq 0$. □

By splitting the separation functions, we introduce the following equivalent formulation of MRF-PATH (7.1):

$$\begin{aligned}
\text{(MRF-PATH)} \quad \min_{\substack{\{x_i\}_{i=1,\ldots,n} \\ \{z_{i,i+1}, z_{i+1,i}\}_{i=1,\ldots,n-1}}} \quad & \sum_{i=1}^{n} f_i(x_i) + \sum_{i=1}^{n-1} h_{i,i+1}(z_{i,i+1}) + \sum_{i=1}^{n-1} h_{i+1,i}(z_{i+1,i}) \\
s.t.\ & x_i - x_{i+1} \leq z_{i,i+1} + c_i,\ i = 1, \ldots, n-1 \\
& x_{i+1} - x_i \leq z_{i+1,i} - c_i,\ i = 1, \ldots, n-1 \\
& z_{i,i+1}, z_{i+1,i} \geq 0,\ i = 1, \ldots, n-1.
\end{aligned} \tag{7.6}$$

The equivalence between problem (7.6) and MRF-PATH (7.1) is proved in the following lemma:

Lemma 21. *MRF-PATH (7.1) and problem (7.6) share the same optimal solution.*

Proof. Let $P(\{x_i\}_{i=1,\ldots,n})$ be the objective value of MRF-PATH (7.1) for any given values $x_1, \ldots, x_n$. Let $\tilde{P}(\{z_{i,i+1}, z_{i+1,i}\}_{i=1,\ldots,n-1} | \{x_i\}_{i=1,\ldots,n})$ be the objective value of problem (7.6) such that, given the values of $\{x_i\}_{i=1,\ldots,n}$, the values of the z variables are selected to minimize the objective function. Note that given x_i and x_{i+1}, either $x_i - x_{i+1} - c_i$ or $x_{i+1} - x_i + c_i$ must be non-positive. Without loss of generality, let's assume that $x_i - x_{i+1} - c_i \geq 0$ and $x_{i+1} - x_i + c_i \leq 0$. Then due to the monotonicity of $h_{i,i+1}$ and $h_{i+1,i}$ on the nonnegative axis (Proposition 20), we have $z_{i,i+1} = x_i - x_{i+1} - c_i$ and $z_{i+1,i} = 0$ being the optimal values for this given pair of x_i and x_{i+1} that minimize the objective. Plugging these two values of $z_{i,i+1}$ and $z_{i+1,i}$ into $h_{i,i+1}(z_{i,i+1})$ and $h_{i+1,i}(z_{i+1,i})$ respectively, we have $h_{i,i+1}(z_{i,i+1}) = h_{i,i+1}(x_i - x_{i+1} - c_i) = h_i(x_i - x_{i+1} - c_i + c_i) - h_i(c_i) = h_i(x_i - x_{i+1}) - h_i(c_i)$, and $h_{i+1,i}(z_{i+1,i}) = h_{i+1,i}(0) = h_i(c_i) - h_i(c_i) = 0$. Applying the above analysis to all pairs of (x_i, x_{i+1}), we have:

$$\begin{aligned}
\tilde{P}(\{z_{i,i+1}, z_{i+1,i}\}_{i=1,\ldots,n-1} | \{x_i\}_{i=1,\ldots,n}) &= \sum_{i=1}^{n} f_i(x_i) + \sum_{i=1}^{n-1} h_i(x_i - x_{i+1}) - \sum_{i=1}^{n-1} h_i(c_i) \\
&= P(\{x_i\}_{i=1,\ldots,n}) - \sum_{i=1}^{n-1} h_i(c_i).
\end{aligned}$$

Since $\sum_{i=1}^{n-1} h_i(c_i)$ is a constant, thus solving $\min_{\{x_i\}_{i=1,\ldots,n}} P(\{x_i\}_{i=1,\ldots,n})$ is equivalent to solving $\min_{\{x_i\}_{i=1,\ldots,n}, \{z_{i,i+1}, z_{i+1,i}\}_{i=1,\ldots,n-1}} \tilde{P}(\{z_{i,i+1}, z_{i+1,i}\}_{i=1,\ldots,n-1} | \{x_i\}_{i=1,\ldots,n})$. Hence the lemma holds. □

Consequently, we refer to both problems (7.1) and (7.6) as MRF-PATH problem.

7.3 KKT-based Algorithms

The KKT Optimality Conditions and Overview of the Algorithms

We illustrate the KKT optimality conditions of MRF-PATH (7.6). For each constraint $x_i - x_{i+1} \leq z_{i,i+1} + c_i$, we introduce a dual variable $\lambda_{i,i+1}$; for each constraint $x_{i+1} - x_i \leq z_{i+1,i} - c_i$, we introduce a dual variable $\lambda_{i+1,i}$; for each constraint $z_{i,i+1} \geq 0$, we introduce a dual variable $\mu_{i,i+1}$ and for each constraint $z_{i+1,i} \geq 0$, we introduce a dual variable $\mu_{i+1,i}$. The KKT optimality conditions state that, $\left(\{x_i^*\}_i, \{z_{i,i+1}^*, z_{i+1,i}^*\}_i\right)$ is an optimal solution to MRF-PATH (7.6) if and only if there exist sub-gradients $\left(\{f_i'(x_i^*)\}_i, \{h_{i,i+1}'(z_{i,i+1}^*), h_{i+1,i}'(z_{i+1,i}^*)\}_i\right)$, and optimal values of the dual variables $\left(\{\lambda_{i,i+1}^*, \lambda_{i+1,i}^*\}_i, \{\mu_{i,i+1}^*, \mu_{i+1,i}^*\}_i\right)$ such that the following conditions hold:

$$\begin{cases} f_1'(x_1^*) + \lambda_{1,2}^* - \lambda_{2,1}^* = 0, \\ f_i'(x_i^*) - \lambda_{i-1,i}^* + \lambda_{i,i+1}^* + \lambda_{i,i-1}^* - \lambda_{i+1,i}^* = 0, \quad i = 2, \ldots, n-1 \\ f_n'(x_n^*) - \lambda_{n-1,n}^* + \lambda_{n,n-1}^* = 0, \end{cases} \tag{7.7}$$

$$\begin{cases} h_{i,i+1}'(z_{i,i+1}^*) - \lambda_{i,i+1}^* - \mu_{i,i+1}^* = 0, \quad i = 1, \ldots, n-1 \\ h_{i+1,i}'(z_{i+1,i}^*) - \lambda_{i+1,i}^* - \mu_{i+1,i}^* = 0, \quad i = 1, \ldots, n-1 \end{cases} \tag{7.8}$$

$$\begin{cases} x_i^* - x_{i+1}^* \leq z_{i,i+1}^* + c_i, \quad i = 1, \ldots, n-1 \\ x_{i+1}^* - x_i^* \leq z_{i+1,i}^* - c_i, \quad i = 1, \ldots, n-1 \\ z_{i,i+1}^*, z_{i+1,i}^* \geq 0, \quad i = 1, \ldots, n-1 \end{cases} \tag{7.9}$$

$$\begin{cases} \lambda_{i,i+1}^*, \lambda_{i+1,i}^* \geq 0, \quad i = 1, \ldots, n-1 \\ \mu_{i,i+1}^*, \mu_{i+1,i}^* \geq 0, \quad i = 1, \ldots, n-1 \end{cases} \tag{7.10}$$

$$\begin{cases} (x_i^* - x_{i+1}^* - z_{i,i+1}^* - c_i)\lambda_{i,i+1}^* = 0, \quad i = 1, \ldots, n-1 \\ (x_{i+1}^* - x_i^* - z_{i+1,i}^* + c_i)\lambda_{i+1,i}^* = 0, \quad i = 1, \ldots, n-1 \\ z_{i,i+1}^* \mu_{i,i+1}^* = 0, \quad i = 1, \ldots, n-1 \\ z_{i+1,i}^* \mu_{i+1,i}^* = 0, \quad i = 1, \ldots, n-1. \end{cases} \tag{7.11}$$

Equations (7.7) and (7.8) are stationarity conditions for x and z variables respectively, inequalities (7.9) and (7.10) are primal and dual feasibility conditions for the primal x, z variables and the dual λ, μ variables respectively, and equations (7.11) are complementary slackness conditions.

The algorithms use a trial-and-error procedure to find a solution to the above KKT optimality conditions: We first guess an optimal value of x_1^*. Then we propagate the value of the guessed x_1^* to determine the (ranges of) optimal values x_2^* to x_n^* that satisfy the KKT optimality conditions. If our guess is correct, i.e. all the equations and inequalities in the KKT optimality conditions are satisfied, we are done. If not, we will make a next guess to x_1^* and propagate the values to x_2^* until x_n^* again following the KKT optimality conditions. We show that, by the convexity of the deviation and separation functions, we can adopt a binary search procedure to converge our guesses to a correct optimal value of x_1 exponentially fast. The optimal value x_1^* also locks the (ranges of) optimal values of x_2^* to x_n^* by the propagation along the KKT optimality conditions.

(Differentiable/Strict)-Algorithm

Recall that the (differentiable/strict)-algorithm solves MRF-PATH of differentiable convex deviation functions and strictly convex separation functions. With differentiable convex deviation functions and strictly convex separation functions, we show that if we fix the value of x_1, then all the values of x_2 to x_n can be *uniquely* determined by the KKT optimality conditions. This is proved in the following propagation lemma:

Lemma 22. *Given any value of x_1, the KKT optimality conditions (7.7), (7.8), (7.9), (7.10) and (7.11) uniquely determine the other values of $x_2, \ldots, x_n$.*

Proof. We prove the lemma by induction on i from 1 to n. The case of $i = 1$ is trivial. Suppose the values of $x_1, \ldots, x_i$ are uniquely determined by x_1 for some i $(1 \leq i \leq n-1)$, we show that the value of x_{i+1} is also uniquely determined.

Adding up the equations of $j = 1, \ldots, i$ in the stationarity conditions (7.7), we have

$$\sum_{j=1}^{i} f_j'(x_j) = -\lambda_{i,i+1} + \lambda_{i+1,i}. \tag{7.12}$$

There are 5 different cases about equation (7.12), depending on the value of $\sum_{j=1}^{i} f_j'(x_j)$:

1. $\sum_{j=1}^{i} f_j'(x_j) > h'_{i+1,i;R}(0) \geq 0$:
   By the dual feasibility conditions (7.10), we have $\lambda_{i,i+1}, \lambda_{i+1,i} \geq 0$. Hence $\lambda_{i+1,i} \geq \sum_{j=1}^{i} f_j'(x_j) > 0$. If $\lambda_{i+1,i} > \sum_{j=1}^{i} f_j'(x_j)$, then $\lambda_{i,i+1} > 0$ as well. Then by the complementary slackness conditions (7.11), we have $x_i - x_{i+1} - z_{i,i+1} - c_i = 0$ and $x_{i+1} - x_i - z_{i+1,i} + c_i = 0$. These two equations imply that $z_{i,i+1} + z_{i+1,i} = 0$. On the other hand, by the primal feasibility conditions (7.9)), we have $z_{i,i+1}, z_{i+1,i} \geq 0$. As a result, it must be $z_{i,i+1} = z_{i+1,i} = 0$. Then we plug $z_{i+1,i} = 0$ into the stationarity condition on $z_{i+1,i}$ in (7.8), we have that there exists a sub-gradient $h'_{i+1,i}(0)$ such that $h'_{i+1,i}(0) = \lambda_{i+1,i} + \mu_{i+1,i} \geq \lambda_{i+1,i} > \sum_{j=1}^{i} f_j'(x_j) > h'_{i+1,i;R}(0)$ ($\mu_{i+1,i} \geq 0$ is due to the dual feasibility conditions (7.10)), which is a contradiction. Therefore we have $\lambda_{i+1,i} = \sum_{j=1}^{i} f_j'(x_j)$ and $\lambda_{i,i+1} = 0$. Then by the stationarity condition on $z_{i+1,i}$ in (7.8), we have that there exists a sub-gradient $h'_{i+1,i}(z_{i+1,i})$ such that $h'_{i+1,i}(z_{i+1,i}) = \lambda_{i+1,i} + \mu_{i+1,i} \geq \lambda_{i+1,i} = \sum_{j=1}^{i} f_j'(x_j) > h'_{i+1,i;R}(0)$. As a result, we have $z_{i+1,i} > 0$, and this implies that $\mu_{i+1,i} = 0$ by the complementary slackness conditions (7.11). And since $h_{i+1,i}(z_{i+1,i})$ is a strictly convex function, $z_{i+1,i}$ is thus uniquely determined by

$$z_{i+1,i} = (\partial h_{i+1,i})^{-1}(\lambda_{i+1,i}) = (\partial h_{i+1,i})^{-1}\big(\sum_{j=1}^{i} f_j'(x_j)\big).$$

Since $\lambda_{i+1,i} > 0$, by the complementary slackness conditions (7.11), we have x_{i+1} is uniquely determined by the equation $x_{i+1} = x_i + z_{i+1,i} - c_i$.

2. $0 < \sum_{j=1}^{i} f_j'(x_j) \leq h_{i+1,i;R}'(0)$:
This case exists only if $h_{i+1,i;R}'(0) > 0$. By the dual feasibility conditions (7.10), we still have $\lambda_{i+1,i}, \lambda_{i,i+1} \geq 0$, and thus $\lambda_{i+1,i} \geq \sum_{j=1}^{i} f_j'(x_j)$. If $\lambda_{i+1,i} > h_{i+1,i;R}'(0)$, we can derive the same contradiction as Case 1. As a result, it must be $0 < \sum_{j=1}^{i} f_j'(x_j) \leq \lambda_{i+1,i} \leq h_{i+1,i;R}'(0)$. Then we consider the stationarity conditions (7.8) on $z_{i+1,i}$: $h_{i+1,i}'(z_{i+1,i}) = \lambda_{i+1,i} + \mu_{i+1,i}$. If $\mu_{i+1,i} > 0$, then by the complementary slackness conditions (7.11), we have $z_{i+1,i} = 0$. If $\mu_{i+1,i} = 0$, then $h_{i+1,i}'(z_{i+1,i}) = \lambda_{i+1,i} \leq h_{i+1,i;R}'(0)$, which still implies that $z_{i+1,i} = 0$ by the strict convexity of $h_{i+1,i}$. In either case, by the complementary slackness conditions (7.11) with $\lambda_{i+1,i} > 0$, we have x_{i+1} is uniquely determined by the equation $x_{i+1} = x_i + z_{i+1,i} - c_i = x_i - c_i$.

3. $\sum_{j=1}^{i} f_j'(x_j) = 0$:
In this case, we have $\lambda_{i,i+1} = \lambda_{i+1,i}$. If both $\lambda_{i,i+1}$ and $\lambda_{i+1,i}$ are positive, then by the complementary slackness conditions (7.11) on $\lambda_{i,i+1}$ and $\lambda_{i+1,i}$, and the primal feasibility conditions (7.9) on $z_{i,i+1}$ and $z_{i+1,i}$, we have $z_{i,i+1} = z_{i+1,i} = 0$. As a result, $x_{i+1} = x_i - c_i$. If $\lambda_{i,i+1} = \lambda_{i+1,i} = 0$, we consider the stationarity conditions (7.8) on both $z_{i,i+1}$ and $z_{i+1,i}$. For $z_{i,i+1}$, we have that there exists a sub-gradient $h_{i,i+1}'(z_{i,i+1})$ such that $h_{i,i+1}'(z_{i,i+1}) = \mu_{i,i+1} \geq 0$. If $\mu_{i,i+1} \leq h_{i+1,i;R}'(0)$, by the strict convexity of $h_{i,i+1}$, we have $z_{i,i+1} = 0$; otherwise $\mu_{i,i+1} > h_{i+1,i;R}'(0)$, then the complementary slackness condition (7.11) also implies that $z_{i,i+1} = 0$. Therefore we always have $z_{i,i+1} = 0$. The same analysis shows that $z_{i+1,i} = 0$. Then by the primal feasibility conditions (7.9) on x_i and x_{i+1}, we have $x_{i+1} = x_i - c_i$ being uniquely determined.

4. $-h_{i,i+1;R}'(0) \leq \sum_{j=1}^{i} f_j'(x_j) < 0$:
This case exists only if $h_{i,i+1;R}'(0) > 0$, it is symmetric to case 2. By the dual feasibility conditions (7.10), we have $\lambda_{i+1,i} \geq 0$ and $\lambda_{i,i+1} \geq -\sum_{j=1}^{i} f_j'(x_j) > 0$. If $\lambda_{i,i+1} > h_{i,i+1;R}'(0)$, then $\lambda_{i+1,i} > 0$. Same as Case 1, it implies that $z_{i,i+1} = z_{i+1,i} = 0$. However, this violates the stationarity conditions (7.8) on $z_{i,i+1}$ because $h_{i,i+1}'(0) = \lambda_{i,i+1} + \mu_{i,i+1} \geq \lambda_{i,i+1} > h_{i,i+1;R}'(0)$. Therefore we have $0 < \lambda_{i,i+1} \leq h_{i,i+1;R}'(0)$. Then we consider the stationarity conditions (7.8) on $z_{i,i+1}$: $h_{i,i+1}'(z_{i,i+1}) = \lambda_{i,i+1} + \mu_{i,i+1}$. If $\mu_{i,i+1} > 0$, then by the complementary slackness conditions (7.11), we have $z_{i,i+1} = 0$. If $\mu_{i,i+1} = 0$, then $h_{i,i+1}'(z_{i,i+1}) = \lambda_{i,i+1} \leq h_{i,i+1;R}'(0)$, which still implies that $z_{i,i+1} = 0$ by the strict convexity of $h_{i,i+1}$. In either case, by the complementary slackness conditions (7.11) with $\lambda_{i,i+1} > 0$, we have x_{i+1} is uniquely determined by the equation $x_{i+1} = x_i - z_{i,i+1} - c_i = x_i - c_i$.

5. $\sum_{j=1}^{i} f_j'(x_j) < -h_{i,i+1;R}'(0) \leq 0$:
This case is symmetric to Case 1. Following the same reasoning in Case 1, we can show that $\lambda_{i,i+1} = -\sum_{j=1}^{i} f_j'(x_j) > h_{i,i+1;R}'(0) \geq 0$ and $\lambda_{i+1,i} = 0$. As a result, we have

$z_{i+1,i} > 0$, thus $\mu_{i,i+1} = 0$ by the complementary slackness conditions (7.11). And since $h_{i,i+1}(z_{i,i+1})$ is a strictly convex function, $z_{i,i+1}$ is uniquely determined by

$$z_{i,i+1} = (\partial h_{i,i+1})^{-1}(\lambda_{i,i+1}) = (\partial h_{i,i+1})^{-1}\big(- \sum_{j=1}^{i} f'_j(x_j)\big).$$

Since $\lambda_{i,i+1} > 0$, by the complementary slackness conditions (7.11), we have x_{i+1} is uniquely determined by the equation $x_{i+1} = x_i - z_{i,i+1} - c_i$.

This completes the proof for the case of $i+1$. □

Lemma 22 implies that, given a value of x_1, the values of x_2 to x_n are uniquely determined by the following iterative equations:

$$x_{i+1} = x_i + z_i - c_i, \quad i = 1, \dots, n-1 \tag{7.13}$$

where

$$z_i = \begin{cases} (\partial h_{i+1,i})^{-1}(\sum_{j=1}^{i} f'_j(x_j)), & \text{if } \sum_{j=1}^{i} f'_j(x_j) > h'_{i+1,i;R}(0) \\ 0, \quad \text{if } -h'_{i,i+1;R}(0) \le \sum_{j=1}^{i} f'_j(x_j) \le h'_{i+1,i;R}(0) \\ -(\partial h_{i,i+1})^{-1}(-\sum_{j=1}^{i} f'_j(x_j)), & \text{if } \sum_{j=1}^{i} f'_j(x_j) < -h'_{i,i+1;R}(0). \end{cases} \tag{7.14}$$

Based on the convexity of functions $f_i(x_i)$, $h_{i,i+1}(z_{i,i+1})$, and $h_{i+1,i}(z_{i+1,i})$, we have the following monotonicity property for any two sequences of $x_1, \dots, x_n$ generated by equations (7.13) and (7.14):

Corollary 23. *Let $x_1^{(1)} < x_1^{(2)}$ be any two given values of variable x_1. Let $(x_1^{(1)}, x_2^{(1)}, \dots, x_n^{(1)})$ and $(x_1^{(2)}, x_2^{(2)}, \dots, x_n^{(2)})$ be the respective sequence of x values determined by the value of x_1 and equations (7.13) and (7.14). Then we have $x_i^{(1)} < x_i^{(2)}$ for all $i = 1, \dots, n$.*

Proof. We prove the corollary by induction on i ($i = 1, \dots, n$). The case of $i = 1$ holds. Suppose $x_j^{(1)} < x_j^{(2)}$ for all $j = 1, \dots, i$ for some i ($1 \le i \le n-1$), we show that $x_{i+1}^{(1)} < x_{i+1}^{(2)}$.

Due to the convexity of $h_{i,i+1}$ and $h_{i+1,i}$, equation (7.14) implies that z_i is a nondecreasing function of $\sum_{j=1}^{i} f'_j(x_j)$. On the other hand, since all f_j ($j = 1, \dots, i$) functions are convex, by the induction hypothesis, we have $f'_j(x_j^{(1)}) \le f'_j(x_j^{(2)})$ for $j = 1, \dots, i$. Hence $\sum_{j=1}^{i} f'_j(x_j^{(1)}) \le \sum_{j=1}^{i} f'_j(x_j^{(2)})$. As a result, we have $z_i^{(1)} \le z_i^{(2)}$. Since we have $x_i^{(1)} < x_i^{(2)}$ by the induction hypothesis, we have $x_{i+1}^{(1)} = x_i^{(1)} + z_i^{(1)} - c_i < x_i^{(2)} + z_i^{(2)} - c_i = x_{i+1}^{(2)}$. □

The only equation in the KKT optimality conditions that a given sequence of $x_1, \dots, x_n$ determined by equations (7.13) and (7.14) may violate is the last stationarity condition (7.7) for x_n. This is because x_n, $\lambda_{n-1,n}$ and $\lambda_{n,n-1}$ are determined in the step of computing x_n from x_{n-1}, based on the equation $\sum_{j=1}^{n-1} f'_j(x_j) = \lambda_{n,n-1} - \lambda_{n-1,n}$, however, the generated values of x_n, $\lambda_{n-1,n}$ and $\lambda_{n,n-1}$ do not necessarily satisfy the last stationarity condition for

x_n, $f'_n(x_n) - \lambda_{n-1,n} + \lambda_{n,n-1} = 0$. On the other hand, we observe that if we sum up all the stationarity conditions (7.7) for the x variables, we have:

$$\sum_{i=1}^{n} f'_i(x_i^*) = 0. \tag{7.15}$$

The equation for x_n in the stationarity conditions (7.7) can be equivalently replaced by equation (7.15). Hence a sequence of $x_1, \ldots, x_n$ determined by equations (7.13) and (7.14) satisfy the KKT optimality conditions if and only if equation (7.15) also holds.

The above analysis implies a trial-and-error binary search algorithm to solve the KKT optimality conditions. In every iteration, we try a value of x_1, and compute the values of x_2 to x_n based on equations (7.13) and (7.14). Then we check whether equation (7.15) holds for the generated sequence of $x_1, \ldots, x_n$. If yes, then the generated sequence of $x_1, \ldots, x_n$ satisfies the KKT optimality conditions, thus it is an optimal solution to MRF-PATH. Otherwise, we determine the next value of x_1 to try based on the sign of $\sum_{i=1}^{n} f'_i(x_i)$ of the currently generated sequence of $x_1, \ldots, x_n$: If $\sum_{i=1}^{n} f'_i(x_i) > 0$, we check a smaller value of x_1; If $\sum_{i=1}^{n} f'_i(x_i) < 0$, we check a larger value of x_1. The binary search efficiently locks the optimal value of x_1, so are the optimal values of x_2 to x_n by equations (7.13) and (7.14). The complete algorithm is presented as follows:

Algorithm for MRF-PATH of differentiable convex deviation functions $f_i(x_i)$ and strictly convex separation functions $h_i(x_i - x_{i+1})$:

Step 0: Solve $c_i = \text{argmin}_{-2U \le z_i \le 2U} h_i(z_i)$ for $i = 1, \ldots, n-1$. Initialize the lower and upper bounds of the search region of an optimal value of x_1 as $\ell = -U$ and $u = U$ respectively.

Step 1: Set $x_1 = \lfloor \frac{\ell+u}{2\epsilon} \rfloor \epsilon$. Compute the values of $x_2, \ldots, x_n$ based on iterative equations (7.13) and (7.14):

$$x_{i+1} = x_i + z_i - c_i, \quad i = 1, \ldots, n-1$$

where

$$z_i = \begin{cases} (\partial h_{i+1,i})^{-1}(\sum_{j=1}^{i} f'_j(x_j)), & \text{if } \sum_{j=1}^{i} f'_j(x_j) > h'_{i+1,i;R}(0) \\ 0, \quad \text{if } -h'_{i,i+1;R}(0) \le \sum_{j=1}^{i} f'_j(x_j) \le h'_{i+1,i;R}(0) \\ -(\partial h_{i,i+1})^{-1}(-\sum_{j=1}^{i} f'_j(x_j)), & \text{if } \sum_{j=1}^{i} f'_j(x_j) < -h'_{i,i+1;R}(0). \end{cases}$$

Step 2: If $u - \ell < \epsilon$ or $\sum_{i=1}^{n} f'_i(x_i) = 0$, return $(x_1, x_2, \ldots, x_n)$ and stop.

Step 3: If $\sum_{i=1}^{n} f'_i(x_i) < 0$, set $\ell = x_1 + \epsilon$; otherwise set $u = x_1$. Go to **Step 1**.

The number of different x_1 values we need to check in the algorithm is $O(\log \frac{U}{\epsilon})$. For each x_1 value fixed, the complexity to compute the values of $x_2, \ldots, x_n$ by equations (7.13) and (7.14) is $O(n \log \frac{U}{\epsilon})$, where the $O(\log \frac{U}{\epsilon})$ term corresponds to the complexity of computing sub-gradient inverse of $h_{i+1,i}$ or $h_{i,i+1}$ functions to compute z_i. Hence the complexity of the Step 1 to Step 3 is $O(n \log^2 \frac{U}{\epsilon})$. At Step 0, it takes $O(n \log \frac{U}{\epsilon})$ time to compute the ϵ-accurate c_i values for $i = 1, \ldots, n-1$. Thus we have:

Theorem 24. *The MRF-PATH problem of differentiable convex deviation functions and strictly convex separation functions is solved in $O(n \log^2 \frac{U}{\epsilon})$ time.*

Note that if the separation functions h_i are convex quadratic, so are the functions $h_{i,i+1}$ and $h_{i+1,i}$, then the complexity of sub-gradient inverse is $O(1)$, thus the (differentiable/strict)-algorithm can be sped-up to $O(n \log \frac{U}{\epsilon})$ complexity.

(Convex/Convex)-Algorithm

We extend the (differentiable/strict)-algorithm to solve MRF-PATH of arbitrary convex deviation and separation functions, leading to the (convex/convex)-algorithm.

The impact of the more general assumptions on deviation and separation functions is two-fold: the non-differentiability of $f_i(x_i)$ implies that a given x_i value corresponds to a non-singleton interval of sub-gradients $\partial f_i(x_i) = [f'_{i;L}(x_i), f'_{i;R}(x_i)]$, instead of a unique derivative $f'(x_i)$ in the differentiable case; the non-strict convexity of $h_{i,i+1}$ (and $h_{i+1,i}$) implies that a sub-gradient value g can be inversely mapped to a non-singleton interval of arguments $[(\partial h_{i,i+1})^{-1}_L(g), (\partial h_{i,i+1})^{-1}_R(g)]$ (and $[(\partial h_{i+1,i})^{-1}_L(g), (\partial h_{i+1,i})^{-1}_R(g)]$), instead of a unique z argument $(\partial h_{i,i+1})^{-1}(g)$ (and $(\partial h_{i+1,i})^{-1}(g)$). Both observations imply that, based on the KKT optimality conditions, a given value of x_1 does not lock the other variables $x_2, \ldots, x_n$ to a unique value respectively, but to unique a range instead.

We first show that, when the sequence of x_1 to x_i and the corresponding sequence of sub-gradients $f'_1(x_1)$ to $f'_i(x_i)$ are fixed, we can extend the proof of Lemma 22 to determine the range of x_{i+1} that satisfies the KKT optimality conditions. This is shown in the following lemma:

Lemma 25. *Given a sequences of x_1 to x_i (for $i : 1 \leq i \leq n-1$), and the corresponding sub-gradients $f'_1(x_1)$ to $f'_i(x_i)$, we can determine the range of x_{i+1}, following the KKT optimality conditions (7.7), (7.8), (7.9), (7.10) and (7.11), with equations (7.16) and (7.17):*

$$x_{i+1} = x_i + z_i - c_i, \tag{7.16}$$

where

$$z_i \in \begin{cases} \Big[(\partial h_{i+1,i})^{-1}_L\big(\sum_{j=1}^i f'_j(x_j)\big), (\partial h_{i+1,i})^{-1}_R\big(\sum_{j=1}^i f'_j(x_j)\big)\Big], & \text{if } \sum_{j=1}^i f'_j(x_j) > 0 \\ \Big[-(\partial h_{i,i+1})^{-1}_R(0), (\partial h_{i+1,i})^{-1}_R(0)\Big], & \text{if } \sum_{j=1}^i f'_j(x_j) = 0 \\ \Big[-(\partial h_{i,i+1})^{-1}_R\big(-\sum_{j=1}^i f'_j(x_j)\big), -(\partial h_{i,i+1})^{-1}_L\big(-\sum_{j=1}^i f'_j(x_j)\big)\Big], & \text{if } \sum_{j=1}^i f'_j(x_j) < 0. \end{cases} \tag{7.17}$$

Proof. The proof is similar to the induction steps in the proof of Lemma 22. Recall that by summing up the equations of $j = 1, \ldots, i$ in the stationarity conditions (7.7), we have

$$\sum_{j=1}^{i} f'_j(x_j) = -\lambda_{i,i+1} + \lambda_{i+1,i}. \tag{7.18}$$

There are 5 different cases about equation (7.18), depending on the value of $\sum_{j=1}^{i} f'_j(x_j)$:

1. $\sum_{j=1}^{i} f'_j(x_j) > h'_{i+1,i;R}(0) \geq 0$:
   Similar to Case 1 in Lemma 22, we first have that $\lambda_{i+1,i} = \sum_{j=1}^{i} f'_j(x_j) > h'_{i+1,i;R}(0)$ and $\lambda_{i,i+1} = 0$. Then by the complementary slackness conditions (7.11) for $\lambda_{i+1,i}$, we have $x_{i+1} = x_i + z_{i+1,i} - c_i$.

   To figure out the range of $z_{i+1,i}$, we look at the stationarity condition on $z_{i+1,i}$ in (7.8). We have that $h'_{i+1,i}(z_{i+1,i}) = \lambda_{i+1,i} + \mu_{i+1,i} \geq \lambda_{i+1,i} > h'_{i+1,i;R}(0)$. Hence $z_{i+1,i} > 0$, and this implies that $\mu_{i+1,i} = 0$ by the complementary slackness conditions (7.11). Hence $h'_{i+1,i}(z_{i+1,i}) = \lambda_{i+1,i} = \sum_{j=1}^{i} f'_j(x_j)$. Therefore the range of $z_{i+1,i}$ is

$$z_{i+1,i} \in \left[(\partial h_{i+1,i})_L^{-1}\big(\sum_{j=1}^{i} f'_j(x_j)\big), (\partial h_{i+1,i})_R^{-1}\big(\sum_{j=1}^{i} f'_j(x_j)\big)\right].$$

2. $0 < \sum_{j=1}^{i} f'_j(x_j) \leq h'_{i+1,i;R}(0)$:
   This case exists only if $h'_{i+1,i;R}(0) > 0$. Similar to Case 2 in Lemma 22, we have that $0 < \sum_{j=1}^{i} f'_j(x_j) \leq \lambda_{i+1,i} \leq h'_{i+1,i;R}(0)$. Thus by the complementary slackness conditions (7.11) for $\lambda_{i+1,i}$, we have $x_{i+1} = x_i + z_{i+1,i} - c_i$.

   To figure out the range of $z_{i+1,i}$, we consider two cases. If $\lambda_{i+1,i} > \sum_{j=1}^{i} f'_j(x_j)$, then $\lambda_{i,i+1} > 0$ as well, by equation (7.18). This implies that $z_{i+1,i} = 0$ by the complementary slackness conditions (7.11) on $\lambda_{i+1,i}$ and $\lambda_{i,i+1}$, and the primal feasibility conditions (7.9) on $z_{i+1,i}$ and $z_{i,i+1}$. Otherwise $\lambda_{i+1,i} = \sum_{j=1}^{i} f'_j(x_j)$ and $\lambda_{i,i+1} = 0$. Then we consider the stationarity conditions (7.8) on $z_{i+1,i}$: $h'_{i+1,i}(z_{i+1,i}) = \lambda_{i+1,i} + \mu_{i+1,i}$. If $\mu_{i+1,i} > 0$, then by the complementary slackness conditions (7.11), we have $z_{i+1,i} = 0$. Otherwise $\mu_{i+1,i} = 0$, and $h'_{i+1,i}(z_{i+1,i}) = \lambda_{i+1,i} = \sum_{j=1}^{i} f'_j(x_j)$. Therefore the range of $z_{i+1,i}$ is

$$z_{i+1,i} \in \left[(\partial h_{i+1,i})_L^{-1}\big(\sum_{j=1}^{i} f'_j(x_j)\big), (\partial h_{i+1,i})_R^{-1}\big(\sum_{j=1}^{i} f'_j(x_j)\big)\right].$$

   Note that as $0 < \sum_{j=1}^{i} f'_j(x_j) \leq h'_{i+1,i;R}(0)$, the above interval contains value 0.

3. $\sum_{j=1}^{i} f'_j(x_j) = 0$:
   This is similar to Case 3 in Lemma 22. In this case, we have $\lambda_{i,i+1} = \lambda_{i+1,i}$. If $\lambda_{i,i+1} = \lambda_{i+1,i} > 0$, then we have $z_{i+1,i} = z_{i,i+1} = 0$, by the complementary slackness

conditions (7.11) on $\lambda_{i,i+1}$ and $\lambda_{i+1,i}$, and the primal feasibility conditions (7.9) on $z_{i,i+1}$ and $z_{i+1,i}$. Thus $x_{i+1} = x_i - c_i$. If $\lambda_{i+1,i} = \lambda_{i,i+1} = 0$, we have, by the primal feasibility conditions (7.9):

$$x_i - z_{i,i+1} - c_i \leq x_{i+1} \leq x_i + z_{i+1,i} - c_i.$$

We consider the stationarity conditions (7.8) on both $z_{i+1,i}$ and $z_{i,i+1}$ to figure out the ranges of $z_{i+1,i}$ and $z_{i,i+1}$ respectively. For $z_{i+1,i}$, we have $h'_{i+1,i}(z_{i+1,i}) = \mu_{i+1,i}$. If $\mu_{i+1,i} > 0$, then by the complementary slackness conditions (7.11), we have $z_{i+1,i} = 0$. Otherwise, $\mu_{i+1,i} = 0$, and $h'_{i+1,i}(z_{i+1,i}) = 0$. Hence the range of $z_{i+1,i}$ is

$$z_{i+1,i} \in \Big[(\partial h_{i+1,i})_L^{-1}(0), (\partial h_{i+1,i})_R^{-1}(0)\Big].$$

Similarly, for $z_{i,i+1}$, we have $h'_{i,i+1}(z_{i,i+1}) = \mu_{i,i+1}$. If $\mu_{i,i+1} > 0$, we also have $z_{i,i+1} = 0$. Otherwise, $\mu_{i,i+1} = 0$, and $h'_{i,i+1}(z_{i,i+1}) = 0$. Hence the range of $z_{i,i+1}$ is

$$z_{i,i+1} \in \Big[(\partial h_{i,i+1})_L^{-1}(0), (\partial h_{i,i+1})_R^{-1}(0)\Big].$$

As a result, we can summarize the range of x_{i+1} as:

$$x_{i+1} = x_i + z_i - c_i, \text{ where } z_i \in \Big[-(\partial h_{i,i+1})_R^{-1}(0), (\partial h_{i+1,i})_R^{-1}(0)\Big].$$

4. $-h'_{i,i+1;R}(0) \leq \sum_{j=1}^{i} f'_j(x_j) < 0$:
   This case exists only if $h'_{i,i+1;R}(0) > 0$. This case is symmetric to the Case 2 here. Similar to Case 4 in Lemma 22, we have that $0 < -\sum_{j=1}^{i} f'_j(x_j) \leq \lambda_{i,i+1} \leq h'_{i,i+1;R}(0)$. Thus by the complementary slackness conditions (7.11) for $\lambda_{i,i+1}$, we have $x_{i+1} = x_i - z_{i,i+1} - c_i$.

   To figure out the range of $z_{i,i+1}$, we consider two cases. If $\lambda_{i,i+1} > -\sum_{j=1}^{i} f'_j(x_j)$, then $\lambda_{i+1,i} > 0$ as well, by equation (7.18). This implies that $z_{i,i+1} = 0$ by the complementary slackness conditions (7.11) on $\lambda_{i,i+1}$ and $\lambda_{i+1,i}$, and the primal feasibility conditions (7.9) on $z_{i,i+1}$ and $z_{i+1,i}$. Otherwise $\lambda_{i,i+1} = -\sum_{j=1}^{i} f'_j(x_j)$ and $\lambda_{i+1,i} = 0$. Then we consider the stationarity conditions (7.8) on $z_{i,i+1}$: $h'_{i,i+1}(z_{i,i+1}) = \lambda_{i,i+1} + \mu_{i,i+1}$. If $\mu_{i,i+1} > 0$, then by the complementary slackness conditions (7.11), we have $z_{i,i+1} = 0$. Otherwise $\mu_{i,i+1} = 0$, and $h'_{i,i+1}(z_{i,i+1}) = \lambda_{i,i+1} = -\sum_{j=1}^{i} f'_j(x_j)$. Therefore the range of $z_{i,i+1}$ is

$$z_{i,i+1} \in \Big[(\partial h_{i,i+1})_L^{-1}\big(-\sum_{j=1}^{i} f'_j(x_j)\big), (\partial h_{i,i+1})_R^{-1}\big(-\sum_{j=1}^{i} f'_j(x_j)\big)\Big].$$

   Note that as $0 < -\sum_{j=1}^{i} f'_j(x_j) \leq h'_{i,i+1;R}(0)$, the above interval contains value 0.

5. $\sum_{j=1}^{i} f'_j(x_j) < -h'_{i,i+1;R}(0) \leq 0$:
   This case is symmetric to the Case 1 here. Similar to Case 5 in Lemma 22, we first have that $\lambda_{i,i+1} = -\sum_{j=1}^{i} f'_j(x_j) > h'_{i,i+1;R}(0)$ and $\lambda_{i+1,i} = 0$. Then by the complementary slackness conditions (7.11) for $\lambda_{i,i+1}$, we have $x_{i+1} = x_i - z_{i,i+1} - c_i$

   To figure out the range of $z_{i,i+1}$, we look at the stationarity condition on $z_{i,i+1}$ in (7.8). We have that $h'_{i,i+1}(z_{i,i+1}) = \lambda_{i,i+1} + \mu_{i,i+1} \geq \lambda_{i,i+1} > h'_{i,i+1;R}(0)$. Hence $z_{i,i+1} > 0$, and this implies that $\mu_{i,i+1} = 0$, by the complementary slackness conditions (7.11). Hence $h'_{i,i+1}(z_{i,i+1}) = \lambda_{i,i+1} = -\sum_{j=1}^{i} f'_j(x_j)$. Therefore the range of $z_{i,i+1}$ is

$$z_{i,i+1} \in \left[(\partial h_{i,i+1})^{-1}_L\big(-\sum_{j=1}^{i} f'_j(x_j)\big), (\partial h_{i,i+1})^{-1}_R\big(-\sum_{j=1}^{i} f'_j(x_j)\big)\right].$$

Equations (7.16) and (7.17) are a compact summary of results of the above 5 cases. □

Lemma 25 is an extension of the propagation lemma (Lemma 22) in that, instead of determining a unique value of x_{i+1}, it determines a range of x_{i+1}. Note that the range of x_{i+1} in Lemma 25 is achieved for *any* given sequence of sub-gradients $f'_1(x_1)$ to $f'_i(x_i)$. Since each $f_i(x_i)$ could be non-differentiable, there could be an interval of sub-gradients for each given x_i. Therefore to obtain a complete range of x_{i+1} values following the KKT optimality conditions, one needs to apply Lemma 25 on *all* sequences of sub-gradients of $f'_1(x_1), \ldots, f'_i(x_i)$.

Given an initial value of x_1, we compute the complete ranges of $x_2, \ldots, x_n$ iteratively. The intuitive idea is that the upper bound of the complete range of x_{i+1} comes from the upper bounds of x_i and z_i according to equation (7.16). According to equation (7.17), the upper bound of z_i depends on the upper bounds of the sub-gradients of functions $f_1, \ldots, f_i$, evaluated at the upper bound values of $x_1, \ldots, x_i$ respectively. Similarly, the lower bound of the complete range of x_i comes from the lower bounds of x_i and z_i. The lower bound of z_i depends on the lower bounds of the sub-gradients of functions $f_1, \ldots, f_i$, evaluated at the lower bound values of $x_1, \ldots, x_i$ respectively.

Formally, let $[\ell_{kkt,i}, u_{kkt,i}]$ be the complete range of x_i determined by the KKT optimality conditions and the initial value of x_1. Then $\ell_{kkt,i}$ and $u_{kkt,i}$ are defined iteratively as follows:

$$\begin{aligned} &\ell_{kkt,1} = u_{kkt,1} = x_1, \\ &\begin{cases} u_{kkt,i+1} = u_{kkt,i} + z_{i;R} - c_i, \\ \ell_{kkt,i+1} = \ell_{kkt,i} + z_{i;L} - c_i, \end{cases} \qquad i = 1, \ldots, n-1 \end{aligned} \tag{7.19}$$

where

$$z_{i;R} = \begin{cases} (\partial h_{i+1,i})_R^{-1}\big(\sum_{j=1}^{i} f'_{j;R}(u_{kkt,j})\big), & \text{if } \sum_{j=1}^{i} f'_{j;R}(u_{kkt,j}) \geq 0 \\ \\ -(\partial h_{i,i+1})_L^{-1}\big(-\sum_{j=1}^{i} f'_{j;R}(u_{kkt,j})\big), & \text{if } \sum_{j=1}^{i} f'_{j;R}(u_{kkt,j}) < 0 \end{cases} \tag{7.20}$$

$$z_{i;L} = \begin{cases} (\partial h_{i+1,i})_L^{-1}\big(\sum_{j=1}^{i} f'_{j;L}(\ell_{kkt,j})\big), & \text{if } \sum_{j=1}^{i} f'_{j;L}(\ell_{kkt,j}) > 0 \\ \\ -(\partial h_{i,i+1})_R^{-1}\big(-\sum_{j=1}^{i} f'_{j;L}(\ell_{kkt,j})\big), & \text{if } \sum_{j=1}^{i} f'_{j;L}(\ell_{kkt,j}) \leq 0. \end{cases} \tag{7.21}$$

The validity of the above defined upper and lower bounds is proved in the following lemma:

Lemma 26. *Given any value of x_1, the range of variables x_i ($i = 1, \ldots, n$), following the KKT optimality conditions (7.7), (7.8), (7.9), (7.10) and (7.11), is $[\ell_{kkt,i}, u_{kkt,i}]$, where $u_{kkt,i}$ and $\ell_{kkt,i}$ are defined iteratively from $u_{kkt,i-1}$ and $\ell_{kkt,i-1}$ respectively according to equations (7.19), (7.20) and (7.21).*

Proof. We prove the lemma by induction on i. The range holds for $i = 1$ since $\ell_{kkt,1} = u_{kkt,1} = x_1$. Suppose the range holds for $j = 1, \ldots, i$ for some i ($1 \leq i \leq n-1$). We prove the range for x_{i+1}. Consider any sequence of x_1 to x_i that is generated according to the KKT optimality conditions from the given value of x_1, with the corresponding sequence of sub-gradients $f'_1(x_1)$ to $f'_i(x_i)$. According to Lemma 25, we have $x_{i+1} = x_i + z_i - c_i$, where z_i is defined in equation (7.17).

We first show the validity of the upper bound $u_{kkt,i+1}$. Based on the induction hypothesis, we have $x_j \leq u_{kkt,j}$ for $j = 1, \ldots, i$. Due to the convexity of functions f_1 to f_i, we have $f'_j(x_j) \leq f'_{j;R}(u_{kkt,j})$. Hence $\sum_{j=1}^{i} f'_j(x_j) \leq \sum_{j=1}^{i} f'_{j;R}(u_{kkt,j})$. Comparing equation (7.17) for z_i and equation (7.20) for $z_{i;R}$, due to convexity of functions $h_{i+1,i}$ and $h_{i,i+1}$, we have $z_i \leq z_{i;R}$. As a result, $x_{i+1} = x_i + z_i - c_i \leq u_{kkt,i} + z_{i;R} - c_i = u_{kkt,i+1}$. All inequalities in this derivation can take equalities.

The validity of the lower bound $\ell_{kkt,i+1}$ can be proved similarly. Based on the induction hypothesis, we have $x_j \geq \ell_{kkt,j}$ for $j = 1, \ldots, i$. Due to the convexity of functions f_1 to f_i, we have $f'_j(x_j) \geq f'_{j;L}(\ell_{kkt,j})$. Hence $\sum_{j=1}^{i} f'_j(x_j) \geq \sum_{j=1}^{i} f'_{j;L}(\ell_{kkt,j})$. Comparing equation (7.17) for z_i and equation (7.21) for $z_{i;L}$, due to convexity of functions $h_{i+1,i}$ and $h_{i,i+1}$, we have $z_i \geq z_{i;L}$. As a result, $x_{i+1} = x_i + z_i - c_i \geq \ell_{kkt,i} + z_{i;L} - c_i = \ell_{kkt,i+1}$. All inequalities in this derivation can take equalities. $\square$

The ranges $[\ell_{kkt,i}, u_{kkt,i}]_{i=1,\ldots,n}$ have a similar monotonicity property as Corollary 23, due to the convexity of $f_i(x_i)$, $h_{i,i+1}(z_{i,i+1})$, and $h_{i+1,i}(z_{i+1,i})$:

Corollary 27. *Let $x_1^{(1)} < x_1^{(2)}$ be any two given values of variable x_1. Let*

$$\left([\ell_{kkt,1}^{(1)}, u_{kkt,1}^{(1)}], [\ell_{kkt,2}^{(1)}, u_{kkt,2}^{(1)}], \ldots, [\ell_{kkt,n}^{(1)}, u_{kkt,n}^{(1)}]\right),$$
$$\left([\ell_{kkt,1}^{(2)}, u_{kkt,1}^{(2)}], [\ell_{kkt,2}^{(2)}, u_{kkt,2}^{(2)}], \ldots, [\ell_{kkt,n}^{(2)}, u_{kkt,n}^{(2)}]\right),$$

be the respective sequence of ranges of x values determined by the value of x_1 and equations (7.19), (7.20) and (7.21). Then we have, for all $i = 1, \ldots, n$:

$$u_{kkt,i}^{(1)} < u_{kkt,i}^{(2)}, \tag{7.22}$$
$$\ell_{kkt,i}^{(1)} < \ell_{kkt,i}^{(2)}. \tag{7.23}$$

Proof. We show the proof of Inequalities (7.22). Inequalities (7.23) are proved in a similar way.

The proof is by induction on i. The inequality is true for $i = 1$ because $u_{kkt,1}^{(1)} = x_1^{(1)} < x_1^{(2)} = u_{kkt,1}^{(2)}$. Suppose the result is true for all $j = 1, \ldots, i$ for some i $(1 \leq i \leq n-1)$. We prove that it is also true for $i + 1$. By the induction hypothesis and the definition equations (7.19), it is sufficient to prove that $z_{i;R}^{(1)} \leq z_{i;R}^{(2)}$. Equation (7.20) shows that $z_{i;R}$ is an nondecreasing function of the term $\sum_{j=1}^{i} f'_{j;R}(u_{kkt,j})$, by the convexity of functions $h_{i,i+1}$ and $h_{i+1,i}$. On the other hand, by the induction hypothesis and the convexity of all the f_j functions, we have $\sum_{j=1}^{i} f'_{j;R}(u_{kkt,j}^{(1)}) \leq \sum_{j=1}^{i} f'_{j;R}(u_{kkt,j}^{(2)})$. As a result, $z_{i;R}^{(1)} \leq z_{i;R}^{(2)}$. Hence

$$u_{kkt,i+1}^{(1)} = u_{kkt,i}^{(1)} + z_{i;R}^{(1)} - c_i < u_{kkt,i}^{(2)} + z_{i;R}^{(2)} - c_i = u_{kkt,i+1}^{(2)}.$$

□

Finally, Corollary 27 implies the following lemma, which forms the basis of our algorithm:

Lemma 28. *For a sequence of intervals $([\ell_{kkt,i}, u_{kkt,i}])_{i=1,\ldots,n}$ for variables $\{x_i\}_{i=1,\ldots,n}$ computed iteratively according to (7.19), (7.20) and (7.21) from an initial value of x_1, we have*

1. *If $\sum_{i=1}^{n} f'_{i;R}(u_{kkt,i}) < 0$, then there exists an optimal value x_1^* such that $x_1^* > x_1$.*
2. *If $\sum_{i=1}^{n} f'_{i;L}(\ell_{kkt,i}) > 0$, then there exists an optimal value x_1^* such that $x_1^* < x_1$.*
3. *If $\sum_{i=1}^{n} f'_{i;L}(\ell_{kkt,i}) \leq 0 \leq \sum_{i=1}^{n} f'_{i;R}(u_{kkt,i})$, then the value of x_1 is optimal.*

Proof. Summing up the stationarity conditions (7.7), we have an equation analogous to (7.15):

$$\sum_{i=1}^{n} f'_i(x_i^*) = 0,$$

for some sub-gradients $f'_i(x_i^*) \in \partial f_i(x_i^*)$ $(i = 1, \ldots, n)$.

If $\sum_{i=1}^n f'_{i;R}(u_{kkt,i}) < 0$, then consider *any* x'_1 such that $x'_1 \le x_1$, and *any* sequence $x'_1, x'_2, \ldots, x'_n$ generated by the KKT optimality conditions according to Lemma 25, where $x'_i \in [\ell'_{kkt,i}, u'_{kkt,i}]$. By Corollary 27, we have $x'_i \le u'_{kkt,i} \le u_{kkt,i}$ for $i = 1, \ldots, n$. Hence by the convexity of functions f_1 to f_n, for the corresponding sequence of sub-gradients $f'_1(x'_1), \ldots, f'_n(x'_n)$, we have

$$\sum_{i=1}^n f'_i(x'_i) \le \sum_{i=1}^n f'_{i;R}(u'_{kkt,i}) \le \sum_{i=1}^n f'_{i;R}(u_{kkt,i}) < 0.$$

This implies that no values smaller than or equal to x_1 are optimal. Hence there exists an optimal value x_1^* such that $x_1^* > x_1$.

If $\sum_{i=1}^n f'_{i;L}(\ell_{kkt,i}) > 0$, then consider *any* x'_1 such that $x'_1 \ge x_1$, and *any* sequence $x'_1, x'_2, \ldots, x'_n$ generated by the KKT optimality conditions according to Lemma 25, where $x'_i \in [\ell'_{kkt,i}, u'_{kkt,i}]$. By Corollary 27, we have $x'_i \ge \ell'_{kkt,i} \ge \ell_{kkt,i}$ for $i = 1, \ldots, n$. Hence by the convexity of functions f_1 to f_n, for the corresponding sequence of sub-gradients $f'_1(x'_1), \ldots, f'_n(x'_n)$, we have

$$\sum_{i=1}^n f'_i(x'_i) \ge \sum_{i=1}^n f'_{i;L}(\ell'_{kkt,i}) \ge \sum_{i=1}^n f'_{i;L}(\ell_{kkt,i}) > 0.$$

This implies that no values greater than or equal to x_1 are optimal. Hence there exists an optimal value x_1^* such that $x_1^* < x_1$.

If the third case holds, then by continuity of the intervals of sub-gradients and their inverses, there exists a sequence $x_1, \ldots, x_n$ generated by the KKT optimality conditions according to Lemma 25, and the corresponding sub-gradients $f'_1(x_1), \ldots, f'_n(x_n)$, such that $x_i \in [\ell_{kkt,i}, u_{kkt,i}]$, and $\sum_{i=1}^n f'_i(x_i) = 0$. Thus the KKT optimality conditions are satisfied and we conclude that the value of x_1 is optimal. □

Lemma 28 implies a binary search algorithm to find an optimal value of x_1, x_1^*: In every step, we try a value of x_1 and compute the endpoints of the intervals $[\ell_{kkt,i}, u_{kkt,i}]$ for $i = 1, \ldots, n$ based on equations (7.19), (7.20) and (7.21). Then we compute the two quantities $\sum_{i=1}^n f'_{i;R}(u_{kkt,i})$ and $\sum_{i=1}^n f'_{i;L}(\ell_{kkt,i})$. If $\sum_{i=1}^n f'_{i;L}(\ell_{kkt,i}) \le 0 \le \sum_{i=1}^n f'_{i;R}(u_{kkt,i})$, then the current value of x_1 is optimal to MRF-PATH (7.6). Otherwise, if $\sum_{i=1}^n f'_{i;R}(u_{kkt,i}) < 0$, we check a larger value of x_1, or if $\sum_{i=1}^n f'_{i;L}(\ell_{kkt,i}) > 0$, we check a smaller value of x_1. The binary search efficiently determines an optimal value of x_1, in time complexity of $O(n \log^2 \frac{U}{\epsilon})$, where one $O(\log \frac{U}{\epsilon})$ term indicates the number of iterations in the binary search, and the other $O(\log \frac{U}{\epsilon})$ term indicates the complexity of sub-gradient inverse on separation functions. After x_1^* is found, we plug it into MRF-PATH (7.1) to reduce the problem from n variables to $n-1$ variables of the same form. In the reduced MRF-PATH problem, $f_1(x_1^*)$ is removed since it is a constant, and the deviation function of x_2 becomes $f_2(x_2) + h_1(x_1^* - x_2)$. This process is repeated in order to find an optimal solution of x_2, x_2^*. As a result, it requires n iterations to solve an optimal solution $x_1^*, \ldots, x_n^*$ for MRF-PATH (7.6). In the ith iteration,

x_i^* is solved and the problem MRF-PATH (7.1) is reduced to a smaller problem, of the same form, with one less variable.

We first present a subroutine that solves an optimal value of x_i on the reduced MRF-PATH problem of $n-i+1$ variables, $x_i, x_{i+1}, \ldots, x_n$, with deviation functions $f_i(x_i), \ldots, f_n(x_n)$, and separation functions $h_i(x_i - x_{i+1}), \ldots, h_{n-1}(x_{n-1} - x_n)$. (The values of $x_1, \ldots, x_{i-1}$ are fixed.)

$x_i^* :=$ SOLVE_REDUCED_MRF_PATH$(\{f_i, \ldots, f_n\}, \{h_i, \ldots, h_{n-1}\})$
Step 0: If $i = n$, solve $x_n^* = \text{argmin}_{x_n: -U \leq x_n \leq U} f_n(x_n)$ in ϵ-accuracy by binary search and return. Otherwise initialize the lower and upper bounds of the search region of the optimal value of x_i as $\ell = -U$ and $u = U$ respectively.
Step 1: Set $x_i = \lfloor \frac{\ell+u}{2\epsilon} \rfloor \epsilon$. Set $\ell_{kkt,i} = u_{kkt,i} = x_i$. Compute the intervals $[\ell_{kkt,j}, u_{kkt,j}]$ for $j = i, \ldots, n-1$ based on the following equations:

$$\begin{cases} u_{kkt,j+1} = u_{kkt,j} + z_{j;R} - c_j, \\ \ell_{kkt,j+1} = \ell_{kkt,j} + z_{j;L} - c_j, \end{cases} \qquad j = i, \ldots, n-1$$

where

$$z_{j;R} = \begin{cases} (\partial h_{j+1,j})_R^{-1}\big(\sum_{p=i}^{j} f'_{p;R}(u_{kkt,p})\big), & \text{if } \sum_{p=i}^{j} f'_{p;R}(u_{kkt,p}) \geq 0 \\ \\ -(\partial h_{j,j+1})_L^{-1}\big(-\sum_{p=i}^{j} f'_{p;R}(u_{kkt,p})\big), & \text{if } \sum_{p=i}^{j} f'_{p;R}(u_{kkt,p}) < 0, \end{cases}$$

$$z_{j;L} = \begin{cases} (\partial h_{j+1,j})_L^{-1}\big(\sum_{p=i}^{j} f'_{p;L}(\ell_{kkt,p})\big), & \text{if } \sum_{p=i}^{j} f'_{p;L}(\ell_{kkt,p}) > 0 \\ \\ -(\partial h_{j,j+1})_R^{-1}\big(-\sum_{p=i}^{j} f'_{p;L}(\ell_{kkt,p})\big), & \text{if } \sum_{p=i}^{j} f'_{p;L}(\ell_{kkt,p}) \leq 0. \end{cases}$$

Step 2: If $u - \ell < \epsilon$ or $\sum_{j=i}^{n} f'_{j;L}(\ell_{kkt,j}) \leq 0 \leq \sum_{j=i}^{n} f'_{j;R}(u_{kkt,j})$, return $x_i^* = x_i$ and stop.
Step 3: If $\sum_{j=i}^{n} f'_{j;R}(u_{kkt,j}) < 0$, set $\ell = x_i + \epsilon$; otherwise set $u = x_i$. Go to **Step 1**.

With the subroutine SOLVE_REDUCED_MRF_PATH, the complete algorithm to solve MRF-PATH is as follows:

Algorithm for MRF-PATH of general convex deviation and separation functions:
Step 0: Solve $c_i = \text{argmin}_{-2U \leq z_i \leq 2U} h_i(z_i)$ for $i = 1, \ldots, n-1$. Set $i = 1$.
Step 1: $x_i^* :=$ SOLVE_REDUCED_MRF_PATH$(\{f_i, f_{i+1}, \ldots, f_n\}, \{h_i, h_{i+1}, \ldots, h_{n-1}\})$.
Step 2: Set $f_{i+1}(x_{i+1}) := f_{i+1}(x_{i+1}) + h_i(x_i^* - x_{i+1})$. $i := i + 1$.
Step 3: If $i \leq n$, go to **Step 1**. Otherwise, return $(x_1^*, x_2^*, \ldots, x_n^*)$ and stop.

The complexity of subroutine SOLVE_REDUCED_MRF_PATH is $O(n \log^2 \frac{U}{\epsilon})$, hence the total complexity of the complete algorithm is $O(n^2 \log^2 \frac{U}{\epsilon})$. Therefore,

Theorem 29. *The MRF-PATH problem of arbitrary convex deviation and separation functions is solved in $O(n^2 \log^2 \frac{U}{\epsilon})$ time.*

Note that if the separation functions h_i are convex quadratic or piecewise linear with constant number of pieces, including ℓ_1-norms, so are the functions $h_{i,i+1}$ and $h_{i+1,i}$ correspondingly, the the complexity of sub-gradient inverse is $O(1)$. As a result, the (convex/convex)-algorithm can save a $O(\log \frac{U}{\epsilon})$ factor, with complexity $O(n^2 \log \frac{U}{\epsilon})$.

The (convex/convex)-algorithm can also be applied to solve MRF-PATH where each variable x_i has a explicit box constraint $\ell_i \leq x_i \leq u_i$. This is because one can remove these box constraints, without loss of generality, by extending the deviation and separation functions while maintaining convexity, as

$$\tilde{f}_i(x_i) = \begin{cases} f_i(\ell_i) - M(x_i - \ell_i), & \text{for } x_i < \ell_i \\ f_i(x_i), & \text{for } \ell_i \leq x_i \leq u_i \\ f_i(u_i) + M(x_i - u_i), & \text{for } x_i > u_i, \end{cases}$$

$$\tilde{h}_i(z_i) = \begin{cases} h_i(\ell_i - u_{i+1}) - M(z_i - (\ell_i - u_{i+1})), & \text{for } z_i < \ell_i - u_{i+1} \\ h_i(z_i), & \text{for } u_i - \ell_{i+1} \leq z_i \leq \ell_i - u_{i+1} \\ h_i(u_i - \ell_{i+1}) + M(z_i - (u_i - \ell_{i+1})), & \text{for } z_i > u_i - \ell_{i+1}, \end{cases}$$

with a sufficiently large M coefficient. We choose M such that any solution that violates the box constraints cannot be an optimal solution. Any value of $M > (L_f - L_b)/\epsilon$ will suffice, where L_f denotes a feasible objective function value and L_b denotes a lower bound on the objective function value. L_f is easy to determine. To determine L_b, we can solve:

$$\begin{aligned} \min_{\{x_i, z_i\}_i} \quad & \sum_{i=1}^{n} f_i(x_i) + \sum_{i=1}^{n-1} h_i(z_i) \\ s.t. \quad & \ell_i \leq x_i \leq u_i, \; i = 1, \ldots, n \\ & \ell_i - u_{i+1} \leq z_i \leq u_1 - \ell_{i+1}, \; i = 1, \ldots, n-1. \end{aligned}$$

This problem can be solved in $O(n \log \frac{U}{\epsilon})$ time by solving each function separately. This complexity is not dominating in the (convex/convex)-algorithm.

7.4 Concluding Remarks

In this chapter we present two efficient algorithms to solve the MRF-PATH problem (7.1) of different forms of deviation and separation functions. The algorithms directly solve the KKT optimality conditions of the MRF-PATH problem. For differentiable convex deviation functions and strictly convex separation functions, we present an algorithm that solves MRF-PATH in time $O(n \log^2 \frac{U}{\epsilon})$. For general convex deviation and separation functions, we present an algorithm that solves MRF-PATH in time $O(n^2 \log^2 \frac{U}{\epsilon})$. Our algorithms have faster running times than the existing best algorithms for MRF-PATH and some of its special cases.

Chapter 8

Evaluating Performance of Image Segmentation Criteria and Techniques

8.1 Introduction

Image segmentation is fundamental in computer vision [93]. It is used in numerous applications, such as in medical imaging [82, 30, 57, 89], and is also of independent interest in clustering [26, 80, 111, 95, 112, 105]. The image segmentation problem is to delineate, or segment, a salient feature in an image. As such, this is a bipartition problem with the goal of separating the foreground from the background. It is not obvious how to construct a quantitative measure for optimizing the quality of a segmentation. The common belief is that normalized cut (NC) criterion [95] (see mathematical formulation (8.1)) is a good model for achieving high quality image segmentation and it is often used.

The normalized cut criterion uses similarity weights that quantify the similarity between pairs of pixels. These weights are typically set to be a function of the difference between the color intensities of the pixels. Such functions are increasing with the perceived similarity between the pixels. Even though the use of normalized cut is common, it is an NP-hard problem [95] and heuristics and approximation algorithms have been employed [95, 112, 31, 105, 32]. The most frequently used method for obtaining an approximate solution for the normalized cut problem is the spectral method that finds the Fiedler eigenvector [95].

In [49] Hochbaum presented a new relaxation of the normalized cut problem, called the Hochbaum normalized cut (HNC) problem (see mathematical formulation (8.3)). The HNC problem was shown in [49] to be solved in polynomial time with a combinatorial (flow-based) algorithm. In addition, Hochbaum introduces in [49, 43] a generalization of normalized cut, called the q-normalized cut problem (q-NC, see mathematical formulation (8.2)). For the q-normalized cut problem, there are, in addition to the similarity weights, also pixel weights. The pixel weights could be a function of some pixel's feature other than color intensity. The combinatorial algorithm that solves the HNC problem was shown to generalize, with the same complexity, to a respective relaxation problem q-Hochbaum normalized cut (q-HNC, see mathematical formulation (8.4)), [49, 43]. It is also shown in [43] that the spectral method heuristic for the normalized cut problem extends to a respective heuristic for q-normalized cut.

Unlike the combinatorial algorithm's solution, the spectral method's solution is a real eigenvector, rather than a discrete bipartition. In order to generate a bipartition, a method, called the *threshold* technique, is commonly used. For a given *threshold* value, all pixels that correspond to entries of the eigenvector that exceed this threshold are set in one side of the bipartition, and the remaining pixels constitute the complement set. For further improvement, the *spectral sweep technique* selects, among all possible thresholds, the one that gives a smallest objective value for the respective normalized cut objective. A different technique, utilized by Shi et al. [113, 28], generates a bipartition from the Fiedler eigenvector which is claimed to give a superior approximation to the objective value of the respective normalized cut problem. This different method will be referred to as *Shi's code* in the remainder of the chapter. Our experimental study implements both the spectral sweep technique and Shi's code for the spectral method.

In this chapter we provide a detailed experimental study comparing the combinatorial algorithm to the spectral method, in terms of approximating the optimal value of both the normalized cut and q-normalized cut criteria; quality of visual segmentation; and in terms of running times in practice.

To compare the approximation quality, we evaluate the objective functions of the normalized cut and q-normalized cut problems for the solutions resulting from solving the HNC problem and the spectral method. These solutions are bipartitions, and hence feasible solutions for the normalized cut and q-normalized cut problems.

To evaluate visual quality, we view the feature(s) that are delineated by the bipartition solutions. The evaluation is inevitably subjective. The manner in which we evaluate the visual quality is explained in detail in Section §8.7.

For running time comparisons, we test the methods not only for the benchmark images given in 160×160 resolution, but also for higher image resolutions.

The main findings of the experimental study presented here are:

1. The combinatorial algorithm solution is a better approximation of the optimal objective value of the normalized cut problem than the solution provided by the spectral method. This dominance of the combinatorial algorithm holds for both the spectral sweep technique and Shi's code's. This is discussed in Section §8.6.

2. The discretizing technique used in Shi's code to generate a bipartition from the eigenvector is shown here to give results inferior to those of the spectral sweep technique, in terms of approximating the objective value of the respective normalized cut problem. This is displayed in Section §8.6.

3. The visual quality of the segmentation provided by the combinatorial algorithm is far superior to that of the spectral method solutions, as presented in Section §8.7.

4. Shi's code includes a variant that uses similarity weights derived with *intervening contour* [64, 71]. The visual quality resulting from segmentation with the intervening contour code is much better than the other spectral segmentations. Yet, the combinatorial algorithm with standard similarity (exponential similarity) weights delivers better visual results than Shi's code with intervening contour (Section §8.7). The combinatorial algorithm does not work well with intervening contour similarity weights since these weights tend to be of uniform value. A detailed discussion of this phenomenon is provided in Section §8.6.

5. Our study compares the visual quality of segmentations resulting from the q-HNC criterion with those resulting from the HNC criterion in Section §8.7. (We use *entropy* for pixel weights in the q-HNC instances.) The results show that q-HNC often provides better visual segmentation than HNC. Therefore for applications such as medical imaging, where each pixel is associated with multiple features, these features can be used to generate characteristic node weights, and q-HNC would be a better criterion than HNC.

6. Over the benchmark images of size 160×160, the combinatorial algorithm runs faster than the spectral method by an average speedup factor, for the normalized cut objective, of 84. Furthermore, the combinatorial algorithm scales much better than the spectral method: The speedup ratio provided by the combinatorial algorithm compared to the spectral method grows substantially with the size of the image, increasing from a factor of 84 for images of size 160×160 to a factor of 5628 for images of size 660×660.

7. For HNC we get a collection of nested bipartitions as a bi-product of the combinatorial algorithm [49, 43]. The best visual bipartition and the best normalized cut objective value bipartition are chosen among these nested bipartitions. Our study results show that in most cases the best visual bipartition does not coincide with the bipartition that gives the best objective value of the normalized cut (or q-normalized cut) problem. (The details are discussed in Section §8.7.) Therefore normalized cut, in spite of its popularity, is not a good segmentation criterion. HNC improves on normalized cut not only in complexity (from NP-hard to polynomial time solvable problem) but also in segmentation quality delivered.

The chapter is organized as follows: Section §8.2 presents the notations employed. The detailed settings of the experiment are discussed in Section §8.3. In Section §8.4, we first evaluate the effect of different selection of seeds–an important component of the combinatorial algorithm–in approximating the optimal objective values of the normalized cut and q-normalized cut problems. Then in Section §8.5, we evaluate and compare the running times of the spectral method and the combinatorial algorithm. In Section §8.6 we present the quantitative evaluation of the spectral method and the combinatorial algorithm, in terms of the quality of approximation to the optimal objective values of the normalized cut and q-normalized cut problems. Section §8.7 provides a comparison of the visual results for the spectral method versus the combinatorial algorithm. Section §8.8 includes concluding remarks.

8.2 Notations and Problem Definitions

In image segmentation an image is formalized as an undirected weighted graph $G = (V, E)$. Each pixel in the image is represented as a node in the graph. A pair of pixels are said to be *neighbors* if they are adjacent to each other. The common neighborhoods used in image segmentation are the *4-neighbor* and *8-neighbor* relations. In the 4-neighbor relation, a pixel is a neighbor of the two vertically adjacent pixels and two horizontally adjacent pixels. The 8-neighbor relation adds also the four diagonally adjacent pixels. Every pair of neighbors $i, j \in V$ is associated with an edge $[i, j] \in E$. Each edge $[i, j] \in E$ has a weight $w_{ij} \geq 0$ representing the similarity between pixel node i and j. We adopt the common notations that $n = |V|$ and $m = |E|$.

For two subsets $V_1, V_2 \subseteq V$, we define $C(V_1, V_2) = \sum_{[i,j]\in E, i\in V_1, j\in V_2} w_{ij}$. A bipartition of a graph is called a *cut*, $(S, \bar{S}) = \{[i, j] \in E | i \in S, j \in \bar{S}\}$, where $\bar{S} = V \setminus S$ is the complement of set S. The *cut capacity* is $C(S, \bar{S})$. Each node has a weight $d(i) = \sum_{[i,j]\in E} w_{ij}$ which is the sum of the weights of its incident edges. For a set of nodes S, $d(S) = \sum_{i\in S} d(i)$. A node may have also an arbitrary nonnegative weight associated with it, $q(i)$. For a set of nodes $S \subseteq V$, $q(S) = \sum_{i\in S} q(i)$.

Let $\mathbf{D}$ be a diagonal $n \times n$ matrix with $\mathbf{D}_{ii} = d(i) = \sum_{[i,j]\in E} w_{ij}$. Let $\mathbf{W}$ be the weighted node-node adjacency matrix of the graph where $\mathbf{W}_{ij} = \mathbf{W}_{ji} = w_{ij}$. The matrix $\mathcal{L} = \mathbf{D} - \mathbf{W}$ is called the Laplacian of the graph.

The mathematical formulations of the normalized cut and q-normalized cut problems are:

Normalized cut (NC), [95]

$$NC_G = \min_{\emptyset\subset S\subset V} \frac{C(S, \bar{S})}{d(S)} + \frac{C(S, \bar{S})}{d(\bar{S})}, \tag{8.1}$$

q-Normalized cut (q-NC), [43]

$$q\text{-}NC_G = \min_{\emptyset\subset S\subset V} \frac{C(S, \bar{S})}{q(S)} + \frac{C(S, \bar{S})}{q(\bar{S})}. \tag{8.2}$$

A relaxation of these problems, introduced by [49, 43], omits the second term in the objective value. We call these relaxations HNC and q-HNC problems respectively:

HNC, [49]

$$HNC_G = \min_{\emptyset\subset S\subset V} \frac{C(S, \bar{S})}{d(S)}, \tag{8.3}$$

q-HNC, [49, 43]

$$q\text{-}HNC_G = \min_{\emptyset \subset S \subset V} \frac{C(S,\bar{S})}{q(S)}. \tag{8.4}$$

It is shown in [49, 43] that

$$\min_{\emptyset \subset S \subset V} \frac{C(S,\bar{S})}{C(S,S)} \tag{8.5}$$

defined in [94] is equivalent to (8.3) in that both have the same optimal solution. Problem (8.5) is a criterion characterizing a good image bipartition by two goals. One requires the salient region segmented to be dissimilar from the rest of the image, or formally to have a small value for $C(S,\bar{S})$. The second goal is to have the pixels in the segmented region as similar to each other as possible. This second goal is to have a large value for $C(S,S)$.

The normalized cut and q-normalized cut problems are NP-hard [95, 43]. The combinatorial algorithm presented in [49] solves the normalized cut and q-normalized cut problems approximately by solving their relaxations, HNC and q-HNC problems respectively. Both the HNC and q-HNC problems are polynomial time solvable by the combinatorial algorithm.

A Bound on the Relation Between the Spectral Method Solution and NC_G

The bounds on the Fiedler eigenvalue were developed for a problem closely associated with normalized cut. This problem, devised by Cheeger in [23] and called the *Cheeger constant* problem (e.g. [25]), is a "half-version" of normalized cut: If the balance constraint $d(S) \le d(V)/2$ is added, the formulation of the Cheeger constant problem is

$$h_G = \min_{\emptyset \subset S \subset V, d(S) \le d(V)/2} \frac{C(S,\bar{S})}{d(S)}. \tag{8.6}$$

h_G is called the Cheeger constant of the graph G. Like the normalized cut problem, the Cheeger constant problem is also NP-hard (e.g. [25]). For any (undirected) graph G, the Cheeger inequality states that its Cheeger constant h_G is bounded by the second smallest eigenvalue of the (normalized) Laplacian of G, the Fiedler eigenvalue λ_1: $\frac{\lambda_1}{2} \le h_G \le \sqrt{2\lambda_1}$ [23, 25]. In addition, the solution resulting from applying the spectral sweep technique to the Fiedler eigenvector evaluated by the objective of the Cheeger constant problem, is of value at most $2\sqrt{h_G}$ [24]. On the other hand, it is easily shown that h_G and NC_G satisfy: $\frac{1}{2}NC_G \le h_G \le NC_G$ (e.g. [43]). These (approximation) bounds for the Cheeger constant (with respect to λ_1 and the sweep solution) can be easily transformed to corresponding bounds for the normalized cut objective. Therefore, the spectral method approximately solves the normalized cut (and the Cheeger constant and q-normalized cut) problem by finding the Fiedler eigenvector [23, 33, 38, 7, 6, 95, 43].

8.3 Experimental Setting

Edge and Node Weights

Similarity Edge Weights

The benchmark images used here consist of grayscale images. A color intensity value is associated with every pixel, represented as an integer in $[0, 255]$ in MATLAB. This is normalized and mapped to $[0, 1]$. The similarity weight between a pair of pixel nodes is a function of the difference of their color intensities. For p_i and p_j the color intensities of two neighboring pixel nodes i and j, the *exponential similarity weight* is defined as

$$w_{ij} = e^{-\alpha|p_i - p_j|}, \tag{8.7}$$

where α can be viewed as amplifying the dissimilarity between two pixels based on the color intensity difference. If α is too small, then the dissimilarity is not significant enough to reflect the color intensity difference. On the other hand, setting the value of α to be too large results in all pairs of pixels very dissimilar and therefore color intensity differences are not sufficiently informative. We tested several settings for α values and found $\alpha = 100$ works well. In all experiments prepared here α is set to 100.

Another similarity weight is *intervening contour* introduced in [64, 71]. Intervening contour uses the contour information in an image to characterize the (local) similarity between two pixels that are not necessarily neighboring. If two pixels are on the two different sides of a boundary, their similarity should be small as they are more likely to belong to different segments. In the experiment, we use the intervening contour similarity weight generated by Shi's code. Since Shi's code with intervening contour is considered to generate good segmentation, therefore we compare it to the combinatorial algorithm.

Node Weights

For q-normalized cut, the *entropy* of a pixel is used as its weight. The entropy of an image is a measure of randomness in the image that can be used to characterize the texture of an image. In MATLAB, by default the local entropy value of a pixel is the entropy value of the 9-by-9 neighborhood around the pixel. In our experiment, the entropy of a pixel is computed directly via the MATLAB built-in function `entropyfilt`.

Images Database

We select twenty benchmark images from the Berkeley Segmentation Dataset and Benchmark (http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/). See Figure C.1 in the appendix. The twenty benchmark images are chosen to cover various segmentation difficulties and have been resized to be 160×160 for testing since it is the default size in Shi's code.

Implementation of the Combinatorial Algorithm

Seed Selection

The combinatorial algorithm requires to designate a node as a seed in one set and a node as a seed in the other set to guarantee that both sets are nonempty [49]. On the other hand, the delineation of foreground versus background depends on the interpretation of what is the main feature. This is not self evident and the purpose of the seeds is to have one seed indicating a pixel in the foreground and the other seed indicating a pixel in the background.

Theoretically, in order to obtain the optimal solutions to the HNC and q-HNC problems, all possible pairs of seed nodes should be considered. This increases the complexity of the combinatorial algorithm by a factor of $O(n)$. To avoid this added complexity we devise a test for automatically choosing the seed nodes.

Two automatic seed selection rules are introduced here. The first rule is to select the pixel with the maximum entropy as a seed node in one set. The second rule is to select the pixel with the maximum *group luminance* value as a seed node in one set. The group luminance value is defined for pixels not on the boundary. For every pixel i, the group luminance value of pixel i is the average of color intensities of the nine neighboring pixels in the 3×3 region centering at i. Intuitively, if a pixel has a greater group luminance value, that pixel and its surrounding pixels are more likely to be in the same segment. The other seed node is any arbitrarily selected node in the complement region to the one occupied by the first seed node. We compare the two automatic seed selection methods with a manual selection of both seed nodes.

Method #	One Seed Node
1	Manual
2	Max Entropy
3	Max Group Luminance

Table 8.1: Three seed selection rules.

For each pair of seed nodes, the combinatorial algorithm is run twice where in the second run the two seed nodes are exchanged between the two sets. Therefore for each image the combinatorial algorithm is executed six times, for the three different seed selection rules.

Nested Cuts

Each run of the combinatorial algorithm for a pair of seed nodes, to either the HNC or the q-HNC problem, produces a series of nested cuts. This is because the combinatorial algorithm uses a parametric minimum cut solver as a subroutine [49, 43]. The parametric minimum cut problem can be solved efficiently. Theoretically, it is shown in [39, 50] that the running time to solve the parametric minimum cut problem is only a small constant factor of the time to solve a single instance of the minimum cut problem. We implement Hochbaum's pseudoflow algorithm in [50] as the parametric minimum cut solver. The implementation is described in [21] and the code is available online at [22].

The number of the nested cuts is typically 5 to 15. The combinatorial algorithm stores the visual segmentations by all the nested cuts, which enables to choose the one that is deemed (subjectively) best visually. The combinatorial algorithm also automatically selects the bipartition which gives the smallest objective values of the normalized cut or q-normalized cut problem among the nested cuts.

Implementation of the Spectral Method

The eigenvector solver subroutine of Shi's code, named eigs2, is based on the MATLAB built-in eigenvector solver eigs with some modifications. Shi's code applies the following two operations to the Laplacian matrix $\mathcal{L} = \mathbf{D} - \mathbf{W}$:

1. Sparsifying operation: It rounds to 0 small values of w_{ij} where the "small" is determined by some threshold value. The default threshold value in Shi's code is 10^{-6}.

2. Offset operation: It adds a constant (1 by default) to $\mathbf{D}_{ii} = d(i)$ $(i = 1, \dots, n)$. It also adds a value to each diagonal entry of the $\mathbf{W}$ matrix. This value for entry $\mathbf{W}_{ii}$ is 0.5 plus a quantity that estimates the round-off error for row i, $e(i) = d(i) - \sum_{j=1}^{n} w_{ij}$.

In our experiment, we exclude the above two operations in Shi's code in order to compare it with the combinatorial algorithm, which contains no any sparsifying or offset operations.

The spectral sweep technique uses the Fiedler eigenvector from Shi's code (without the two operations) and then chooses the best bipartition threshold as described in Section §8.1.

Algorithm, Optimization Criterion, and Similarity Classifications and Nomenclatures

Each experimental set is characterized by the choice of algorithm, the choice of optimization objective, and the choice of similarity weight definition. For the algorithm, we choose among the combinatorial algorithm, *COMB*, Shi's code, *SHI* and the spectral sweep technique, *SWEEP*. For the optimization objective, we choose among normalized cut, *NC*, and q-normalized cut, *qNC*. For the similarity weight definition, we choose among the exponential similarity weights, *EXP*, and the intervening contour similarity weights, *IC*. The format of *Algorithm-Criterion-Similarity* is used to represent an experimental set.

For each choice of optimization objective and similarity weight definition, the combinatorial algorithm outputs a series of nested cuts for a pair of seeds (see Section §8.3), among which the cut that gives the smallest *objective value* of NC or q-NC is selected. The pairs of seeds are selected according to the automatic seed selection criterion, including both the maximum entropy criterion and the maximum group luminance criterion, described in Section §8.3. The numerically best cut is selected among the four series of corresponding nested cuts. The segmentation of the selected cut is considered as the output of the combinatorial algorithm and the objective value of NC or q-NC of the selected cut is considered as the objective value output by the combinatorial algorithm.

We test the following experimental sets:
COMB-NC-EXP
COMB-qNC-EXP
COMB-NC-IC
COMB-qNC-IC
SHI-NC-EXP
SHI-qNC-EXP
SHI-NC-IC
SHI-qNC-IC
SWEEP-NC-EXP
SWEEP-qNC-EXP
SWEEP-NC-IC
SWEEP-qNC-IC.

8.4 Assessing Quality of Seed Selection Methods of the Combinatorial Algorithm

In this section we evaluate the three seed selection methods introduced in Section §8.3 for the combinatorial algorithm in terms of approximating the objective values of NC and q-NC. We evaluate the objective values of NC and q-NC for each of the solutions provided by the three seed selection methods and count the number of images on which each seed selection method gives the smallest objective values of NC and q-NC. Table 8.2 gives the percentage of the number of images, out of the twenty images, in which each method gets the best values for NC and q-NC respectively. The exponential similarity weight is used in both cases.

	Method 1 (manual)	Method 2 (max-entropy)	Method 3 (max-group luminance)
NC	33.33%	37.04%	29.63%
q-NC	26.09%	47.83%	26.09%

Table 8.2: Portions of each seed selection method in yielding the smallest NC and q-NC objective values.

The results given in Table 8.2 show that method 2 (max entropy) is best for NC and q-NC. This indicates that the maximum entropy is a good seed selection method for image segmentation.

Since method 3 (max group luminance) is automatic, and also works well, we derive an automatic seed selection method which combines method 2 and method 3. This is done by running the combinatorial algorithm for the pairs of seeds generated by method 2 and method 3, and the output is the one corresponds to the best of these four values. The automatic seed selection method is best 66.67% of the time for NC and 73.92% of the time for q-NC. This improves a great deal on method 1, where the two seeds are selected manually.

As a result of the comparison, in the following comparisons the cut that gives the smallest objective values of NC or q-NC of COMB is selected from the four series of nested cuts corresponding to the four pairs of seeds selected according to the automatic seed selection method defined above.

8.5 Running Time Comparison Between the Spectral Method and the Combinatorial Algorithm

In order to provide a fair comparison of the running times of the two algorithms, we disregard the input and output processing parts of each algorithm. We run both algorithms on a 2010 Apple MacBook Pro Laptop (2.4GHz Intel Core 2 Duo processor and 4GB of 1067 MHz DDR3). The running times of the two algorithms over all the twenty benchmark images with size 160×160 are reported in Table 8.3. The exponential similarity weights are used and both NC and q-NC objectives are applied.

Table 8.3 shows that the combinatorial algorithm runs much faster than the spectral method for the NC objective by an average speedup factor of 84. For the q-NC objective, in most cases the combinatorial algorithm is still faster. The same comparison results also apply to the case of intervening contour similarity weights. It is not clear why the spectral method runs so much faster for q-NC than NC. We note, however, that the results delivered by the spectral method for q-NC are dramatically inferior to those provided by the combinatorial algorithm, both in terms of approximating the optimal objective value of q-NC (Figure 8.2 and 8.4), and in terms of visual quality (Section §8.7).

We further evaluate the scalability of the two algorithms by creating input sequences of images each based on one image at increasing resolutions: We employ six different image sizes: 160×160, 260×260, 360×360, 460×460, 560×560 and 660×660. For every image size we run the two algorithms on five of the twenty benchmark images (Image 4, 8, 12, 16 and 20) and average the running times over the five images. The running times of the spectral method and the combinatorial algorithm with these different image sizes are plotted in Figure 8.1.

Figure 8.1 shows that as the input size increases, the running time of the spectral method grows significantly faster than that of the combinatorial algorithm, with an average speedup factor increasing from 84 for images of size 160×160 to 5628 for images of size 660×660. The running time of the combinatorial algorithm appears insensitive to changes in the input size. Interestingly, we observe that the running time of the combinatorial algorithm does not

Time(s)	SHI/SWEEP -NC-EXP	COMB -NC-EXP	SHI/SWEEP -qNC-EXP	COMB -qNC-EXP
Image1	19.5905	0.468	0.67045	0.595
Image2	20.5193	0.029	0.56141	0.047
Image3	19.9446	0.090	0.34875	0.208
Image4	19.0059	0.186	0.79889	0.287
Image5	20.4605	0.526	0.32494	0.810
Image6	19.2924	0.138	0.95232	0.543
Image7	20.4066	0.214	0.65428	0.505
Image8	17.6540	0.390	0.82868	0.440
Image9	17.5387	0.150	0.36554	0.241
Image10	17.3702	0.119	0.37487	0.206
Image11	19.6832	0.024	0.34565	0.162
Image12	17.7123	0.226	0.39409	0.468
Image13	17.3662	0.034	0.58339	0.046
Image14	17.5793	0.456	0.54604	0.615
Image15	18.9113	0.376	0.35412	0.617
Image16	19.9957	0.125	0.78376	0.329
Image17	19.6383	0.068	0.53126	0.073
Image18	17.5165	0.263	0.56119	0.430
Image19	20.3009	0.455	0.54215	0.575
Image20	22.3611	0.227	0.54404	0.267

Table 8.3: Running times of SHI/SWEEP-NC-EXP, COMB-NC-EXP, SHI/SWEEP-qNC-EXP and COMB-qNC-EXP.

increase with the size of the image. This is because for these images in higher resolutions, the number of breakpoints is smaller and therefore there are fewer updates required between consecutive breakpoints [43].

8.6 Quantitative Evaluation for Objective Function Values

In this section, we compare the performance of the spectral method and the combinatorial algorithm in terms of how well they approximate the optimal objective values of the normalized cut and q-normalized cut problems. In Section §8.6 we compare SHI with COMB and in Section §8.6 we compare SWEEP with COMB. Both exponential similarity weights and intervening contour weights are used in the comparisons.

In order to compare the performance of the spectral method, either SHI or SWEEP,

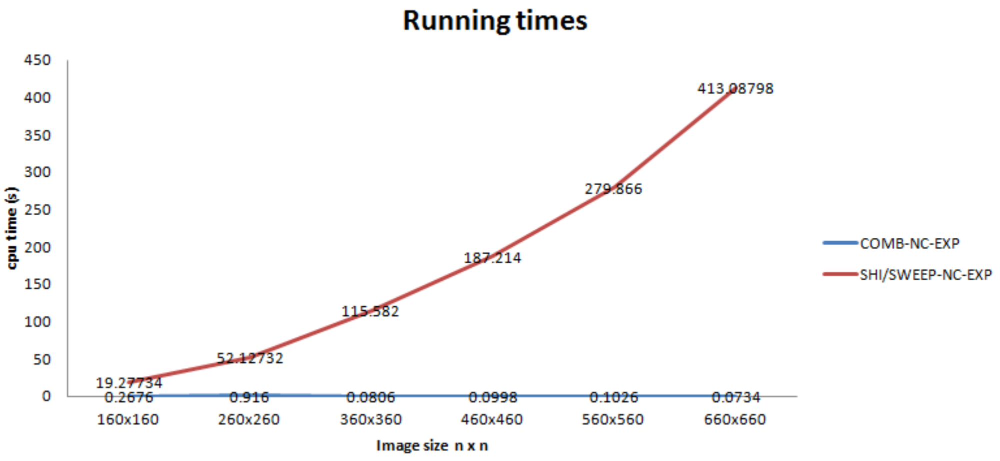


Figure 8.1: Running times of SHI/SWEEP-NC-EXP and COMB-NC-EXP for images with increasing resolutions.

with COMB in approximating the optimal objective value of NC or q-NC, we compute a ratio of the objective value of NC or q-NC generated by the spectral method to the corresponding objective value generated by COMB. If the ratio is greater than 1, it indicates that COMB performs better than the spectral method, while the ratio smaller than 1 is indicative of the spectral method having better performance. If the ratio is smaller than 1, its reciprocal characterizes the improvement of the spectral method on COMB in approximating the optimal objective value of NC or q-NC.

Comparing Approximation Quality of SHI and COMB

Comparing Instances with Exponential Similarity Weights: Comparing SHI-NC-EXP with COMB-NC-EXP and SHI-qNC-EXP with COMB-qNC-EXP

Table 8.4 and Table 8.5 show the ratios over the twenty benchmark images using exponential similarity weights. Table 8.4 and Table 8.5 present the ratios with respect to NC and q-NC respectively, for SHI versus COMB.

As seen in Table 8.4, for every case COMB yields smaller NC objective values than SHI, with the mean improvement factor exceeding 2.3×10^6. For q-NC the results in Table 8.5 show that COMB not only yields smaller objective values than SHI, but these improvements are dramatically larger than those for NC, with a mean improvement factor exceeding 9.2×10^{11}. We conclude that the relative performance of SHI is non-competitive for NC and worse still for q-NC. Figure 8.2 is a bar chart for the ratios in Table 8.4 and Table 8.5. (Note that the bar chart ratios are given in log scale, and same for the rest bar charts.) Table 8.6 displays the means and medians of the ratios in Table 8.4 and Table 8.5. They demonstrate the

Image 1 10.034375	Image 2 34.945958	Image 3 228.88261	Image 4 1.0776067	Image 5 522467.35
Image 6 45242414	Image 7 757898.1	Image 8 800.08425	Image 9 7.1908952	Image 10 357.12512
Image 11 11514.768	Image 12 125.05640	Image 13 4.8974465	Image 14 1340.9285	Image 15 233.03002
Image 16 11.050608	Image 17 16.897142	Image 18 345.39787	Image 19 471.05938	Image 20 6.5424435

Table 8.4: The ratios of the NC objective values of SHI-NC-EXP to COMB-NC-EXP.

Image 1 257689.45	Image 2 3599904.9	Image 3 654763.23	Image 4 5261971.8	Image 5 1418996100
Image 6 1.8295880×10^{13}	Image 7 1.6418361×10^{11}	Image 8 92852128	Image 9 10836.986	Image 10 63071852
Image 11 6835417700	Image 12 5388176	Image 13 431894.04	Image 14 3397368.9	Image 15 2755701700
Image 16 13524963	Image 17 1440666.1	Image 18 14317766	Image 19 5203545.6	Image 20 681058.90

Table 8.5: The ratios of the q-NC objective values of SHI-qNC-EXP to COMB-qNC-EXP.

extent of the improvement of COMB over SHI.

	Mean of Improvements	Median of Improvements
NC	2326914.4	230.95632
q-NC	9.2356417×10^{11}	5325073.9

Table 8.6: The means and medians of the improvements of COMB-NC-EXP on SHI-NC-EXP and COMB-qNC-EXP on SHI-qNC-EXP.

Comparing Instances with Intervening Contour Similarity Weights: Comparing SHI-NC-IC with COMB-NC-IC and SHI-qNC-IC with COMB-qNC-IC

For the intervening contour similarity weights, we observe that the graph has nodes of (roughly) equal degrees. The edge weights are almost the same value close to 1. Hence the cut separating a small set of nodes consists of fewer edges, and thus smaller capacity, than those cuts that separate sets of roughly equal sizes. Consequently, COMB favors unbalanced cuts with a small number of nodes on one side of the bipartition. This phenomenon is illustrated for Image 8 in Table 8.7.

The bipartition $(S8, \bar{S}8)$ obtained by COMB-NC-EXP in Figure (8.7a) has the background pixels all black and the foreground pixels unmodified. In this particular bipartition

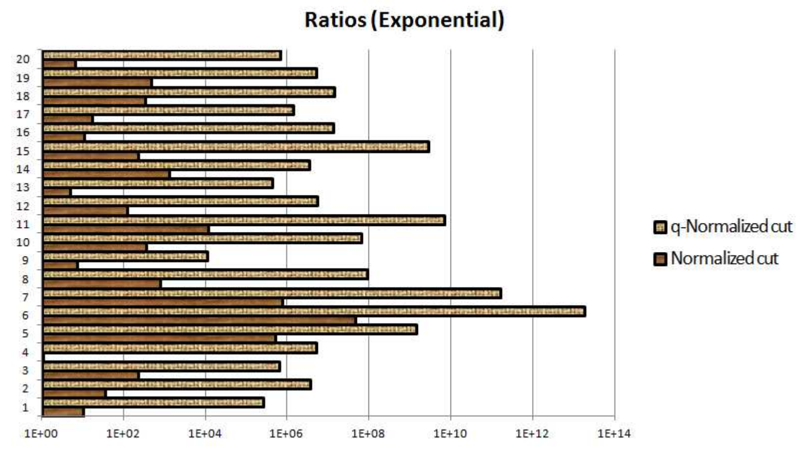


Figure 8.2: Bar chart for the ratios in Table 8.4 and Table 8.5. The darker bars represent ratios for NC (Table 8.4) and the lighter bars represent ratios for q-NC (Table 8.5).

the background is the sky. Thus the similarity weights of edges in the cut $(S8, \bar{S}8)$ should be small. We compute the capacity of cut $(S8, \bar{S}8)$, $C(S8, \bar{S}8)$, with respect to exponential weights and intervening contour weights.

For the exponential and intervening contour similarity weights, the maximum similarity value is 1. Note however, that the average intervening contour edge weight in cut $(S8, \bar{S}8)$ is 0.67385345, which is quite close to 1. This is not the case for exponential similarity weights where the average exponential edge weight in cut $(S8, \bar{S}8)$ is 0.0000018360302. This demonstrates that intervening contour similarity weights are almost uniform and close to 1 throughout the graph.

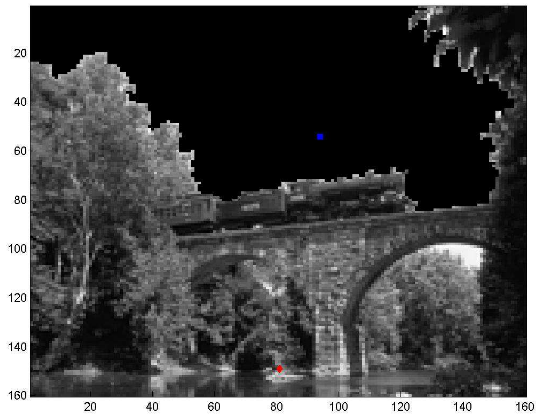
We now select a single pixel v (highlighted with the square) in the background (sky), which implies it is highly similar to its neighbors, and consider the cut $(\{v\}, V \setminus \{v\})$. For exponential similarity weights the capacity of this cut, $C(\{v\}, V \setminus \{v\})$, is 3.9985 and therefore substantially higher than the capacity of the cut $(S8, \bar{S}8)$. For intervening contour similarity weights, however, the capacity of the cut $(\{v\}, V \setminus \{v\})$ is 8, which is far smaller than the capacity of the cut $(S8, \bar{S}8)$.

Therefore intervening contour similarity weights do not work well and produce unbalanced cuts with algorithms that consider the cut capacity such as COMB.

Table 8.8 and Table 8.9 show the ratios over the twenty benchmark images using intervening contour similarity weights. Table 8.8 demonstrates the ratios with respect to NC objective values and Table 8.9 for q-NC objective values, for SHI versus COMB.

For the NC results shown in Table 8.8, there are 5 images where COMB gives better approximations while for the rest 15 images SHI performs better. In most of the cases among the 15 images, COMB just favors an unbalanced cut.

For q-NC, as can be seen from the ratios displayed in Table 8.9, COMB performs much better than SHI. There are 15 images where COMB gives better approximations to q-NC



(a) Image8-Cut $(S8, \bar{S}8)$

	$C(S8, \bar{S}8)$	Number of edges in $(S8, \bar{S}8)$	Average edge weight of $(S8, \bar{S}8)$	$C(\{v\}, V \setminus \{v\})$
EXP	0.00095473570	520	0.0000018360302	3.9985
IC	886.11728	1315	0.67385345	8

(b) The capacities of the cut $(S8, \bar{S}8)$ and $(\{v\}, V \setminus \{v\})$, and the average edge weight of edges in cut $(S8, \bar{S}8)$ with two similarity weights.

Table 8.7: Illustrating why COMB favors unbalanced cut with intervening contour similarity weights.

Image 1 0.0059191506	Image 2 3.013586×10^{-5}	Image 3 2.0904086×10^{16}	Image 4 1	Image 5 7.42769
Image 6 0.0026249997	Image 7 0.00019634185	Image 8 2.8444556	Image 9 0.0047000578	Image 10 0.014255762
Image 11 0.0053146160	Image 12 0.0086205694	Image 13 0.0071974529	Image 14 9.7893761×10^{-6}	Image 15 1.5935882
Image 16 0.0015875799	Image 17 0.0045286429	Image 18 2.0502743	Image 19 0.00040627891	Image 20 0.033866313

Table 8.8: The ratios of the NC objective values of SHI-NC-IC to COMB-NC-IC.

than SHI. Furthermore, for q-NC, the improvements of COMB on SHI are very significant. Table 8.10 shows the average improvements of COMB on SHI with respect to NC and q-NC with intervening contour similarity weights. They are the means and medians of the ratios that are greater than 1 in Table 8.8, for NC, and Table 8.9, for q-NC, respectively. We also display the average improvements of SHI on COMB for NC and q-NC with intervening contour similarity weights in Table 8.11. For all the ratios in Table 8.8 and Table 8.9 that

Image 1	Image 2	Image 3	Image 4	Image 5
426.94051	0.49236168	51.416438	1291202.2	10270.311
Image 6	Image 7	Image 8	Image 9	Image 10
44.623615	0.64894629	110933.38	1.5971419	0.68595629
Image 11	Image 12	Image 13	Image 14	Image 15
0.61131118	1.1039278	1.1031171	1.3381326	3497.6627
Image 16	Image 17	Image 18	Image 19	Image 20
11.839168	0.50931088	34866.574	165.89208	307.61301

Table 8.9: The ratios of the q-NC objective values of SHI-qNC-IC to COMB-qNC-IC.

are smaller than 1, the corresponding reciprocals characterize the improvements of SHI. We take the means and medians of these reciprocals from Table 8.8, for NC, and Table 8.9, for q-NC, respectively, to produce Table 8.11. We display the ratios of Table 8.8 and Table 8.9 in bar chart Figure 8.3. The bars extending to the right (the ratios are greater than 1) indicate the improvements of COMB on SHI while the bars extending to the left (the ratios are smaller than 1) indicate the improvements of SHI on COMB.

	Mean of Improvements	Median of Improvements
NC	4.1808173×10^{15}	2.8444556
q-NC	96785.570	165.89208

Table 8.10: The means and medians of the improvements of COMB-NC-IC on SHI-NC-IC and COMB-qNC-IC on SHI-qNC-IC.

	Mean of Improvements	Median of Improvements
NC	9669.7517	212.76334
q-NC	1.7258142	1.6358281

Table 8.11: The means and medians of the improvements of SHI-NC-IC on COMB-NC-IC and SHI-qNC-IC on COMB-qNC-IC.

Comparing Approximation Quality of SWEEP and COMB

In general SWEEP should perform better than just taking the eigenvector with some predetermined procedure for bipartition. Let NC_{SWEEP} be the normalized cut objective value generated by SWEEP. By the analysis in Section §8.2, it can be shown that $NC_{SWEEP} \leq 4\sqrt{NC_G}$. Therefore one may expect that SWEEP will improve on SHI in approximating the optimal objective value of NC. We illustrate the potential improvement of SWEEP over SHI for Image 6, where the gap between the approximation of COMB-NC-EXP and

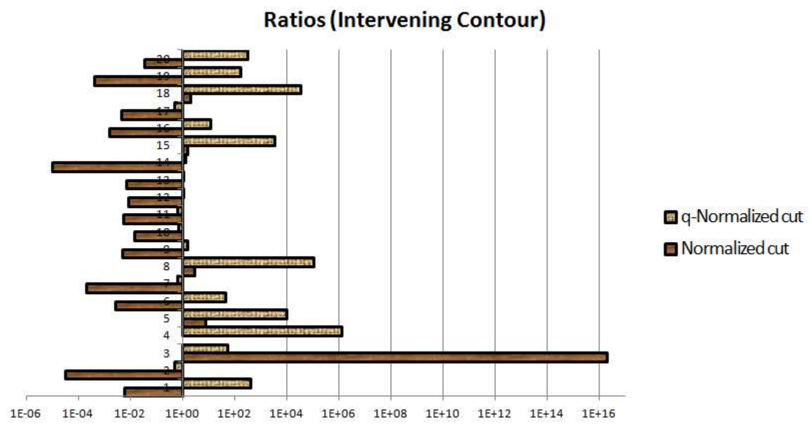


Figure 8.3: Bar chart for the ratios in Table 8.8 and Table 8.9. The darker bars represent ratios for NC (Table 8.8) and the lighter bars represent ratios for q-NC (Table 8.9).

the approximation of SHI-NC-EXP is largest among the twenty images, as shown in Table 8.4. From the original data, the approximate objective value of NC achieved by COMB is $NC_{COMB} = 3.8129621 \times 10^{-11}$. Therefore the optimal NC objective of Image 6, NC_G, is less than or equal to the value of NC_{COMB}. If we use NC_{COMB} as an estimation to NC_G, then we obtain an upper bound for NC_{SWEEP}:

$$NC_{SWEEP} \leq 4\sqrt{NC_G} \leq 4\sqrt{NC_{COMB}} = 4\sqrt{3.8129621 \times 10^{-11}} = 2.4699675 \times 10^{-5}. \quad (8.8)$$

However, our original data show that the objective value of NC achieved by SHI for Image 6 is 1.7250761×10^{-3}. Since NC_{SWEEP} can only be smaller than the upper bound 2.4699675×10^{-5}, the objective value of NC achieved by SWEEP improves by at least a factor of 70 on the objective value generated by SHI. Indeed the experimental results match the theoretical prediction that SWEEP does better than SHI for NC. But still, COMB is better than SWEEP with exponential similarity weights.

In addition to the improvement of SWEEP over SHI or fixed threshold bipartition of the Fiedler eigenvector, SWEEP can improve on COMB for intervening contour similarity weights. As discussed in Section §8.6, COMB tends to provide unbalanced bipartitions for intervening contour similarity weight matrices. For SWEEP this is not an issue, because each threshold bipartition is considered, and the best threshold will obviously correspond to a balanced bipartition. Therefore we expect SWEEP to do better than COMB for intervening contour similarity weights.

In the following we display the comparisons of approximation quality of SWEEP-NC-EXP with COMB-NC-EXP and SWEEP-qNC-EXP with COMB-qNC-EXP.

Table 8.12 and Table 8.13 show the ratios over the twenty benchmark images using exponential similarity weights for SWEEP versus COMB. Table 8.12 and Table 8.13 present the ratios with respect to NC and q-NC respectively.

Image 1 0.0066413908	Image 2 0.030815693	Image 3 1.7318891	Image 4 0.25438092	Image 5 1.021992
Image 6 32.769017	Image 7 151053.5	Image 8 1.0124601	Image 9 0.65635751	Image 10 16.970223
Image 11 3229.7642	Image 12 1.7051154	Image 13 0.59025832	Image 14 0.073415316	Image 15 13.387584
Image 16 0.025636729	Image 17 1.2725133	Image 18 1.6439209	Image 19 0.56137852	Image 20 0.22995850

Table 8.12: The ratios of the NC objective values of SWEEP-NC-EXP to COMB-NC-EXP.

Image 1 44415.346	Image 2 295255	Image 3 222060.29	Image 4 4827.1580	Image 5 26524576
Image 6 20416303000	Image 7 16686194000	Image 8 248921.64	Image 9 5306.5587	Image 10 23558163
Image 11 2097520100	Image 12 1344474.1	Image 13 80934.898	Image 14 921433.37	Image 15 25749626
Image 16 54403.132	Image 17 216141.34	Image 18 1191186.4	Image 19 2289243	Image 20 70300.882

Table 8.13: The ratios of the q-NC objective values of SWEEP-qNC-EXP to COMB-qNC-EXP.

For the NC results shown in Table 8.12, there are 11 out of the twenty benchmark images where COMB gives a better approximation than SWEEP and the improvements of COMB over SWEEP are smaller than those over SHI. For the q-NC results displayed in Table 8.13, COMB dominates SWEEP and gives better approximations in every case. The results establish that while SWEEP delivers better results than SHI, COMB is still dominant, and gives better results in most cases. We display the means and medians of the improvements of each method to the other in Table 8.14 and 8.15 respectively. Table 8.14 is for the average improvement of COMB on SWEEP while Table 8.15 is for the average improvement of SWEEP on COMB. The means and medians are obtained from the ratios in Table 8.12 and Table 8.13 for NC and q-NC respectively, using the same method introduced in Section §8.6. The ratios in Table 8.12 and Table 8.13 are displayed as a bar chart in Figure 8.4.

	Mean of Improvements	Median of Improvements
NC	14032.252	1.7318891
q-NC	1964141900	608344.19

Table 8.14: The means and medians of the improvements of COMB-NC-EXP on SWEEP-NC-EXP and COMB-qNC-EXP on SWEEP-qNC-EXP.

	Mean of Improvements	Median of Improvements
NC	27.658703	4.3486107

Table 8.15: The means and medians of the improvements of SWEEP-NC-EXP on COMB-NC-EXP. Notice that for q-NC, there is no improvement of SWEEP-qNC-EXP on COMB-qNC-EXP.

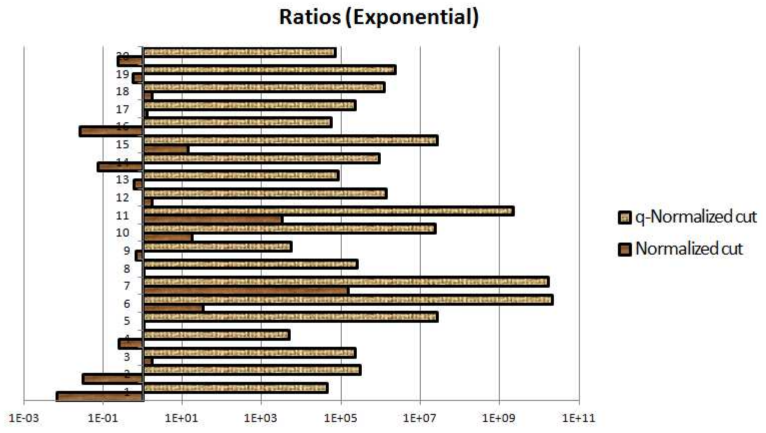


Figure 8.4: Bar chart for the ratios in Table 8.12 and Table 8.13. The darker bars represent ratios for NC (Table 8.12) and the lighter bars represent ratios for q-NC (Table 8.13).

8.7 Visual Segmentation Quality Evaluation

In this section, we first evaluate the visual segmentation quality among the three methods, COMB, SHI and SWEEP. Then we compare the criteria HNC and q-HNC to NC and q-NC respectively to see which is a better criterion to give good visual segmentation results. Since visual quality is subjective, we provide a subjective assessment, which may not agree with the readers' judgement.

In some of the comparisons made in this section, we select for COMB the cut which gives the visually best segmentation among the four series of nested cuts corresponding to the four pairs of seeds selected according to the automatic seed selection criterion, as the output of COMB. This visually best cut is often not the numerically best cut that gives the smallest value for NC or q-NC objectives. When the visually best cut is chosen as the output of COMB, we use the experimental set notation *COMB(HNC)-Similarity* or *COMB(qHNC)-Similarity*. Here the (*HNC*) or (*qHNC*) are used to denote which optimization objective that COMB actually solves, and the *-Similarity* choice can be either exponential or intervening contour similarity weights. Notice that COMB(HNC)-Similarity represents a different experimental set from COMB-NC-Similarity introduced in Section §8.3, since the former experimental set uses the visually best cut while the latter experimental set uses the numerically best cut. So are COMB(qHNC)-Similarity and COMB-qNC-Similarity.

For SHI and SWEEP, since each of them outputs a unique cut as the solution, there is no distinction between the numerically and visually best cuts. We still use the experimental set notations defined in Section §8.3 for experimental sets of SHI and SWEEP.

SHI uses a discretization method to generate a bipartition from the Fiedler eigenvector [113] which is considered to give good visual segmentations. Hence when comparing SHI with COMB, we use the visually best cut as the output of COMB. We conduct the following four comparisons between SHI and COMB:

SHI-NC-EXP and COMB(HNC)-EXP
SHI-qNC-EXP and COMB(qHNC)-EXP
SHI-NC-IC and COMB(HNC)-IC
SHI-qNC-IC and COMB(qHNC)-IC.

When comparing SWEEP with COMB, we use the numerically best cut as the output of COMB. This is because SWEEP outputs the cut that gives the smallest objective value of NC or q-NC among all potential threshold values. Hence we conduct the following two comparisons between SWEEP and COMB:

SWEEP-NC-EXP and COMB-NC-EXP
SWEEP-qNC-EXP and COMB-qNC-EXP.

We assess the visual quality of segmentations generated by COMB to compare the performance of different optimization criteria in producing visually good segmentation results. We compare the visual segmentation quality of COMB(HNC)-EXP with COMB(qHNC)-EXP to determine which criterion, HNC or q-HNC, works better visually. We then compare NC with HNC and q-NC with q-HNC by comparing the visual segmentation quality of the following two pairs of experimental sets:

COMB-NC-EXP and COMB(HNC)-EXP
COMB-qNC-EXP and COMB(qHNC)-EXP.

For each of the above comparisons of two experimental sets, namely *experimental set 1* and *experimental set 2*, we classify each of the twenty benchmark images into the following three categories:

1. Experimental set 1 gives a better visual segmentation result than experimental set 2. This is denoted as $1 \succ_{\text{v}} 2$, where the subscript "v" stands for "visual" and same for the rest.

2. Experimental set 2 gives a better visual segmentation result than experimental set 1. This is denoted as $2 \succ_{\text{v}} 1$.

3. Both experimental set 1 and experimental set 2 give segmentations of similar visual quality. It includes both cases where the segmentations generated by the two experimental sets are either both good or both bad. This is denoted as $1 \simeq_{\text{v}} 2$.

For each of the above visual comparisons, we count how many benchmark images belong to each category. The results are summarized in Table 8.16.

Experimental set 1	Experimental set 2	$1 \succ_{\mathrm{v}} 2$	$2 \succ_{\mathrm{v}} 1$	$1 \simeq_{\mathrm{v}} 2$
SHI-NC-EXP	COMB(HNC)-EXP	2	14	4
SHI-qNC-EXP	COMB(qHNC)-EXP	0	20	0
SHI-NC-IC	COMB(HNC)-IC	10	5	5
SHI-qNC-IC	COMB(qHNC)-IC	0	5	15
SWEEP-NC-EXP	COMB-NC-EXP	2	8	10
SWEEP-qNC-EXP	COMB-qNC-EXP	3	8	9
COMB(HNC)-EXP	COMB(qHNC)-EXP	0	7	13
COMB-NC-EXP	COMB(HNC)-EXP	0	14	6
COMB-qNC-EXP	COMB(qHNC)-EXP	0	13	7

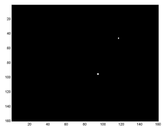
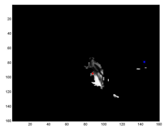
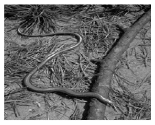
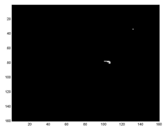
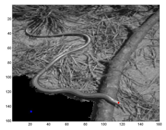
Table 8.16: Visual comparison results.

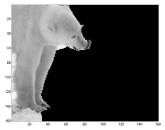
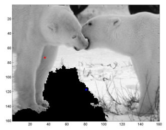
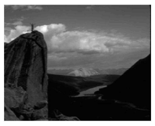
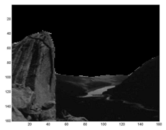
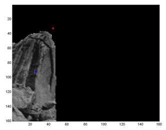
Based on the data in the first six rows of Table 8.16, we find that with exponential similarity weights, in general the visual quality of segmentations generated by COMB is superior to both SHI and SWEEP. If the q-NC (or q-HNC) optimization objective is applied, the visual superiority of COMB over SHI and SWEEP is dominant. Based on the data in the seventh row of Table 8.16, we find that q-HNC works better visually than HNC. According to the data in the last two rows of Table 8.16, we find that the criteria NC or q-NC are *not* good segmentation criteria. Since the visually best segmentations are obtained through solving HNC or q-HNC, they should be preferred segmentation criteria, for good visual segmentation quality and their tractability.

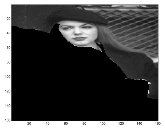
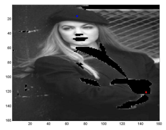
We find from Table 8.16 that in general SHI-NC-IC delivers best visual segmentations among all the experimental sets using method SHI or SWEEP. That is, SHI works better with intervening contour similarity weights. We also find that COMB(HNC)-EXP provides better visual results than COMB(HNC)-IC, meaning that COMB works better with exponential similarity weights.

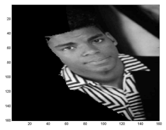
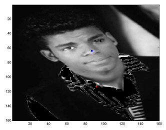
We provide here the images and their segmentations for the two leading methods, SHI-NC-IC and COMB(HNC)-EXP. The segmentation is displayed by setting all the pixels in the background part to be black. For each of the twenty benchmark images shown, we give the segmentations generated by SHI-NC-IC and COMB(HNC)-EXP in Figure 8.4. Notice that for Images 1, 2, 10, 12, 19 and 20, the segmentations generated by SHI-NC-IC are almost entirely black. This is because the segmentations of these images by SHI-NC-IC have almost all pixels in the background parts.

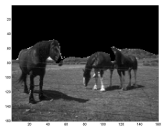
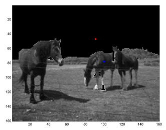
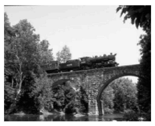
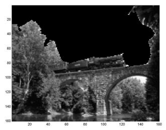
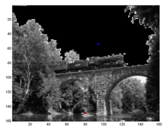
Our judgment is that for Image 1, Image 2, Image 5, Image 10, Image 12, Image 13, Image 14, Image 15, Image 16, Image 17, Image 19 and Image 20, COMB(HNC)-EXP gives visually better segmentations than SHI-NC-IC; for Image 3 and Image 6, SHI-NC-IC is visually better than COMB(HNC)-EXP; for Image 4, Image 7, Image 8 and Image 18, both SHI-NC-IC and COMB(HNC)-EXP generate visually good segmentations of similar quality; for the rest two images, Image 9 and Image 11, neither COMB(HNC)-EXP nor SHI-NC-IC

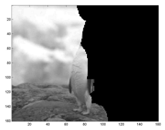
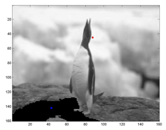
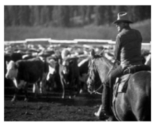
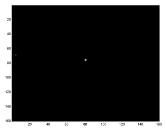
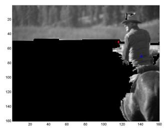
Img1-Ori

Img1-SHI

Img1-COMB

Img2-Ori

Img2-SHI

Img2-COMB

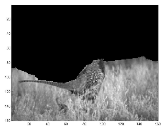
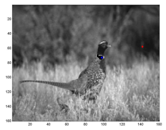
Img3-Ori

Img3-SHI

Img3-COMB

Img4-Ori

Img4-SHI

Img4-COMB

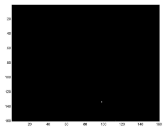
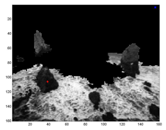
Img5-Ori

Img5-SHI

Img5-COMB

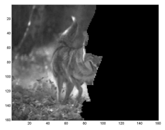
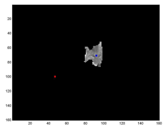
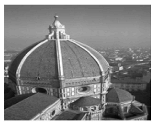
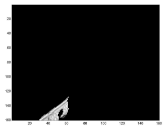
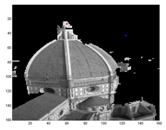
Img6-Ori

Img6-SHI

Img6-COMB

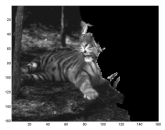
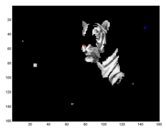
Img7-Ori

Img7-SHI

Img7-COMB

Img8-Ori

Img8-SHI

Img8-COMB

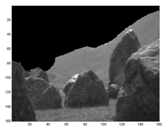
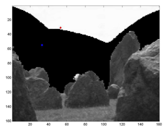
Img9-Ori

Img9-SHI

Img9-COMB

Img10-Ori

Img10-SHI

Img10-COMB

Img11-Ori

Img11-SHI

Img11-COMB

Img12-Ori

Img12-SHI

Img12-COMB

Img13-Ori

Img13-SHI

Img13-COMB

Img14-Ori

Img14-SHI

Img14-COMB

Img15-Ori

Img15-SHI

Img15-COMB

Img16-Ori

Img16-SHI

Img16-COMB

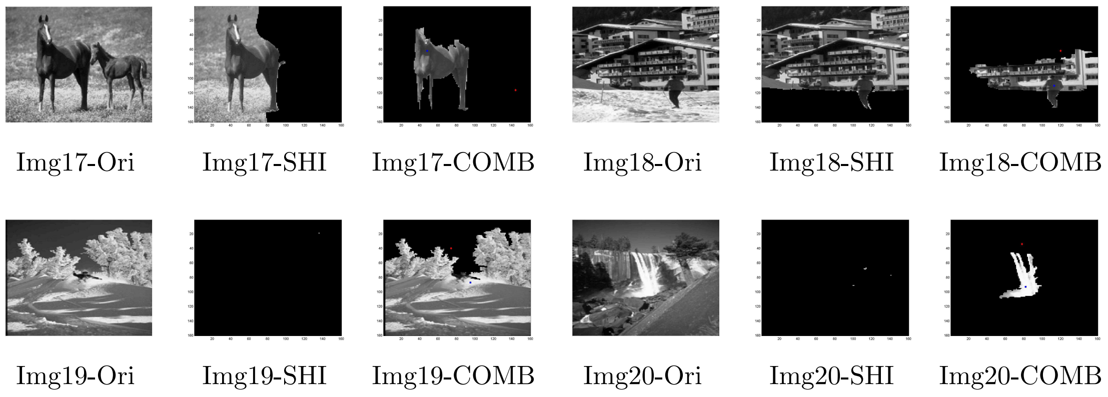


Figure 8.4: The visual segmentations of SHI-NC-IC (-SHI) and COMB(HNC)-EXP (-COMB), and their respective original image (-Ori).

gives visually good segmentations.

8.8 Conclusions

We report here on detailed experiments conducted on algorithms for the normalized cut and its generalization as quantity normalized cut applied to image segmentation problems. We find that, in general, the combinatorial flow algorithm of [49, 43] outperforms the spectral method both numerically and visually. In most cases, the combinatorial algorithm yields tighter objective function values of the two criteria we test. Furthermore, we find that the combinatorial algorithm almost always produces a visual segmentation which is at least as good as that of the spectral method, and often better.

Another important finding in our experiments, is that contrary to prevalent belief, the normalized cut criterion is not a good model for image segmentation, since it does *not* provide good quality solutions, in terms of visual quality. Moreover, the normalized cut problem is NP-hard. We conclude that instead of modeling the image segmentation problem as the normalized cut problem, it is more effective to model and solve the problem as the polynomial time solvable Hochbaum normalized cut problem.

For future research, we plan on investigating other methods of solving image segmentation and other clustering problems, such as the k-means clustering method discussed in [31, 32].

Chapter 9

Concluding Remarks

In the dissertation, we enlarge the algorithmic toolbox of solving the MRF-BL and MRF models on simple path graphs. We introduce three different algorithmic techniques for different types of objective functions. We explore the application of graph minimum cut and network flow based algorithms to solve MRF-BL on a path, and we are the first to propose directly solving the KKT optimality conditions of MRF on a path by leveraging the simplicity of a path graph. The three techniques all yield the fastest algorithms for the respective class of problems. As paths are building blocks of more complicated graphs, one may use the algorithms devised here as subroutines to solve MRF and related problems on more complicated graphs.

For image segmentation, the first interesting lesson learned from the empirical study is that computational complexity should be taken into account when we build models. Image segmentation is one of the types of problems with no consensus on criterion. Although the normalized cut criterion is theoretically intuitive and intriguing, its NP-hardness discounts its performance in practice as no algorithms are actually solving the exact normalized cut problem, but instead some sorts of relaxations. On the other hand, as we have shown in the experimental study, some polynomially solvable criteria, such as Hochbaum normalized cut, not only enjoy theoretical tractability, but also deliver practically better quality solutions. As a result, these kinds of criteria should be preferred. Another interesting observation made through the empirical comparison between the combinatorial algorithm and the spectral method is the gap between theory and practice. The spectral method, although celebrating some nice theoretical properties, does not perform as good as the combinatorial algorithm in practice for image segmentation.

Bibliography

[1] R. K. Ahuja and D. S. Hochbaum. “Solving Linear Cost Dynamic Lot-Sizing Problems in $O(n \log n)$ Time”. In: *Operations Research* 56.1 (2008), pp. 255–261.

[2] R. K. Ahuja, D. S. Hochbaum, and J. B. Orlin. “Solving the Convex Cost Integer Dual Network Flow Problem”. In: *Management Science* 49.7 (2003), pp. 950–964.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. 1993.

[4] R. K. Ahuja and J. B. Orlin. “A Fast Scaling Algorithm for Minimizing Separable Convex Functions subject to Chain Constraints”. In: *Operations Research* 49.5 (2001).

[5] M. Alamgir and U. V. Luxburg. “Phase Transition in the Family of p-Resistances”. In: *Neural Information Processing Systems (NIPS)* (2011), pp. 379–387.

[6] N. Alon. “Eigenvalues and Expanders”. In: *Combinatorica* 6.2 (1986), pp. 83–96.

[7] N. Alon and V. D. Milman. “λ_1, Isoperimetric Inequalities for Graphs, and Superconcentrators”. In: *Journal of Combinatorial Theory, Series B* 38.1 (1985), pp. 73–88.

[8] S. Angelov et al. “Weighted Isotonic Regression under the L_1 Norm”. In: *Symposium on Discrete Algorithms (SODA)* (2006), pp. 783–791.

[9] M. Ayer et al. “An Empirical Distribution Function for Sampling with Incomplete Information”. In: *Annals of Mathematical Statistics* 26.4 (1955), pp. 641–647.

[10] R. E. Barlow and H. D. Brunk. “The Isotonic Regression Problem and its Dual”. In: *Journal of the American Statistical Association* 67.337 (1972), pp. 140–147.

[11] R. E. Barlow et al. *Statistical Inference under Order Restrictions: The Theory and Application of Isotonic Regression*. New York: Wiley, 1972.

[12] D. J. Bartholomew. “A Test for Homogeneity for Ordered Alternatives”. In: *Biometrika* 46.1 (1959), pp. 36–48.

[13] D. J. Bartholomew. “A Test for Homogeneity for Ordered Alternatives II”. In: *Biometrika* 46.3 (1959), pp. 328–335.

[14] A. Blake and A. Zisserman. *Visual Reconstruction*. Boston: MIT Press, 1987.

[15] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge: Cambridge University Press, 2004.

[16] Y. Boykov, O. Veksler, and R. Zabih. "Markov Random Fields with Efficient Approximations". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (1998), pp. 648–655.

[17] N. Bridle and X. J. Zhu. "p-Voltage: Laplacian Regularization for Semi-supervised Learning on High-dimensional Data". In: *Eleventh Workshop on Mining and Learning with Graphs (MLG2013)* (2013).

[18] N. Chakravarti. "Isotonic Median Regression: A Linear Programming Approach". In: *Mathematics of Operations Research* 14.2 (1989), pp. 303–308.

[19] A. Chambolle and T. Pock. "A First-order Primal-dual Algorithm for Convex Problems with Applications to Imaging". In: *Journal of Mathematical Imaging and Vision* 40.1 (2011), pp. 120–145.

[20] T. Chan and S. Esedoglu. "Aspects of Total Variation Regularized L^1 Function Approximation". In: *SIAM Journal on Applied Mathematics* 65.5 (2005), pp. 1817–1837.

[21] B. G. Chandran and D. S. Hochbaum. "A Computational Study of the Pseudoflow and Push-relabel Algorithms for the Maximum Flow Problem". In: *Operations Research* 57.2 (2009), pp. 358–376.

[22] B. G. Chandran and D. S. Hochbaum. "Pseudoflow Parametric Maximum Flow Solver Version 1.0". In: *http://riot.ieor.berkeley.edu/Applications/Pseudoflow/parametric.html* (2012).

[23] J. Cheeger. "A Lower Bound for the Smallest Eigenvalue of the Laplacian". In: *Problems in Analysis, R. C. Gunning, ed., Princeton University Press* (1970), pp. 195–199.

[24] F. R. K. Chung. "Four Proofs for the Cheeger Inequality and Graph Partition Algorithms". In: *Proceedings of ICCM* 2 (2007), p. 378.

[25] F. R. K. Chung. *Spectral Graph Theory.* American Mathematical Soc., 1997.

[26] G. B. Coleman and H. C. Andrews. "Image Segmentation by Clustering". In: *Proceedings of the IEEE* 67.5 (1979), pp. 773–785.

[27] T. H. Cormen et al. *Introduction to Algorithms.* Cambridge, MA: The MIT Press, 2009.

[28] T. Cour, S. Yu, and J. Shi. "MATLAB Normalized Cut Image Segmentation Code". In: *http://www.cis.upenn.edu/~jshi/software/* (2011).

[29] K. Dembczyński, W. Kotlowski, and R. Slowiński. "Learning Rule Ensembles for Ordinal Classification with Monotonicity Constraints". In: *Fundamenta Informaticae* 94.2 (2009), pp. 163–178.

[30] P. A. Dhawan. *Medical Imaging Analysis*. Hoboken, NJ: Wiley-Interscience Publication, 2003.

[31] I. S. Dhillon, Y. Q. Guan, and B. Kulis. “Kernel k-Means: Spectral Clustering and Normalized Cuts”. In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2004), pp. 551–556.

[32] I. S. Dhillon, Y. Q. Guan, and B. Kulis. “Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 29.11 (2007), pp. 1944–1957.

[33] W. E. Donath and A. J. Hoffman. “Lower Bounds for the Partitioning of Graphs”. In: *IBM Journal of Research and Development* 17.5 (1973), pp. 420–425.

[34] L. Dümbgen and A. Kovac. “Extensions of Smoothing via Taut Strings”. In: *Electronic Journal of Statistics* 3 (2009), pp. 41–75.

[35] P. H. C. Eilers and R. X. de Menezes. “Quantile Smoothing of Array CGH Data”. In: *Bioinformatics* 21.7 (2005), pp. 1146–1153.

[36] A. El Alaoui et al. “Asymptotic Behavior of ℓ_p-based Laplacian Regularization in Semi-supervised Learning”. In: *JMLR: Workshop and Conference Proceedings* 49 (2016), pp. 1–28.

[37] P. F. Felzenszwalb and R. Zabih. “Dynamic Programming and Graph Algorithms in Computer Vision”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 33.4 (2011), pp. 721–740.

[38] M. Fiedler. “A Property of Eigenvectors of Nonnegative Symmetric Matrices and its Applications to Graph Theory”. In: *Czechoslovak Mathematical Journal* 25.4 (1975), pp. 619–633.

[39] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. “A Fast Parametric Maximum Flow Algorithm and Applications”. In: *SIAM Journal on Computing* 18.1 (1989), pp. 30–55.

[40] D. Geiger and F. Girosi. “Parallel and Deterministic Algorithms for MRFs: Surface Reconstruction”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 13 (1991), pp. 401–412.

[41] S. Geman and D. Geman. “Stochastic Relaxation, Gibbs Distributions and the Bayesian Restoration of Images”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 6 (1984), pp. 721–741.

[42] S. J. Grotzinger and C. Witzgall. “Projections onto Simplices”. In: *Applied Mathematics and Optimization* 12.1 (1984), pp. 247–270.

[43] D. S. Hochbaum. “A Polynomial Time Algorithm for Rayleigh Ratio on Discrete Variables: Replacing Spectral Techniques for Expander Ratio, Normalized Cut and Cheeger Constant”. In: *Operations Research* 61.1 (2013). Early version in Replacing Spectral Techniques for Expander Ratio, Normalized Cut and Conductance by Combinatorial Flow Algorithms. arXiv:1010.4535v1 [math.OC], 2010, pp. 184–198.

[44] D. S. Hochbaum. “An Efficient Algorithm for Image Segmentation, Markov Random Fields and Related Problems”. In: *Journal of the ACM* 48.2 (2001), pp. 686–701.

[45] D. S. Hochbaum. “Complexity and Algorithms for Nonlinear Optimization Problems”. In: *Annals of Operations Research* 153.1 (2007), pp. 257–296.

[46] D. S. Hochbaum. *Graph Algorithms and Network Flows*. UC Berkeley: Lecture Notes for IEOR 266, 2016.

[47] D. S. Hochbaum. “Lower and Upper Bounds for the Allocation Problem and Other Nonlinear Optimization Problems”. In: *Mathematics of Operations Research* 19.2 (1994), pp. 390–409.

[48] D. S. Hochbaum. “Multi-label Markov Random Fields as an Efficient and Effective Tool for Image Segmentation, Total Variations and Regularization”. In: *Numerical Mathematics: Theory, Methods and Applications* 6.1 (2013), pp. 169–198.

[49] D. S. Hochbaum. “Polynomial Time Algorithms for Ratio Regions and a Variant of Normalized Cut”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 32.5 (2010), pp. 889–898.

[50] D. S. Hochbaum. “The Pseudoflow Algorithm: A New Algorithm for the Maximum Flow Problem”. In: *Operations Research* 56.4 (2008), pp. 992–1009.

[51] D. S. Hochbaum, C. Lu, and E. Bertelli. “Evaluating Performance of Image Segmentation Criteria and Techniques”. In: *EURO Journal on Computational Optimization* 1.1-2 (2013), pp. 155–180.

[52] D. S. Hochbaum and J. B. Orlin. “Simplifications and Speedups of the Pseudoflow Algorithm”. In: *Networks* 61.1 (2013), pp. 40–57.

[53] D. S. Hochbaum and M. Queyranne. “Minimizing a Convex Cost Closure Set”. In: *SIAM Journal on Discrete Mathematics* 16.2 (2003). Extended abstract in Proceedings of the 8th Annual European Symposium on Algorithms, ESA, 2000, pp. 192–207.

[54] D. S. Hochbaum and J. G. Shanthikumar. “Convex Separable Optimization is not Much Harder than Linear Optimization”. In: *Journal of the ACM* 37.4 (1990), pp. 843–862.

[55] H. Hoefling. “A Path Algorithm for the Fused Lasso Signal Approximator”. In: *Journal of Computational and Graphical Statistics* 19.4 (2010), pp. 984–1006.

[56] K. Hohm, M. Storath, and A. Weinmann. "An Algorithmic Framework for Mumford-Shah Regularization of Inverse Problems in Imaging". In: *Inverse Problems* 31.11 (2015), p. 115011.

[57] M. S. Hosseini, B. N. Araabi, and H. Soltanian-Zadeh. "Pigment Melanin: Pattern for Iris Recognition". In: *IEEE Transactions on Instrumentation and Measurement* 59.4 (2010), pp. 792–804.

[58] H. Ishikawa and D. Geiger. "Segmentation by Grouping Junctions". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (1998), pp. 125–131.

[59] N. A. Johnson. "A Dynamic Programming Algorithm for the Fused Lasso and L_0-Segmentation". In: *Journal of Computational and Graphical Statistics* 22.2 (2013), pp. 246–260.

[60] A. T. Kalai and R. Sastry. "The Isotron Algorithm: High-dimensional Isotonic Regression". In: *Proceedings of the Conference on Learning Theory (COLT)* (2009).

[61] Y. Kaufman and A. Tamir. "Locating Service Centers with Precedence Constraints". In: *Discrete Applied Mathematics* 47.3 (1993), pp. 251–261.

[62] V. Kolmogorov, T. Pock, and M. Rolinek. "Total Variation on a Tree". In: *SIAM Journal on Imaging Sciences* 9.2 (2016), pp. 605–636.

[63] R. Kyng et al. "Algorithms for Lipschitz Learning on Graphs". In: *Proceedings of the Conference on Learning Theory (COLT)* (2015), pp. 1190–1223.

[64] T. Leung and J. Malik. "Contour Continuity in Region Based Image Segmentation". In: *European Conference on Computer Vision, Springer Berlin Heidelberg* (1998), pp. 544–559.

[65] S. Z. Li, K. L. Chan, and H. Wang. "Bayesian Image Restoration and Segmentation by Constrained Optimization". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (1996), pp. 1–6.

[66] Y. J. Li and J. Zhu. "Analysis of Array CGH Data for Cancer Studies using Fused Quantile Regression". In: *Bioinformatics* 23.18 (2007), pp. 2470–2476.

[67] M. A. Little and N. S. Jones. "Generalized Methods and Solvers for Noise Removal from Piecewise Constant Signals. I. Background Theory". In: *Proceedings of the Royal Society A* 467.2135 (2011), pp. 3088–3114.

[68] D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming.* Springer, 2016.

[69] R. Luss and S. Rosset. "Generalized Isotonic Regression". In: *Journal of Computational and Graphical Statistics* 23.1 (2014), pp. 192–210.

[70] R. Luss, S. Rosset, and M. Shahar. "Efficient Regularized Isotonic Regression with Application to Gene-gene Interaction Search". In: *The Annals of Applied Statistics* 6.1 (2012), pp. 253–283.

[71] J. Malik et al. “Contour and Texture Analysis for Image Segmentation”. In: *International Journal of Computer Vision* 43.1 (2001), pp. 7–27.

[72] E. Mammen and S. van de Geer. “Locally Adaptive Regression Splines”. In: *The Annals of Statistics* 25.1 (1997), pp. 387–413.

[73] W. L. Maxwell and J. A. Muckstadt. “Establishing Consistent and Realistic Reorder Intervals in Production-Distribution Systems”. In: *Operations Research* 33.6 (1985), pp. 1316–1341.

[74] J. A. Mendendez and B. Salvador. “An Algorithm for Isotonic Median Regression”. In: *Computational Statistics & Data Analysis* 5.4 (1987), pp. 399–406.

[75] R. E. Miles. “The Complete Amalgamation into Blocks, by Weighted Means, of a Finite Set of Real Numbers”. In: *Biometrika* 46.3 (1959), pp. 317–327.

[76] B. Nadler, N. Srebro, and X. Y. Zhou. “Semi-supervised Learning with the Graph Laplacian: The Limit of Infinite Unlabelled Data”. In: *Neural Information Processing Systems (NIPS)* (2009), pp. 1330–1338.

[77] M. Nikolova. “Minimizers of Cost-functions Involving Nonsmooth Data-fidelity Terms. Application to the Processing of Outliers”. In: *SIAM Journal on Numerical Analysis* 40.3 (2002), pp. 965–994.

[78] J. B. Orlin. “Max Flow in $O(nm)$ Time, or Better”. In: *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing (STOC)* (2013), pp. 765–774.

[79] A. Painsky and S. Rosset. “Isotonic Modeling with Non-differentiable Loss Functions with Application to Lasso Regularization”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 38.2 (2016), pp. 308–321.

[80] T. N. Pappas. “An Adaptive Clustering Algorithm for Image Segmentation”. In: *IEEE Transactions on Signal Processing* 40.4 (1992), pp. 901–914.

[81] P. M. Pardalos, G. L. Xue, and L. Yong. “Efficient Computation of an Isotonic Median Regression”. In: *Applied Mathematics Letters* 8.2 (1995), pp. 67–70.

[82] D. L. Pham, C. Y. Xu, and J. L. Prince. “Current Methods in Medical Image Segmentation”. In: *Annual Review of Biomedical Engineering* 2.1 (2000), pp. 315–337.

[83] K. Punera and J. Ghosh. “Enhanced Hierarchical Classification via Isotonic Smoothing”. In: *Proceedings of the 17th International Conference on World Wide Web* (2008), pp. 151–160.

[84] A. Restrepo and A. C. Bovik. “Locally Monotonic Regression”. In: *IEEE Transactions on Signal Processing* 41.9 (1993), pp. 2796–2810.

[85] T. Robertson and P. Waltman. “On Estimating Monotone Parameters”. In: *The Annals of Mathematical Statistics* 39.3 (1968), pp. 1030–1039.

[86] T. Robertson and F. T. Wright. “Algorithms in Order Restricted Statistical Inference and the Cauchy Mean Value Property”. In: *The Annals of Statistics* 8.3 (1980), pp. 645–651.

[87] T. Robertson and F. T. Wright. “Multiple Isotonic Median Regression”. In: *The Annals of Statistics* 1.3 (1973), pp. 422–432.

[88] T. Robertson, F. T. Wright, and R. L. Dykstra. *Order Restricted Statistical Inference*. New York: Wiley, 1988.

[89] C. A. Roobottom, G. Mitchell, and G. Morgan-Hughes. “Radiation-Reduction Strategies in Cardiac Computed Tomographic Angiography”. In: *Clinical Radiology* 65.11 (2010), pp. 859–867.

[90] R. Roundy. “98%-Effective Integer-ratio Lot-sizing for One-warehouse Multi-retailer systems”. In: *Management Science* 31.11 (1985), pp. 1416–1430.

[91] L. I. Rudin, S. J. Osher, and E. Fatemi. “Nonlinear Total Variation based Noise Removal Algorithms”. In: *Physica D: Nonlinear Phenomena* 60.1-4 (1992), pp. 259–268.

[92] Y. U. Ryu, R. Chandrasekaran, and V. Jacob. “Prognosis using an Isotonic Prediction Technique”. In: *Management Science* 50.6 (2004), pp. 777–785.

[93] L. G. Shapiro and G. C. Stockman. *Computer Vision*. New Jersey: Prentice-Hall, 2001.

[94] E. Sharon et al. “Hierarchy and Adaptivity in Segmenting Visual Scenes”. In: *Nature* 442.7104 (2006), pp. 810–813.

[95] J. Shi and J. Malik. “Normalized Cuts and Image Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 22.8 (2000), pp. 888–905.

[96] T. S. Shively, S. G. Walker, and P. Damien. “Nonparametric Function Estimation subject to Monotonicity, Convexity and Other Shape Constraints”. In: *Journal of Econometrics* 161.2 (2011), pp. 166–181.

[97] D. D. Sleator and R. E. Tarjan. “A Data Structure for Dynamic Trees”. In: *Journal of Computer and System Sciences* 26.3 (1983), pp. 362–391.

[98] M. Storath, A. Weinmann, and M. Unser. “Exact Algorithms for L^1-TV Regularization of Real-valued or Circle-valued Signals”. In: *SIAM Journal on Scientific Computing* 38.1 (2016), A614–A630.

[99] Q. F. Stout. “Isotonic Regression for Multiple Independent Variables”. In: *Algorithmica* 71.2 (2015), pp. 450–470.

[100] Q. F. Stout. “Isotonic Regression via Partitioning”. In: *Algorithmica* 66.1 (2013), pp. 93–112.

[101] W. Tansey and J. G. Scott. “A Fast and Flexible Algorithm for the Graph-fused Lasso”. In: *arXiv: 1505.06475v3* (2015).

[102] R. J. Tibshirani, H. Hoefling, and R. Tibshirani. “Nearly-isotonic Regression”. In: *Technometrics* 53.1 (2011), pp. 54–61.

[103] R. J. Tibshirani and J. Taylor. “The Solution Path of the Generalized Lasso”. In: *The Annals of Statistics* 39.3 (2011), pp. 1335–1371.

[104] R. Tibshirani et al. “Sparsity and Smoothness via the Fused Lasso”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.1 (2005), pp. 91–108.

[105] D. A. Tolliver and G. L. Miller. “Graph Partitioning by Spectral Rounding: Applications in Image Segmentation and Clustering”. In: *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Volume 1* (2006), pp. 1053–1060.

[106] A. F. Jr. Veinott. “Least *d*-Majorized Network Flows with Inventory and Statistical Applications”. In: *Management Science* 17.9 (1971), pp. 547–567.

[107] H. S. Wang, G. D. Li, and G. H. Jiang. “Robust Regression Shrinkage and Consistent Variable Selection through the LAD-Lasso”. In: *Journal of Business & Economic Statistics* 25.3 (2007), pp. 347–355.

[108] A. Weinmann, L. Demaret, and M. Storath. “Mumford-Shah and Potts Regularization for Manifold-Valued Data”. In: *Journal of Mathematical Imaging and Vision* 55.3 (2016), pp. 428–445.

[109] A. Weinmann, L. Demaret, and M. Storath. “Total Variation Regularization for Manifold-valued Data”. In: *SIAM Journal on Imaging Sciences* 7.4 (2014), pp. 2226–2257.

[110] A. Weinmann and M. Storath. “Iterative Potts and Blake-Zisserman Minimization for the Recovery of Functions with Discontinuities from Indirect Measuremens”. In: *Proceedings of the Royal Society A* 471.2176 (2015), p. 20140638.

[111] Z. Wu and R. Leahy. “An Optimal Graph Theoretic Approach to Data Clustering: Theory and its Application to Image Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 15.11 (1993), pp. 1101–1113.

[112] E. P. Xing and M. I. Jordan. “On Semidefinite Relaxations for Normalized k-Cut and Connections to Spectral Clustering”. In: *Technical Report No. UCB/CSD-3-1265* (2003).

[113] S. X. Yu and J. Shi. “Multiclass Spectral Clustering”. In: *Proceedings of International Conference on Computer Vision* (2003), pp. 313–319.

[114] X. J. Zhu, Z. Ghahramani, and J. Lafferty. “Semi-supervised Learning using Gaussian Fields and Harmonic Functions”. In: *Proceedings of The 31st International Conference on Machine Learning (ICML)* 3 (2003), pp. 912–919.

Appendix A

Red-Black Tree Data Structure to Maintain s-Intervals

A red-black tree is a binary search tree. Each node of the tree contains the following five fields [27]:
color: The "color" of a node. Its value is either RED or BLACK.
key: The "key" value of a node. It is a scalar.
left, right: The pointers to the left and the right child of a node. If the corresponding child does not exist, the corresponding pointer has value NIL.
p: The pointer to the parent of a node. If the node is the root node, the pointer value is NIL.

As it is a binary search tree, the keys of the nodes are comparable. Furthermore, it has the following two properties [27]:

1. Binary-search-tree property: Let x be a node in a binary search tree. If y is a node in the left subtree of x, then $key[y] \leq key[x]$. If y is a node in the right subtree of x, then $key[y] \geq key[x]$.

2. Tree height property: A red-black tree with n nodes has height at most $2\log(n+1)$.

We use a red-black tree data structure T to represent the set of s-intervals. Each node of the tree represents one s-interval. Due to the disjointness property of s-intervals, every s-interval is uniquely identified by its two endpoints. Thus we extend the *key* field from a scalar to a *tuple*: For a node x representing an s-interval $[i_\ell, i_r]$, the *key* field of x, $key[x]$, contains two values, $key[x].second$ and $key[x].second$, such that $key[x].first = i_\ell$ and $key[x].second = i_r$. As a result, we also define a comparison between the *key tuples* of two nodes, which can also be viewed as a comparison between their corresponding s-intervals: For any two nodes x_1 (representing an s-interval $[i_{\ell 1}, i_{r1}]$) and x_2 (representing an s-interval $[i_{\ell 2}, i_{r2}]$), we define:

1. $key[x_1] < key[x_2]$: if $key[x_1].second = i_{r1} < i_{\ell 2} = key[x_2].first$. It is the case where $[i_{\ell 1}, i_{r1}]$ is on the left of $[i_{\ell 2}, i_{r2}]$.

2. $key[x_1] = key[x_2]$: if $key[x_1].first = i_{\ell 1} = i_{\ell 2} = key[x_2].first$. It implies that $key[x_1].second = i_{r1} = i_{r2} = key[x_2].second$. It is the case where x_1 and x_2 refer to the same tree node and the same s-interval.

3. $key[x_1] > key[x_2]$: if $key[x_1].first = i_{\ell 1} > i_{r2} = key[x_2].second$. It is the case where $[i_{\ell 1}, i_{r1}]$ is on the right of $[i_{\ell 2}, i_{r2}]$.

As s-intervals do not overlap, the comparison of any two (possibly identical) nodes in the tree must fall into exactly one of the above three outcomes.

The above is our only extension of the red-black tree discussed in [27] in the algorithm presented in Chapter 5. Since any two different nodes in the tree represent different s-intervals, the binary-search-tree property still holds with the two inequalities are strict, i.e., "$\le$" $\rightarrow$ "$<$". The tree height property still satisfies as its proof in [27] does not involve the *key* fields.

A.1 Initializing the Red-Black Tree with a Single s-Interval

In each of the two algorithms presented in Chapter 5, the red-black tree is initialized with a single s-interval $[1, n]$. This is achieved by a subroutine $z :=$ new_node(i_ℓ, i_r) initializing a new node z for s-interval $[i_\ell, i_r]$, such that $color[z] = RED$, $key[z].first = i_\ell$, $key[z].second = i_r$, and $left[z] = right[z] = p[z] = NIL$ [27]. This is done in $O(1)$ time [27]. Thus the initialization of the tree is completed by calling $z :=$ new_node$(1, n)$. Node z is the root of the tree.

A.2 Pseudo-code of Subroutine $[i_{k\ell}, i_{kr}] :=$ get_s_interval(i_k)

Let $root[T]$ represent the root node of the red-black tree T. The pseudo-code of get_s_interval is:

```
$[i_{k\ell}, i_{kr}] :=$ get_s_interval$(i_k)$
begin

    $z := root[T]$;

    while $z \neq NIL$:

        if $i_k \ge key[z].first$ and $i_k \le key[z].second$ {node $i_k$ is in the $s$-interval represented
        by node $z$}
        then $i_{k\ell} := key[z].first$,  $i_{kr} := key[z].second$; return $[i_{k\ell}, i_{kr}]$;
```

```
            else if $i_k < key[z].first$
            then $z := left[z]$;
            else $z := right[z]$;
            end if
        end while
end
```

The correctness of the pseudo-code is justified by the binary-search-tree property with the extended comparison for the *key tuples*. The complexity is determined by the height of the tree. Note that since all s-intervals are originally decomposed from the initial s-interval $[1, n]$, hence the number of s-intervals generated throughout the algorithm is at most n. Thus the red-black tree has at most n nodes. Therefore the tree height is at most $2\log(n+1)$. As a result, the complexity of get_s_interval is $O(\log n)$.

A.3 Pseudo-code of Subroutine update_s_interval$(i_{k\ell}, i^*_{k1}, i^*_{k2}, i_{kr})$

Cormen et al. in [27] define and analyze the following three operations on a red-black tree with scalar *key* values:

1. TREE-SEARCH(T, k): Searching for a node in red-black tree T with a given *key* value k. It returns a pointer to a node with *key* k if one exists; otherwise it return NIL. In our case in Chapter 5 the *key* value is a tuple and the comparison of scalar *key* values is extended to *key tuples*.
2. RB-INSERT(T,z): Inserting a node z into red-black tree T. The pseudo-code involves comparing the *key* values of two nodes, where we can simply apply our definition of *key tuple* comparison. As a result, *literally* there is no change to the pseudo-code of RB-INSERT(T, z) in our extension.
3. RB-DELETE(T, z): Deleting a node z from red-black tree T. The pseudo-code does not involve the *key* field, hence it is directly applicable to our extension.

Cormen et al. in [27] prove that each of the above operation has complexity $O(\log n)$ for a tree of at most n nodes. The complexities are the same in our case with *key tuples*.

update_s_interval is implemented by calling the above three built-in operations, and the new_node subroutine. It changes the red-black tree T.

```
update_s_interval$(i_{k\ell}, i^*_{k1}, i^*_{k2}, i_{kr})$
begin
    $z :=$ TREE-SEARCH$(T, (i_{k\ell}, i_{kr}))$;
```

```
        RB-DELETE($T, z$);
        if $i_{k\ell} \leq i^*_{k1} - 1$ then $z :=$ new_node($i_{k\ell}, i^*_{k1} - 1$); RB-INSERT($T, z$); end if
        if $i^*_{k2} + 1 \leq i_{kr}$ then $z :=$ new_node($i^*_{k2} + 1, i_{kr}$); RB-INSERT($T, z$); end if
end
```

As the tree has at most n nodes, each call to update_s_interval takes $O(\log n)$ time.

Appendix B

Dynamic Path Data Structure to Maintain Chapter 5's Four Arrays

Dynamic path [97] is a data structure for a collection of vertex-disjoint paths. Each path in the collection is an undirected symmetric path, where each edge has a real-valued cost.

Each internal vertex of a path has two edges adjacent to it. To distinguish the two edges, we define the following terminology. We designate one end of the path as *head* and the other end as *tail* [97]. For any internal vertex v, we define the vertex *before* vertex v as the index of the adjacent vertex of v that is closer to the head of the path. Similarly, we define the vertex *after* vertex v as the index of the adjacent vertex of v that is closer to the tail of the path [97]. For head vertex v, the vertex before vertex v does not exist. Likewise, for tail vertex v, the vertex after vertex v does not exist. The designation of the head and tail vertices is arbitrary. If the head and tail vertices are reversed, the references to the vertices before and after vertex v are also reversed accordingly.

The following 11 operations are supported in dynamic paths [97]:

$p := \mathit{path}(v)$: Return the path p containing vertex v.

$v := \mathit{head}(p)$: Return the head vertex v of path p.

$v := \mathit{tail}(p)$: Return the tail vertex v of path p.

$u := \mathit{before}(v)$: Return the vertex u before vertex v on $\mathit{path}(v)$. If v is the head of the path, return NIL.

$u := \mathit{after}(v)$: Return the vertex u after vertex v on $\mathit{path}(v)$. If v is the tail of the path, return NIL.

$x := \mathit{pcost}(v)$: Return the real-valued cost x of the edge $(v, \mathit{after}(v))$. If vertex v is the tail of the path, return NIL.

$v := \mathit{pmincost}(p)$: Return the vertex v closest to $\mathit{tail}(p)$ such that $(v, \mathit{after}(v))$ has minimum cost among edges on path p. If p contains only one vertex (degenerate case), return NIL.

$\mathit{pupdate}(p, x)$: Add real value x to the cost of every edge on path p.

$\mathit{reverse}(p)$: Reverse the direction of path p, making the head as the tail and vice versa.

$p_3 := \mathit{concatenate}(p_1, p_2, x)$: Merge paths p_1 and p_2 by adding the edge $(\mathit{tail}(p_1), \mathit{head}(p_2))$ of real-valued cost x. Return the merged path p_3.

$[p_1, p_2, x, y] := \mathit{split}(v)$: Divide $\mathit{path}(v)$ into (up to) three parts by deleting the edges incident to v. Return a list $[p_1, p_2, x, y]$, where p_1 is the subpath consisting of all vertices from $\mathit{head}(\mathit{path}(v))$ to $\mathit{before}(v)$, p_2 is the subpath consisting of all vertices from $\mathit{after}(v)$ to $\mathit{tail}(\mathit{path}(v))$, x is the cost of the deleted edge $(\mathit{before}(v), v)$, and y is the cost of the deleted edge $(v, \mathit{after}(v))$. If v is originally the head of $\mathit{path}(v)$, p_1 is NIL and x is undefined; if v is originally the tail of $\mathit{path}(v)$, p_2 is NIL and y is undefined.

A dynamic path is implemented as a full balanced binary tree [97]. Each vertex of the path is constructed as a leaf node of the tree and each edge of the path is constructed as a non-leaf node of the tree, which stores the edge cost as a node field. Besides, each node in the tree contains various other fields in support of efficient implementation of the above 11 operations on dynamic paths. The complete details of the binary tree implementation of a dynamic path was presented in [97], Chap. 4.

We highlight the complexity of each of the above operations: Sleator and Tarjan in [97] show that, for a collection of dynamic paths with a total of $O(n)$ vertices, $\mathit{head}(p)$, $\mathit{tail}(p)$, $\mathit{pupdate}(p, x)$ and $\mathit{reverse}(p)$ each takes $O(1)$ time and $\mathit{path}(v)$, $\mathit{before}(v)$, $\mathit{after}(v)$, $\mathit{pcost}(v)$, $\mathit{pmincost}(p)$, $\mathit{concatenate}(p_1, p_2, x)$ and $\mathit{split}(v)$ each takes $O(\log n)$ time.

We define the following two additional split operations that are more convenient to use for our purpose:

$[p_1, p_2, x] := \mathit{split\text{-}before}(v)$: Divide $\mathit{path}(v)$ into (up to) two parts by deleting the edge $(\mathit{before}(v), v)$. Return a list $[p_1, p_2, x]$, where p_1 is the subpath consisting of all vertices from $\mathit{head}(\mathit{path}(v))$ to $\mathit{before}(v)$, p_2 is the subpath consisting of all vertices from v to $\mathit{tail}(\mathit{path}(v))$, x is the cost of the deleted edge $(\mathit{before}(v), v)$. If v is originally the head of $\mathit{path}(v)$, p_1 is NIL and x is undefined.

$[p_1, p_2, y] := \mathit{split\text{-}after}(v)$: Divide $\mathit{path}(v)$ into (up to) two parts by deleting the edge $(v, \mathit{after}(v))$. Return a list $[p_1, p_2, y]$, where p_1 is the subpath consisting of all vertices from $\mathit{head}(\mathit{path}(v))$ to v, p_2 is the subpath consisting of all vertices from $\mathit{after}(v)$ to $\mathit{tail}(\mathit{path}(v))$, y is the cost of the deleted edge $(v, \mathit{after}(v))$. If v is originally the tail of $\mathit{path}(v)$, p_2 is NIL and y is undefined.

Both *split-before* and *split-after* can be implemented efficiently using *concatenate* and *split*:

```
$[p_1, p_2, x] :=$ split-before$(v)$
begin
    $[p_1, p_2, x, y] :=$ split$(v)$;
    if $p_2 \neq NIL$ then $p_2 :=$ concatenate$(v, p_2, y)$;
    else $p_2 := [v, v]$; {a path with single vertex $v$}
    end if
    return $[p_1, p_2, x]$;
end

$[p_1, p_2, y] :=$ split-after$(v)$
begin
    $[p_1, p_2, x, y] :=$ split$(v)$;
    if $p_1 \neq NIL$ then $p_1 :=$ concatenate$(p_1, v, x)$;
    else $p_1 := [v, v]$; {a path with single vertex $v$}
    end if
    return $[p_1, p_2, y]$;
end
```

Since each subroutine calls one *split* and one *concatenate* operation, the complexity of *split-before*(v) and *split-after*(v) each is $O(\log n)$. We include *split-before*(v) and *split-after*(v) into our pool of dynamic path operations.

B.1 Initializing the Four Arrays for G_0

For GIMR (5.1), the four arrays are initiated for G_0 as follows:

```
begin
    $sa(0) := 0$, $sa(i) := sa(i-1) - w_{i,0}$, for $i = 1, \ldots, n$;
    $p_{sa} :=$ init_dynamic_path$((sa(i))_{i=0,1,\ldots,n})$;

    $ta(i) := 0$, for $i = 0, 1, \ldots, n$;
    $p_{ta} :=$ init_dynamic_path$((ta(i))_{i=0,1,\ldots,n})$;

    $tms(0) := 0$, $tms(i) := ta(i) - sa(i) + d_{i,i+1}$, for $i = 1, \ldots, n-1$, $tms(n) := ta(n) - sa(n)$;
    $p_{tms} :=$ init_dynamic_path$((tms(i))_{i=0,1,\ldots,n})$;

    $smt(0) := 0$, $smt(i) := sa(i) - ta(i) + d_{i+1,i}$, for $i = 1, \ldots, n-1$; $smt(n) := sa(n) - ta(n)$;
    $p_{smt} :=$ init_dynamic_path$((smt(i))_{i=0,1,\ldots,n})$;
```

end

Note that for ℓ_1-GIMR (5.10), $w_{i,0} = -w_i$. The conversion of an ordinary array into a dynamic path is achieved in subroutine $p_{array} := \mathsf{init\_dynamic\_path}((array(i))_{i=0,1,\dots,n})$, which takes as argument an array $(array(i))_{i=0,1,\dots,n}$ and returns a dynamic path p_{array} constructed from the array.

Let $(array(i))_{i=0,1,\dots,n}$ be any of the four arrays of $n+1$ elements. We can construct a dynamic path p_{array} of $n+2$ vertices, from vertex v_0^{array} to v_{n+1}^{array}, such that the cost of edge $(v_i^{array}, v_{i+1}^{array})$ is $array(i)$ for $i = 0, 1, \dots, n$ (See Figure B.1).

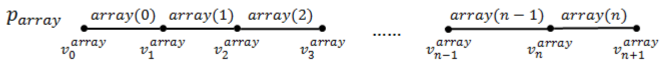


Figure B.1: p_{array} is a dynamic path constructed from array $(array(i))_{i=0,1,\dots,n}$. In p_{array}, we designate vertex v_0^{array} as head and vertex v_{n+1}^{array} as tail. p_{array} is implemented as a single full balanced binary tree of $n+1$ non-leaf nodes and $n+2$ leaf nodes.

We use the *concatenate* operation to construct p_{array} from array $(array(i))_{i=0,1,\dots,n}$. The pseudo-code is the following:

```
$p_{array} := \mathsf{init\_dynamic\_path}((array(i))_{i=0,1,\dots,n})$
begin
    Initialize dynamic path $p_{array}$ of a single vertex $v_0^{array}$, i.e., $p_{array} := [v_0^{array}, v_0^{array}]$;
    for $i := 0, \dots, n$:
        Create dynamic path $q$ of a single vertex $v_{i+1}^{array}$, i.e., $q := [v_{i+1}^{array}, v_{i+1}^{array}]$;
        $p_{array} := concatenate(p_{array}, q, array(i))$;
    end for
    return $p_{array}$;
end
```

It takes $O(1)$ time to create a single vertex dynamic path [97], therefore the complexity of the subroutine is $O(n \log n)$.

Note that by the definition of the *concatenate* operation, p_{array} has vertex v_0^{array} as head and vertex v_{n+1}^{array} as tail. Recall that $pcost(v)$ returns the cost of edge $(v, after(v))$, hence $array(i)$ is accessed by calling $pcost(v_i^{array})$.

B.2 Pseudo-code of Subroutine update_arrays$(i_k, w_{i_k,j_k-1}, w_{i_k,j_k})$

We define subroutine update_constant(p_{array}, i_k, w) that add a constant value w to the subpath of dynamic path p_{array} that corresponds to the subarray from $array(i_k)$ to $array(n)$.

```
update_constant($p_{array}, i_k, w$)
begin
    $[p, q, x] := split\text{-}before(v_{i_k}^{array})$; {$q$ is the subpath corresponding to the subarray $array(i_k)$ to $array(n)$}
    $pupdate(q, w)$; {$\forall j \in [i_k, n] : array(j) := array(j) + w$}
    if $p \neq NIL$ then $p_{array} := concatenate(p, q, x)$; else $p_{array} := q$; end if {Merge the split vertex-disjoint paths $p$ and $q$ back to a single dynamic path $p_{array}$ corresponding to the whole array $(array(i))_{i=0,1,\ldots,n}$}
end
```

The complexity of each call to update_constant is $O(\log n)$.

With update_constant, update_arrays is implemented as follows:

```
update_arrays($i_k, w_{i_k,j_k-1}, w_{i_k,j_k}$)
begin
    if $w_{i_k,j_k-1} \leq 0$ and $w_{i_k,j_k} \leq 0$ then
        update_constant($p_{sa}, i_k, -(w_{i_k,j_k} - w_{i_k,j_k-1})$);
        update_constant($p_{tms}, i_k, w_{i_k,j_k} - w_{i_k,j_k-1}$);
        update_constant($p_{smt}, i_k, -(w_{i_k,j_k} - w_{i_k,j_k-1})$);
    else if $w_{i_k,j_k-1} \leq 0$ and $w_{i_k,j_k} \geq 0$ then
        update_constant($p_{sa}, i_k, w_{i_k,j_k-1}$);
        update_constant($p_{ta}, i_k, w_{i_k,j_k}$);
        update_constant($p_{tms}, i_k, w_{i_k,j_k} - w_{i_k,j_k-1}$);
        update_constant($p_{smt}, i_k, -(w_{i_k,j_k} - w_{i_k,j_k-1})$);
    else if $w_{i_k,j_k-1} \geq 0$ and $w_{i_k,j_k} \geq 0$ then
        update_constant($p_{ta}, i_k, w_{i_k,j_k} - w_{i_k,j_k-1}$);
        update_constant($p_{tms}, i_k, w_{i_k,j_k} - w_{i_k,j_k-1}$);
        update_constant($p_{smt}, i_k, -(w_{i_k,j_k} - w_{i_k,j_k-1})$);
        end if
end
```

As update_arrays makes constant number of calls to update_constant, hence the complexity of update_arrays is $O(\log n)$.

For the special case of ℓ_1-GIMR (5.10), update_arrays is called with argument $w_{i_k,j_k-1} = -w_{i_k}$ and $w_{i_k,j_k} = w_{i_k}$.

B.3 Pseudo-code of Subroutine $[i^*_{k1}, i^*_{k2}] :=$ **find_status_change_interval**$(i_{k\ell}, i_k, i_{kr})$

The following pseudo-code operates on the dynamic paths of the four arrays:

```
$[i^*_{k1}, i^*_{k2}] :=$ find_status_change_interval$(i_{k\ell}, i_k, i_{kr})$
begin
    {Identify $\hat{i}_{k1}$}
    if $i_{k\ell} = i_k$ then $\hat{i}_{k1} := i_{k\ell}$, $f_1(\hat{i}_{k1}) = pcost(v^{sa}_{i_k}) - pcost(v^{sa}_{i_{k\ell}-1})$;
    else
        {Identify $\tilde{i}_{k1} \in [i_{k\ell}+1, i_k]$}
        $[p_1, p_2, x] := split\text{-}before(v^{tms}_{i_{k\ell}})$; {Path $p_{tms}$ is split into path $p_1$ and path $p_2$}
        $[q_1, q_2, y] := split\text{-}after(v^{tms}_{i_k})$; {Path $p_2$ is split into path $q_1$ and path $q_2$. Path $q_1$
        corresponds to the subarray $tms(i_{k\ell})$ to $tms(i_k - 1)$}
        $\tilde{i}_{k1} := pmincost(q_1) + 1$;
        {Recover a single dynamic path $p_{tms}$ for $(tms(i))_{i=0,1,\dots,n}$}
        if $q_2 \neq NIL$ then $p_{tms} := concatenate(q_1, q_2, y)$; else $p_{tms} := q_1$; end if
        if $p_1 \neq NIL$ then $p_{tms} := concatenate(p_1, p_{tms}, x)$; end if
        $f_1(\tilde{i}_{k1}) := pcost(v^{tms}_{\tilde{i}_{k1}-1}) - pcost(v^{ta}_{i_{k\ell}-1}) + pcost(v^{sa}_{i_k}) + d_{i_{k\ell},i_{k\ell}-1}$;
        $f_1(i_{k\ell}) := pcost(v^{sa}_{i_k}) - pcost(v^{sa}_{i_{k\ell}-1})$;
        if $f_1(\tilde{i}_{k1}) \leq f_1(i_{k\ell})$ then $\hat{i}_{k1} := \tilde{i}_{k1}$, $f_1(\hat{i}_{k1}) := f_1(\tilde{i}_{k1})$; else $\hat{i}_{k1} := i_{k\ell}$, $f_1(\hat{i}_{k1}) :=$
        $f_1(i_{k\ell})$; end if
    end if
    {Identify $\hat{i}_{k2}$}
    if $i_{kr} = i_k$ then $\hat{i}_{k2} := i_{kr}$; $f_2(\hat{i}_{k2}) := pcost(v^{sa}_{i_{kr}}) - pcost(v^{sa}_{i_k})$;
    else
        {Identify $\tilde{i}_{k2} \in [i_k, i_{kr} - 1]$}
        $[p_1, p_2, x] := split\text{-}before(v^{smt}_{i_k})$; {Path $p_{smt}$ is split into path $p_1$ and path $p_2$}
        $[q_1, q_2, y] := split\text{-}after(v^{smt}_{i_{kr}})$; {Path $p_2$ is split into path $q_1$ and path $q_2$. Path $q_1$
        corresponds to the subarray $smt(i_k)$ to $smt(i_{kr}-1)\}$
        $reverse(q_1)$; {Make $v^{smt}_{i_k}$ as the tail and $v^{smt}_{i_{kr}}$ as the head}
        $\tilde{i}_{k2} := pmincost(q_1)$;
        $reverse(q_1)$; {Resume $v^{smt}_{i_k}$ as the head and $v^{smt}_{i_{kr}}$ as the tail}
        {Recover a single dynamic path $p_{smt}$ for $(smt(i))_{i=0,1,\dots,n}$}
        **if** $q_2 \neq NIL$ **then** $p_{smt} := concatenate(q_1, q_2, y)$; **else** $p_{smt} := q_1$; **end if**
        **if** $p_1 \neq NIL$ **then** $p_{smt} := concatenate(p_1, p_{smt}, x)$; **end if**
        $f_2(\tilde{i}_{k2}) := pcost(v^{smt}_{\tilde{i}_{k2}}) - pcost(v^{sa}_{i_k}) + pcost(v^{ta}_{i_{kr}}) + d_{i_{kr},i_{kr}+1}$;
        $f_2(i_{kr}) := pcost(v^{sa}_{i_{kr}}) - pcost(v^{sa}_{i_k})$;
        **if** $f_2(\tilde{i}_{k2}) \leq f_2(i_{kr})$ **then** $\hat{i}_{k2} := \tilde{i}_{k2}$, $f_2(\hat{i}_{k2}) := f_2(\tilde{i}_{k2})$; **else** $\hat{i}_{k2} := i_{kr}$, $f_2(\hat{i}_{k2}) :=$
        $f_2(i_{kr})$; **end if**
    **end if**
    $Z([\hat{i}_{k1}, \hat{i}_{k2}]) := f_1(\hat{i}_{k1}) + f_2(\hat{i}_{k2})$;
    $Z(\emptyset) := pcost(v^{ta}_{i_{kr}}) - pcost(v^{ta}_{i_{k\ell}-1}) + d_{i_{k\ell},i_{k\ell}-1} + d_{i_{kr},i_{kr}+1}$;
    **if** $Z(\emptyset) \leq Z([\hat{i}_{k1}, \hat{i}_{k2}])$ **then** $[i^*_{k1}, i^*_{k2}] := \emptyset$; **else** $[i^*_{k1}, i^*_{k2}] := [\hat{i}_{k1}, \hat{i}_{k2}]$; **end if**
    **return** $[i^*_{k1}, i^*_{k2}]$;
**end**
```

The number of calls to the dynamic path operations is constant. More precisely, there are 16 calls to *pcost*, 2 calls to *split-before*, 2 calls to *split-after*, 2 calls to *pmincost*, 4 calls to *concatenate* and 2 calls to *reverse*. Therefore the complexity of find_status_change_interval is $O(\log n)$.

Appendix C

Benchmark Images

Figure C.1 contains all the twenty benchmark images we use in the experiment. We sequentially name them from Image 1 to Image 20.

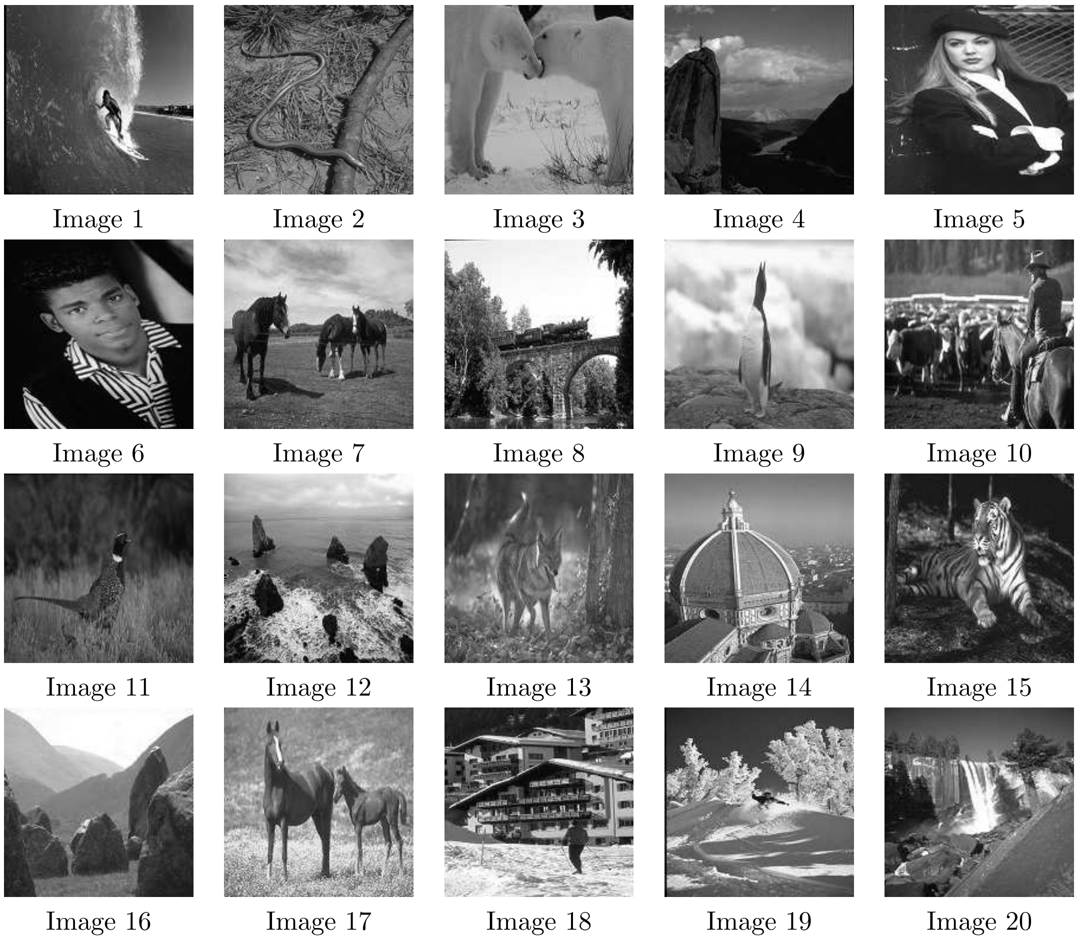


Figure C.1: The twenty benchmark images