Two Simple Algorithmic Applications of Convex Optimization

Owen Li

CMU-CS-25-117

May 2025

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Barnabas Poczos, Chair
Jason Li

Submitted in partial fulfillment of the requirements for the degree of Master of Science

Keywords: Algorithm, Optimization, Max-Flow, Min-Cut, Approximation, Fair-Cuts, Convexity, Relaxation, GAMs.

Abstract

Methods and concepts from convex optimization have close relations to traditionally discrete algorithms, and we investigate two separate cases of such. In part I, we leverage a variant of the Mirror-Prox algorithm from Sherman's 2017 paper on area-convexity and multicommodity flow, to design a fast $\tilde{O}(m/\epsilon)$ algorithm for ϵ-fair cuts, a special type of approximate st-min-cut that requires some st-flow to $1-\epsilon$ saturate all edges across the cut. Such runtime is an improvement over the state-of-the-art $\tilde{O}(m/\epsilon^3)$, and the resulting algorithm is much simpler. In part II, we design a continuous counter part of the Graham Scan convex hull algorithm that computes the tight convex envelope of degree-n univariate polynomials in $O(n^3 + n\log^2 n \log b + nb^2)$ with respect to an interval domain, and updates in $O(nb^2)$ with respect to a new interval domain, where 2^{-b} is the relative precision for float point arithmetic. Such algorithm relies on properties of convex functions for its proof of correctness, and can be used to construct high quality convex relaxations for Generalized Additive Models (GAM) with monotone link functions.

Acknowledgments

This thesis was completed during my first year of serious research experience, and originally emerged as an informal side project called "Adaptive Optimization (ADOPT)", which aimed to speed up deterministic global optimization by preconditioning the optimization problems via domain-specific heuristics. While the project evolved in completely unexpected ways and deviated from the original goal, I feel extremely lucky to have received help from numerous professors, scholars, and peers along my journey.

In particular, I appreciate the generous help from Dr. Jason Li and from Dr. Barnabas Poczos on my research. Dr. Jason Li advised me on the research project for improving the runtime of his great invention, the *fair-cut*, which had broad applications in speeding up graph algorithms; his hands-on approach and patience in mentorship helped me learn how to conduct theoretical research, from brainstorming ideas to organizing proofs. Our research was eventually accepted to IPCO 2025 and became Chapter 2 of this thesis. Dr. Poczos appreciated my initiative and agreed to be my advisor for a Master's Thesis; he encouraged me to think about practical applications and helped me plan the project from a broad perspective, which was very helpful for my research progress, and the project eventually led to Chapter 3 of this thesis.

Besides my awesome advisors, several peers and graduate scholars also helped me greatly during my research. In particular, my friend and housemate Thomas Liao played a central role by introducing me to the wonderful world of optimization; Thomas invited me to participate in the aforementioned ADOPT project, taught me how to collect & read academic articles during a summer, and inspired me to pursue graduate school in Mathematical Optimization; Chemical Engineering PhD Candidate Daniel Ovalle, whom we encountered during class, significantly helped my research project (specifically Chapter 3) by teaching me how to use Pyomo and how to structure research papers. Although they are not my formal academic advisors, their roles are equally important to my achievement.

Apart from my direct academic collaborators, I thank all 7 of my housemates for their assistance. They provided useful suggestions on research and graduate school application, supported me through my stressful months via free food and cheerful words, and further encouraged me to pursue a graduate degree in my area of interest. In addition, my housemates' cats, Primal and Dual, offered me companionship and inspiration during my research and application season.

Contents

Chapter 1

Introduction

This work is divided into two parts and surveys some bridges between continuous optimization and discrete algorithms.

In the first part, we present a simple and faster algorithm for computing *fair cuts* on undirected graphs, a concept introduced in recent work of Li et al. (SODA 2023). Informally, for any parameter $\epsilon > 0$, a $(1+\epsilon)$-fair (s,t)-cut is an (s,t)-cut such that there exists an (s,t)-flow that uses $1/(1+\epsilon)$ fraction of the capacity of *every* edge in the cut. Our algorithm computes a $(1+\epsilon)$-fair cut in $\tilde{O}(m/\epsilon)$ time, improving on the $\tilde{O}(m/\epsilon^3)$ time algorithm of Li et al. and matching the $\tilde{O}(m/\epsilon)$ time algorithm of Sherman (STOC 2017) for standard $(1+\epsilon)$-approximate min-cut. Our main idea is to run Sherman's approximate max-flow/min-cut algorithm iteratively on a (directed) *residual* graph. While Sherman's algorithm is originally stated for undirected graphs, we show that it provides guarantees for directed graphs that are good enough for our purposes.

In the second part, we present a continuous version of the famous convex hull algorithm, Graham Scan, for computing the tight convex relaxation of univariate polynomials, and subsequently a high quality relaxation for Generalized Additive Models (GAM) with a monotone link function and polynomial expressions. Convex relaxation methods are widely used in spatial branch-and-bound type algorithms; However, these relaxation methods do not produce tight convex relaxations of expressions in general, even for one dimensional cases, and yet tight relaxations facilitate branch pruning. We design a continuous version of the Graham Scan convex hull algorithm, which given a floating point precision of 2^{-b}, computes the tight convex envelope of a degree-n polynomial in $O(n^3 + n\log^2 n \log b + nb^2)$ time and updates it in $O(nb^2)$ with respect to any interval domain. We further show that our algorithm paired with the famous recursive convex relaxation from Garth P. McCormick yields a convex relaxation of GAM expressions that shares the same global optima. We further note that our relaxation is efficient to update as the variable scopes change during sBB, as re-computation from scratch is not needed.

These two parts highlight the interplay between convex optimization and more traditional aspects of discrete algorithms. On one hand, Sherman's work is essentially a variant of the Mirror-Prox algorithm, used to solve saddle point problems, and came in handy for approximating max flows & min cuts; On the other hand, properties of convex functions help us design a continuous version of Graham Scan, a discrete convex hull algorithm, and our algorithm yields good convex relaxations of nonliear terms like GAM, which is again useful for optimization purposes.

Chapter 2

Laplacian-based Flow Algorithm and its Application in Efficiently Computing Fair-Cuts

2.1 Background Information on Laplacian-based Flow Algorithm

Continuous optimization methods have seen many prominent uses in algorithm design, and they often serve as approximation schemes for graph algorithm that are computationally efficient.

One notable attempt to speed up the well-studied max-flow problem is due to Kelner et. al. [5], which incrementally computes the maximum st-flow by solving a sequence of electrical flow problems on a network of resistors. In particular, for a graph G with n vertices and m edges, the authors treats network edges of capacity c_e as resistors of resistance r_e, sets the electrical potentials at s and t to be 1 and 0 respectively, and minimizes the electrical potential

$$\mathcal{E}(f) := \|R^{1/2} f\|$$

subject to $Bf = \mathbb{1}_s - \mathbb{1}_t$, where $f \in \mathbb{R}^n_+$ is the network flow, $R \in \mathbb{R}^{m \times m}$ is the diagonal matrix indicating the electrical resistance on each edge, and B is the discrete divergence operator of G defined as

$$B_{v,e} = \begin{cases} 1, \text{ if } e \in E^-(v) \\ -1, \text{ if } e \in E^+(v) \\ 0, \text{ otherwise} \end{cases}$$

Note that such electrical flow has a closed-form solution

$$f^* = CB^\top (BCB^\top)^\dagger (\mathbb{1}_s - \mathbb{1}_t)$$

and the authors designed an approximate max-flow algorithm by repeatedly computing electrical flow, then removing the edges where such electrical flows violate the capacities by too much.

Another independent continuous approach is due to Sherman [29], which instead of solving for an st-max-flow, routes a general distributed demand vector $d \in \mathbb{R}^n$, such formulates the optimization problem

$$\min_f \|C^{-1}f\|_\infty \qquad \text{subject to: } Bf = d$$

and subsequently uses $\text{lmax}(x) := \log\left(\sum_i e^{x_i} + e^{-x_i}\right)$ to approximate the infinity norm $\|\cdot\|_\infty$, while converting the equality constraint into a regularization term, which results in the alternative objective:

$$\phi(f) := \text{lmax}(C^{-1}f) + \text{lmax}(2\alpha R(Bf - d))$$

where R is some α-congestion approximator of graph, as defined in Definition 3. Such objective is subsequently minimized via update rule

$$f_e \leftarrow f_e - \eta \cdot \text{sign}(\nabla\phi(f)_e)c_e$$

with step size $\eta = \|C\nabla\phi(f)\|_1/(1 + 4\alpha^2)$.

A more recent work from Sherman [30] in STOC 2017 generalizes to multi-commodity flow; it formulates maxflow as a minimax saddle point problem:

$$\min_x \max_y y^\top Ax - b^\top y - c^\top x$$

where x is the flow and y is the fractional cut (as a dual variable) and then solves the problem with a modified version of a continuous optimization algorithm named MirrorProx, first introduced by A. Nemirovski in [26]. Note that Sherman originally presented the algorithm as a numerical PDE technique, and its relationship with MirrorProx is suggested by Cohen et. al. in [6].

In subsequent sections, we present an approximate min-cut algorithm (specifically, for fair cuts, to be introduced later) that makes crucial use of Sherman's optimization routine in [30]. Specifically, we prove a modified version of Sherman's Theorem 9 that applies to residual graphs, and use it to compute ϵ-fair-cuts by iteratively update an initial st-cut. Our result shows that continuous optimization methods are applicable to approximating variant(s) of st-min-cuts.

2.2 Introduction

The (s,t)-min-cut and (s,t)-max-flow problems are among the most basic and well-studied problems in combinatorial optimization. A long line of research on fast algorithms [7, 8, 12, 22] culminated in the recent breakthrough $m^{1+o(1)}$ time algorithm of Chen et al. [4]. A separate line of research has focused on applying max-flow to solve other cut-based problems in combinatorial optimization, most notably Steiner min-cut [19], Gomory-Hu tree [1], and expander decomposition [28]. Using the algorithm of Chen et al. as a black box, all of these problems are now solvable in $m^{1+o(1)}$ time, which is optimal up to the factor $m^{o(1)}$.

On the other hand, the algorithm of Chen et al. (and subsequent improvements [31]) have a few downsides. First, the algorithms do not quite achieve "near"-linear time, which colloquially means $\tilde{O}(m)$ time where $\tilde{O}(\cdot)$ suppresses polylogarithmic factors. In fact, a near-linear time algorithm appears out of reach with the current techniques, which exploit recursion at the cost of $m^{o(1)}$ factors. Also, the algorithms are inherently sequential, leaving open the question of parallel max-flow in $m^{1+o(1)}$ work and sublinear time. These downsides carry over to any algorithm that requires max-flow as a black box, and hence to the cut-based problems mentioned above.

To address these issues, Li et al. [20] introduced the concept of *fair cuts*, a robust form of *approximate* min-cut. They present an algorithm for $(1+\epsilon)$-fair cut in $\tilde{O}(m/\epsilon^3)$ time that can be parallelized,[1] and then show how to solve $(1+\epsilon)$-approximate Steiner min-cut and Gomory-Hu tree using fair cut as a black box, leading to $\tilde{O}(m/\epsilon^{O(1)})$ time algorithms for both problems that can be parallelized. They also establish the first $\tilde{O}(m)$ time algorithm for expander decomposition that can also be parallelized.

The fair cut problem should be viewed as a generalization of $(1+\epsilon)$-approximate min-cut, which can be solved in $\tilde{O}(m/\epsilon)$ time by a recent breakthrough of Sherman [30] but is not robust enough for the above applications.[2] Nevertheless, there was a gap between the $\tilde{O}(m/\epsilon)$ time algorithm for $(1+\epsilon)$-approximate min-cut (and max-flow) and the $\tilde{O}(m/\epsilon^3)$ time algorithm for $(1+\epsilon)$-fair cut.

In this paper, we close the gap between the two problems by solving $(1+\epsilon)$-fair cut in $\tilde{O}(m/\epsilon)$ time. Conceptually, we present evidence that fair cut is no harder than approximate min-cut despite being more robust and powerful.

Theorem 1. *There is an $\tilde{O}(m/\epsilon)$ time randomized algorithm that, with high probability,[3] solves $(1+\epsilon)$-fair cut on an undirected graph with integral and polynomial capacities.*

Our algorithm is iterative, sending flow on each iteration and updating the *residual* graph, which is directed. Our main idea is observing that Sherman's approximate max-flow/min-cut algorithm (for undirected graphs) actually performs well on certain *directed* graphs, such as residual graphs of originally undirected graphs.

[1] The parallelization requires $m^{1+o(1)}$ work and $m^{o(1)}$ time, but the $m^{o(1)}$ factors can be improved to polylogarithmic by recent work [2]. For simplicity, we do not discuss parallelization in this paper.

[2] In more technical terms, the concept of *uncrossing two cuts* breaks down for arbitrary approximate min-cuts. Fair cuts are designed to satisfy an approximate version of uncrossing, which suffices for the applications.

[3] We adopt the convention that *with high probability* means with probability $1 - 1/n^{O(1)}$ for arbitrarily large polynomial in n.

2.2.1 Preliminaries

We work with both undirected and directed graphs in this paper. For an undirected graph $G = (V, E)$, let $\overleftrightarrow{G} = (V, \overleftrightarrow{E})$ be the directed version of G with each edge replaced by bidirectional arcs of the same capacity. Given a vertex set $S \subseteq V$, let ∂S in an undirected graph be the set of edges with exactly one endpoint in S, and let $\vec{\partial} S$ in a directed graph be the set of arcs whose tail is in S and whose head is not in S. We may also use $\vec{\partial} S$ for an undirected graph G, in which case we are referring to the bidirected $\overleftrightarrow{G}$.

Throughout the paper, we use $c_G(\cdot)$ to denote edge and arc capacities. For an arc/edge set F, let $c_G(F)$ denote the total capacity of arcs/edges in F. We assume that all capacities are integers and polynomially bounded; in general, we would incur extra $\log W$ terms where W is the maximum integral capacity, but we stick with polynomially bounded for simplicity.

We represent a *flow* as a nonnegative vector $f \in \mathbb{R}^{\overleftrightarrow{E}}$ for an undirected graph and $f \in \mathbb{R}^E$ for a directed graph. The *congestion* of a flow is $\max_{(u,v)\in\overleftrightarrow{E}} f(u,v)/c_G(u,v)$, where $\overleftrightarrow{E}$ is replaced by E for a directed graph. Sometimes we abuse notation and say the flow *has congestion* κ if the congestion of the flow is *at most* κ. A flow is *feasible* if its congestion is at most 1. A *demand* is a vector $d \in \mathbb{R}^V$ with $\sum_v d_v = 0$. A flow *satisfies* or *routes* demand $d \in \mathbb{R}^V$ if for each vertex $v \in V$, $\sum_{(v,w)} f(v,w) - \sum_{(u,v)} f(u,v) = d_v$, i.e., the net flow out of v is exactly d_v. A flow is an (s,t)-flow of *value* τ if it satisfies demand $\tau(\mathbb{1}_s - \mathbb{1}_t)$. Here, $\mathbb{1}_v$ is the vector with entry 1 at v and entry 0 elsewhere. We also use $\mathbb{1}$ as the all-ones vector.

Given an undirected/directed graph $G = (V, E)$ and a flow f, the *residual* graph G' of G for flow f is the directed graph with arc capacities $c_{G'}(u,v) = c_G(u,v) - f(u,v) + f(v,u)$ for each (u,v) where either $(u,v) \in E$ or $(v,u) \in E$. Here, $c_G(u,v), f(u,v)$ are zero if $(u,v) \notin E$ and likewise for (v,u).

A *cut* is a bipartition $(S, V \setminus S)$ of the vertex set where $S, V \setminus S \neq \emptyset$. It is an (s,t)-cut if $s \in S$ and $t \notin S$. For a directed graph G, the *value* of the cut $(S, V \setminus S)$ is $c_G(\vec{\partial} S)$. We require the following fact about the *submodularity* of the directed cut function: for any directed graph G and two sets $A, B \subseteq V$, $c_G(\vec{\partial} A) + c_G(\vec{\partial} B) \geq c_G(\vec{\partial}(A \cup B)) + c_G(\vec{\partial}(A \cap B))$.

We now define the object of study in this paper, a *fair cut*.

Definition 2. *Let s, t be two vertices in V. For any parameter $\alpha \geq 1$, we say that a cut $(S, V \setminus S)$ is an* α-fair (s,t)-cut *if there exists a feasible (s,t)-flow f such that $f(u,v) \geq \frac{1}{\alpha} \cdot c(u,v)$ for every arc $(u,v) \in \vec{\partial} S$.*

Definition 3. *For any* undirected *graph G, an α-congestion approximator of G is a matrix $R \in \mathbb{R}^{[r]\times V}$ (where dimension r is unspecified) such that for any demand d whose optimal flow has congestion $OPT(d)$, it holds that $\|Rd\|_\infty \leq OPT(d) \leq \alpha \|Rd\|_\infty$.*

We defer the matrix notation from Sherman's approximate max-flow/min-cut algorithm to its relevant Section 2.4.

2.3 Fair Cut Algorithm

In this section, we present our fair cut algorithm, establishing Theorem 1. It will be more convenient to prove the following version, where ϵ is replaced by $O(\epsilon \log n)$.

Theorem 4. *There is an $\tilde{O}(m/\epsilon)$ time randomized algorithm that, with high probability, solves $(1 + O(\epsilon \log n))$-fair cut on an undirected graph with integral and polynomial capacities.*

We will use the following approximate max-flow/min-cut primitive for *residual* graphs, which we present in Section 2.4.

Theorem 5. *Given an undirected graph G, a residual graph G' of G, two vertices s, t, and a parameter $\tau > 0$, there is a randomized algorithm that runs in time $\tilde{O}(m/\epsilon)$ and computes, with high probability,*

1. *Either an (s,t)-cut of value less than τ, or*
2. *A feasible flow f in G' routing a demand d such that the residual demand $\tau(\mathbb{1}_s - \mathbb{1}_t) - d$ can be routed in G with congestion ϵ.*

Equipped with this flow/cut primitive, the fair cut algorithm is quite simple. We iteratively maintain a cut $(S_i, V \setminus S_i)$ and a flow f_i that both gradually improve over time. On each iteration, we temporarily remove the edges in ∂S_i that are nearly saturated in the right direction, and then call the flow primitive on the residual graph (minus the removed edges) with a careful choice of τ. Depending on whether the flow primitive returns a flow or cut, we either update the current flow or the current cut, leaving the other unchanged.

We present the formal algorithm below.

1. Let f_1 be the empty flow and $(S_1, V \setminus S_1)$ be an arbitrary (s,t)-cut.
2. For iteration $i = 1, 2, \ldots, L = \Theta(\log n)$:
   (a) Let $\vec{U}_i \subseteq \overleftrightarrow{E}$ be all arcs $(u,v) \in \vec{\partial} S_i$ satisfying $f_i(u,v) \leq (1 - 4\epsilon) c_G(u,v)$, i.e., the "unsaturated" arcs in $\vec{\partial} S_i$.
   (b) Let $U_i \subseteq E$ be $\vec{U}_i$ with all arcs undirected (and parallel edges removed).
   (c) Let $G_i \subseteq G$ be the undirected graph $G \setminus (\partial S_i \setminus U_i)$, i.e., remove all edges in ∂S_i that are "saturated" in the right direction.
   (d) Let G_i' be the residual graph of G_i for the restricted flow $f_i|_{G_i}$, defined as the flow f_i with flow on arcs outside $\overleftrightarrow{G}_i$ removed.
   (e) Call Theorem 5 on graph G_i, its residual graph G_i', vertices s, t, and parameter $\tau = 0.5 c_{G_i'}(\vec{\partial} S_i)$.
   (f) If Theorem 5 returns a flow h:
      - Set $f_{i+1} = f_i + h$ and $S_{i+1} = S_i$, i.e., add the new flow but keep the current cut.
   (g) If Theorem 5 returns a cut $(X_i, V \setminus X_i)$:
      - Set $f_{i+1} = f_i$ and S_{i+1} as either $S_i \cup X_i$ or $S_i \cap X_i$, i.e., update the cut but keep the current flow. Of the two choices, pick the S_{i+1} minimizing $c_{G_i'}(\vec{\partial} S_{i+1})$.
3. Output the (s,t)-cut $(S_{L+1}, V \setminus S_{L+1})$.

It is clear that the algorithm makes $O(\log n)$ calls to Theorem 5 and runs in $\tilde{O}(m)$ time outside

these calls, for an overall running time of $\tilde{O}(m/\epsilon)$. For the rest of this section, we prove its correctness.

Our measure of progress is the quantity $c_{G'_i}(\vec{\partial} S_i)$, i.e., the total residual capacity of all "unsaturated" arcs in $\vec{\partial} S_i$, which we show drops by a constant factor on each iteration.

Lemma 6. *If Theorem 5 returns a flow h, then $c_{G'_{i+1}}(\vec{\partial} S_{i+1}) \leq 0.75 c_{G'_i}(\vec{\partial} S_i)$.*

Proof. Let d be the demand routed by flow h. By Theorem 5, there is a feasible flow r in G_i routing the residual demand $\tau(\mathbb{1}_s - \mathbb{1}_t) - d$ with congestion ϵ. Then, the flow $h + r$ in G'_i routes demand $\tau(\mathbb{1}_s - \mathbb{1}_t)$ (with arbitrary congestion). This flow must pass through the cut $\vec{\partial} S_i$ in G'_i, so $h + r$ sends a net flow of τ across $\vec{\partial} S_i$. Since the flow r has congestion ϵ in G_i, removing it from $h + r$ affects the net flow across $\vec{\partial} S_i$ by at most $\epsilon c_{G_i}(\partial S_i)$, so the flow h sends at least $\tau - \epsilon c_{G_i}(\partial S_i)$ across $\vec{\partial} S_i$. Each arc $(u, v) \in \vec{\partial} S_i$ in G'_i satisfies $(u, v) \in U_i$, so $f_i(u, v) \leq (1 - 4\epsilon) c_G(u, v)$ and $c_{G'_i}(u, v) \geq 4\epsilon c_G(u, v) = 4\epsilon c_{G_i}(u, v)$. Summing over all arcs $(u, v) \in \vec{\partial} S_i$ gives $c_{G'_i}(\vec{\partial} S_i) \geq 4\epsilon c_{G_i}(\partial S)$, so the flow h sends at least $\tau - 0.25 c_{G'_i}(\vec{\partial} S_i) = 0.25 c_{G'_i}(\vec{\partial} S_i)$ flow across $\vec{\partial} S_i$.

Let H be the residual graph of G_i for the restricted flow $f_{i+1}|_{G_i}$. By definition of residual graph, the quantity $c_{G_i}(\partial S_i) - c_H(\vec{\partial} S_i)$ is exactly the net flow that $f_{i+1}|_{G_i}$ sends across $\vec{\partial} S_i$. Since G'_i is the residual graph of G_i for the restricted flow $f_i|_{G_i}$, the quantity $c_{G_i}(\partial S_i) - c_{G'_i}(\vec{\partial} S_i)$ is exactly the net flow that $f_i|_{G_i}$ sends across $\vec{\partial} S_i$. Since $f_{i+1}|_{G_i} = f_i|_{G_i} + h$, the difference of quantities $(c_{G_i}(\partial S_i) - c_H(\vec{\partial} S_i)) - (c_{G_i}(\partial S_i) - c_{G'_i}(\vec{\partial} S_i))$ is exactly the net flow that h sends across $\vec{\partial} S_i$, which is at least $0.25 c_{G'_i}(\vec{\partial} S_i)$. In other words, $c_H(\vec{\partial} S_i) \leq c_{G'_i}(\vec{\partial} S_i) - 0.25 c_{G'_i}(\vec{\partial} S_i) = 0.75 c_{G'_i}(\vec{\partial} S_i)$. Any previously "saturated" arc $(u, v) \in \vec{\partial} S_i \setminus \vec{U}_i$ is still "saturated" in flow f_{i+1} (i.e., $f_{i+1}(u, v) > (1 - 4\epsilon) c_G(u, v)$) since the arc is absent from G'_i and hence carries no flow in h. Since $S_{i+1} = S_i$, we have $\partial S_i \setminus U_i \subseteq \partial S_{i+1} \setminus U_{i+1}$, which means that $G_i \supseteq G_{i+1}$. In particular, the arcs in $\vec{\partial} S_i$ present in G'_{i+1} are also present in H, and they have the same capacity since both G'_{i+1} and H are residual graphs for a restriction of f_{i+1}. We conclude that $c_{G'_{i+1}}(\vec{\partial} S_i) \leq c_H(\vec{\partial} S_i)$, and

$$c_{G'_{i+1}}(\vec{\partial} S_{i+1}) = c_{G'_{i+1}}(\vec{\partial} S_i) \leq c_H(\vec{\partial} S_i) \leq 0.75 c_{G'_i}(\vec{\partial} S_i),$$

as promised. □

Lemma 7. *If Theorem 5 returns a cut $(X_i, V \setminus X_i)$, then $c_{G'_{i+1}}(\vec{\partial} S_{i+1}) \leq 0.75 c_{G'_i}(\vec{\partial} S_i)$.*

Proof. By Theorem 5, the cut $(X_i, V \setminus X_i)$ satisfies $c_{G'_i}(\vec{\partial} X_i) \leq \tau = 0.5 c_{G'_i}(\vec{\partial} S_i)$. By submodularity of the cut function $c_{G'_i}(\vec{\partial} S)$,

$$c_{G'_i}(\vec{\partial} S_i) + c_{G'_i}(\vec{\partial} X_i) \geq c_{G'_i}(\vec{\partial}(S_i \cup X_i)) + c_{G'_i}(\vec{\partial}(S_i \cap X_i)),$$

and by the choice of S_{i+1},

$$c_{G'_i}(\vec{\partial} S_{i+1}) \leq \frac{1}{2}\big(c_{G'_i}(\vec{\partial}(S_i \cup X_i)) + c_{G'_i}(\vec{\partial}(S_i \cap X_i))\big) \leq \frac{1}{2}\big(c_{G'_i}(\vec{\partial} S_i) + c_{G'_i}(\vec{\partial} X_i)\big) \leq 0.75 c_{G'_i}(\vec{\partial} S_i).$$

We now claim that $c_{G'_{i+1}}(\vec{\partial} S_{i+1}) \leq c_{G'_i}(\vec{\partial} S_{i+1})$. Note that arcs present in both G'_i and G'_{i+1} must have the same capacity since $f_{i+1} = f_i$, so it suffices to show that the arcs in $\vec{\partial} S_{i+1}$ present

in G'_{i+1} are a subset of those present in G'_i. Any arc $(u,v) \in \vec{\partial} S_{i+1}$ present in G'_{i+1} satisfies $(u,v) \in \vec{U}_{i+1}$, so $f_i(u,v) = f_{i+1}(u,v) \le (1-4\epsilon)c_G(u,v)$. If $(u,v) \in \vec{\partial} S_i$ as well, then $(u,v) \in \vec{U}_i$ and the arc belongs to G'_i. Otherwise, if $(u,v) \notin \vec{\partial} S_i$, then there are two cases. If $S_{i+1} = S_i \cap X_i$, then since $u \in S_{i+1} \subseteq S_i$, we must have $v \in S_i$ as well. So the edge (u,v) is not in ∂S_i, which means the arc (u,v) belongs to G'_i, establishing the claim. If $S_{i+1} = S_i \cup X_i$, then since $v \in V \setminus S_{i+1} \subseteq V \setminus S_i$, we must have $u \in V \setminus S_i$ as well. So the edge (u,v) is not in ∂S_i, and the same argument follows.

Putting everything together, we conclude that $c_{G'_{i+1}}(\vec{\partial} S_{i+1}) \le c_{G'_i}(\vec{\partial} S_{i+1}) \le 0.75 c_{G'_i}(\vec{\partial} S_i)$. □

Finally, we prove the correctness of the algorithm, establishing Theorem 4.

Lemma 8. *The output $(S_{L+1}, V \setminus S_{L+1})$ is a $(1 + O(\epsilon \log n))$-fair cut with high probability.*

Proof. By Lemmas 6 and 7, we have $c_{G'_{i+1}}(\vec{\partial} S_{i+1}) \le 0.75 c_{G'_i}(\vec{\partial} S_i)$ for each iteration i. Since capacities are polynomially bounded, we start with $c_{G'_1}(\vec{\partial} S_1) \le n^{O(1)}$, so for large enough $L = \Theta(\log n)$ we have $c_{G'_{L+1}}(\vec{\partial} S_{L+1}) < 4\epsilon$ with high probability. Any arc $(u,v) \in \vec{U}_{L+1}$ belongs to $\vec{\partial} S_{L+1}$ and satisfies $c_{G'_{L+1}}(u,v) \ge 4\epsilon c_G(u,v) \ge 4\epsilon$, so no such arcs exist. In other words, $\vec{U}_{L+1} = \emptyset$, and it follows that $f_{L+1}(u,v) \ge (1-4\epsilon)c_G(u,v)$ for all arcs $(u,v) \in \vec{\partial} S_{L+1}$. To establish fairness, it remains to augment f_{L+1} to an (s,t)-flow.

By construction, $f_{L+1} = h_1 + h_2 + \cdots + h_L$, and there exist flows $r_1, \ldots, r_L$ in G of congestion ϵ such that each $h_i + r_i$ is an (s,t)-flow. In particular, the flow $f' = f_{L+1} + r_1 + \cdots + r_L$ is an (s,t)-flow. Since $r_1 + \cdots + r_L$ has congestion $O(\epsilon \log n)$, we have $|f'(u,v) - f_{L+1}(u,v)| \le O(\epsilon \log n) \cdot c_G(u,v)$ for all arcs (u,v). In particular, for each arc $(u,v) \in \vec{\partial} S_{L+1}$,

$$\begin{aligned} f'(u,v) &\ge f_{L+1}(u,v) - O(\epsilon \log n) \cdot c_G(u,v) \\ &\ge (1-4\epsilon)c_G(u,v) - O(\epsilon \log n) \cdot c_G(u,v) \\ &\ge \frac{1}{1 + O(\epsilon \log n)} c_G(u,v), \end{aligned}$$

so the (s,t)-flow f' certifies that $(S_{L+1}, V \setminus S_{L+1})$ is a $(1 + O(\epsilon \log n))$-fair cut. □

2.4 Approximate Max-Flow on Residual Graphs

In this section, we show how to apply Sherman's approximate max-flow/min-cut algorithm on directed *residual graphs* of an underlying undirected graph. The flow may not satisfy the input demand, but the leftover demand will be routable with low congestion on the *undirected* graph. Our main goal is to prove Theorem 5, restated below.

Theorem 5. *Given an undirected graph G, a residual graph G' of G, two vertices s, t, and a parameter $\tau > 0$, there is a randomized algorithm that runs in time $\tilde{O}(m/\epsilon)$ and computes, with high probability,*

1. *Either an (s,t)-cut of value less than τ, or*
2. *A feasible flow f in G' routing a demand d such that the residual demand $\tau(\mathbb{1}_s - \mathbb{1}_t) - d$ can be routed in G with congestion ϵ.*

We first introduce some preliminaries from Sherman [30]. For a matrix $A \in \mathbb{R}^{n\times m}$, define the matrix norm $\|A\|_{\infty\to\infty} = \max_{\|v\|_\infty=1} \|Av\|_\infty$, and define $\text{nnz}(A)$ as the number of nonzero entries in A. Define $\Delta_k^m \subseteq \mathbb{R}^{m\times k}$ as the set of matrices $X \in \mathbb{R}^{m\times k}$ with $X \geq \mathbf{0}$ and $\sum_j X_{ij} = 1$ for all $i \in [m]$. We now present a key subroutine from [30]:

Theorem 9 (Corollary 1.8 of [30])**.** *There is an algorithm that, given $B \in \mathbb{R}^{n\times k}$, and $A \in \mathbb{R}^{n\times m}$ with $\|A\|_{\infty\to\infty} \leq 1$, takes $\tilde{O}(k\text{nnz}(A)\epsilon^{-1})$ time and outputs either,*

(1) $X \in \Delta_k^m$ such that $AX \leq B + \epsilon R$ where $R \in \Delta_k^n$.

(2) $Y \in \mathbb{R}^{k\times n}, Y \geq \mathbf{0}$ such that $\text{tr}(Y(AX - B)) > 0$ for all $X \in \Delta_k^m$.

This result can be used to solve approximate multi-commodity flow, as indicated by Lemma 4.2 of [30] (re-stated as Lemma 10). Sherman only provided a sketch proof in the original paper. For specificity and completeness, we state and prove Lemma 11, which is (i) a refined and two-commodity flow version of Sherman's Lemma 4.2 for some residual network G' of G and some st-demand B, and (ii) equivalent to the Theorem 5 that we seek to prove in this section.

For a given directed graph $G = (V, \vec{E})$, we represent a flow f by its vector of congestions on edges, which is a nonnegative vector in $\mathbb{R}^{\vec{E}}$ where for each arc $(u,v) \in \vec{E}$, the flow f sends $f_{(u,v)}c_G(u,v)$ flow. Note that $\|f\|_\infty$ is exactly the congestion of the flow. We define $C_G \in \mathbb{R}^{\vec{E}\times\vec{E}}$ as the diagonal matrix whose entry (u,v) is the capacity of arc $(u,v) \in \vec{E}$. We define $D_G \in \mathbb{R}^{\vec{E}\times V}$ as the matrix whose row $(u,v) \in \vec{E}$ has vector $(\mathbb{1}_u - \mathbb{1}_v)^\top$, also called the discrete divergence operator which maps any vector D_GC_Gf to the demand satisfied by the flow f. If $G = (V, E)$ is an undirected graph, we treat it as a directed graph with a bi-directed edge set $\overleftrightarrow{E}$.

Given a directed graph $G = (V, E)$, we say $G' = (V, E')$ is a subgraph of G (denoted as $G' \leq G$) iff for all $(u,v) \in V \times V$, it holds that $c_{G'}(u,v) \leq c_G(u,v)$. In particular, if G' is some residual network for undirected G, then it holds that $G' \leq 2G$.

Now we present the original version and also our refined version of Lemma 4.2 of [30] side by side:

Lemma 10 (Lemma 4.2 of [30])**.** *Let R by any matrix with $\|RDC\|_{\infty\to\infty} \leq 1$. Then, there is an algorithm that takes $\tilde{O}(knnz(RDC)\epsilon^{-1})$ time and outputs either,*

(1) A feasible flow F such that $\|R(DF - B)\|_{\infty\to\infty} \leq \epsilon$

(2) A dual solution showing B is infeasible to route

Lemma 11 (Refined version of Lemma 4.2 of [30])**.** *Let G' be the residual network of some undirected graph G, and $B = \tau(\mathbb{1}_s - \mathbb{1}_t)$ is some st-demand vector with weight $\tau > 0$. Then there exists an algorithm that takes $\tilde{O}(m\epsilon^{-1})$ time and outputs either,*

(1) A feasible flow $F = [f, f_\varnothing]$ such that $\|R(D_{G'}C_{G'}F - B)\|_{\infty\to\infty} \leq \epsilon$

(2) An st-cut of value less than τ, which certifies that B is infeasible to route

where R is an $\alpha = (\log n)^{O(1)}$-congestion approximator of G.

A key observation is that Lemma 11 is the same as Theorem 5! This is because with an additional poly-logarithmic time, we can use a high-enough precision ϵ that enables use to treat α as a constant.

Note the setting of the refined Lemma 11: the congestion approximator R is for the undirected graph G, while the divergence operator $D_{G'}$ and the capacities $C_{G'}$ are for some residual graph G' of G. This distinction is important because good congestion approximators for directed graphs do not exist in general: let H be a 2-vertex graph with a pair of antiparallel edges, each with capacity c_1, c_2, and fix any demand vector d; then for any α-congestion approximator R of H, we have $\|Rd\|_\infty = \|R(-d)\|_\infty$, so that by definition $OPT(d), OPT(-d)$ both need to be within the interval $[\|Rd\|_\infty, \alpha\|Rd\|_\infty]$, which means that $\alpha \geq \frac{\max(c_1,c_2)}{\min(c_1,c_2)}$ must hold. This means either (1) the relative ratio between c_1, c_2 can be arbitrarily large, which means α can be large, making R a poor congestion approximator, or (2) H cannot be an arbitrary directed graph, which limits the direct use of Sherman's flow algorithm on residual networks.

Sherman's proof of Lemma 10 depends on Theorem 9; yet to prove Lemma 11 using Theorem 9, there are a few problems to be addressed:

(a) Let R be some α-congestion approximator of G from Theorem 4.4 from [30], then What is an upper-bound of $nnz(RD_{G'}C_{G'})$? This will affect the algorithm's complexity.

(b) Using Theorem 9 requires some matrix R' that satisfies $\|R'D_{G'}C_{G'}\|_{\infty\to\infty} \leq 1$, while also serve as a good congestion approximator for G (since otherwise Lemma 11 will not equal to Theorem 5 up to a poly-logarithmic factor). On a high level, it suffices to first compute some α-congestion approximator of R of G, then down-scale R by a constant factor to satisfy the operator-norm constraint.

(c) Theorem 9 outputs a continuous dual variable Y in case of infeasibility; it is necessary to round Y into a proper st-cut.

We use Lemma 12 to address (a), Lemma 13 to address (b), and Lemma 14 to address (c).

Lemma 12. *Given an undirected graph G and let G' be a residual graph of G, there exists an algorithm that succeeds with high probability in $\tilde{O}(m)$ time, and constructs some α-congestion approximator R of G, which satisfies*

$$nnz(RD'_GC'_G) \leq O(m \log n)$$

.

Proof. We cite Theorem 4.4 from Sherman [30] to compute an α-congestion approximator R of G with $\alpha = (\log n)^{O(1)}$ in $\tilde{O}(m)$ time with high probability, where R is specifically column sparse, i.e. each column contains $O(\log n)$ many nonzero entries. Observe that $(RD'_GC'_G)_{r,e} \neq 0$ iff the directed edge e is in the cut represented by row r of R, and since R is column sparse, there

can only be $O(\log n)$ many cuts in R that contain e; overall we obtain

$$nnz(RD'_GC'_G) \leq O(m \log n)$$

□

Lemma 13. *Given an undirected graph G, a subgraph $G' \leq \overleftrightarrow{G}$, and an α-congestion approximator R of G, it holds that $\|RD_{G'}C_{G'}\|_{\infty\to\infty} \leq 2$.*

Proof. Fix any vector $f \in \mathbb{R}^{\overleftrightarrow{E}}$ with $\|f\|_\infty = 1$, and let $d := D_{\overleftrightarrow{G}}C_{\overleftrightarrow{G}}f$. Notice that each pair of anti-parallel arcs e^+, e^- of $\overleftrightarrow{G}$ has the same capacity, and adding a constant to f_{e^+}, f_{e^-} will not change the demand routed by the flow; we perform such operations on each pair of anti-parallel arcs to obtain some f' such that $\min(f'_{e^+}, f'_{e^-}) = 0$ for all pairs of anti-parallel arcs. Now it is evident that $\|f'\|_\infty \leq 2$, and that f' corresponds to a flow f_G of G with demand d and congestion ≤ 2, so

$$\|RD_GC_Gf\|_\infty = \|Rd\|_\infty \leq OPT(d) \leq \|f_G\|_\infty = 2$$

and it follows that

$$\|RD_GC_Gf\|_\infty = \|Rd\|_\infty \leq 2$$

so

$$\|RD_GC_G\|_{\infty\to\infty} \leq 2.$$

Let $F := \{f \in \mathbb{R}^{\overleftrightarrow{E}} : \|f\|_\infty \leq 1\}$ be the set of all directed flows of congestion 1. Since $c_{G'}(u,v) \leq c_G(u,v)$ for all arcs $(u,v) \in \overleftrightarrow{E}$, we have the set inclusion

$$\{D_{G'}C_{G'}f : f \in F\} \subseteq \{D_GC_Gf : f \in F\}$$

so

$$\max_{f\in F} \|RD_{G'}C_{G'}f\|_\infty \leq \max_{f\in F} \|RD_GC_Gf\|_\infty \leq 2$$

and it follows that $\|RD_{G'}C_{G'}\|_{\infty\to\infty} \leq 2$. □

Note that Sherman's Stochastic Matrix Algorithm, as per Corollary 1.8 of [30], may output a dual, and when leveraging Sherman's algorithm to solve approximate (s,t)-max-flow, we may need to explicitly compute a corresponding (s,t)-cut that is integral. The following lemma constructs such a cut in a directed graph, given an infeasibility criterion from Sherman's algorithm:

Lemma 14. *Given a directed graph G and a "potential" vector $\phi \in \mathbb{R}^n$ on vertices, we define a corresponding flow f_ϕ via the following:*

$$(f_\phi)_{uv} = \begin{cases} 1, \text{ if } \phi_u > \phi_v \\ 0, \text{ otherwise} \end{cases}$$

We also suppose that

$$\phi^\top(d - D_GC_Gf_\phi) > 0$$

for some demand d. Then if we sort the vertices by decreasing the potential ϕ_v, there must be some prefix $S \subset V$, such that $\sum_{v \in S} d_v > c_G(\vec{\partial} S)$, certifying the infeasibility of such a demand. Furthermore, the cut $(S, V \setminus S)$ can be computed in $O(m + n \log n)$ time.

Moreover, if $d = \tau(\mathbb{1}_s - \mathbb{1}_v)$ for two vertices $s, t \in V$ and parameter $\tau > 0$, then the cut $(S, V \setminus S)$ is an (s,t)-cut with $c_G(\vec{\partial} S) < \tau$.

Proof. We begin with some notation. Let $V_{>x} := \{v \in V(G) \ : \ \phi_v > x\}$ denote the set of vertices of G whose potential is strictly greater than x, and let $\Delta(S) := \sum_{v \in S} d_v$ denote the sum of demands in the set S of vertices.

Let M be some positive real number such that $|\phi_v| < M$ for all $v \in V(G)$. We seek to prove

$$\int_{-M}^{M} \Delta(V_{>x}) dx = \phi^\top d > \phi^\top (B_G C_G f_\phi) = \int_{-M}^{M} c_G(\vec{\partial} V_{>x}) dx$$

because then there must be some $-M \leq x \leq M$ s.t. $\Delta(V_{>x}) > c_G(\vec{\partial} V_{>x})$, and setting $S = V_{>x}$ achieves the desired $\Delta(S) > c_G(\vec{\partial} S)$.

1. For the first equality, it holds that

$$\int_{-M}^{M} \Delta(V_{>x}) dx = \int_{-M}^{M} \sum_{v \in V} d_v \cdot \mathbb{1}[\phi_v > x] dx = \sum_{v \in V} \int_{-M}^{M} d_v \cdot \mathbb{1}[\phi_v > x] dx = \sum_{v \in V} d_v (\phi_v + M)$$

   and since d is a demand, it holds that $\sum_v d_v = 0$, so the above equals

$$\sum_{v \in V} d_v \phi_v = \phi^\top d$$

2. For the second equality, we start from the definition of f_ϕ and have

$$\begin{aligned}
\phi^\top (B_G C_G f_\phi) &= \sum_v \phi_v \cdot (- \sum_{(u,v) \in E} c_G(u,v) \mathbb{1}[\phi_u > \phi_v] + \sum_{(v,w) \in E} c_G(v,w) \mathbb{1}[\phi_v > \phi_w]) \\
&= \sum_{(u,v) \in E} c_G(u,v) \mathbb{1}(\phi_u > \phi_v)(\phi_u - \phi_v) \\
&= \sum_{(u,v) \in E} c_G(u,v) \max(0, \phi_u - \phi_v) \\
&= \int_{-M}^{M} \sum_{(u,v) \in E} c_G(u,v) \mathbb{1}((u,v) \in \vec{\partial}(V_{>x})) dx \\
&= \int_{-M}^{M} c_G(\vec{\partial} V_{>x}) dx
\end{aligned}$$

3. The inequality follows from the assumption $\phi^\top (d - B_G C_G f_\phi) > 0$.

Furthermore, if $d = \tau(\mathbb{1}_s - \mathbb{1}_t)$, then notice that

$$\Delta(S) > c_G(\vec{\partial}S) \geq 0$$

and the only way for $\Delta(S) > 0$ is if $s \in S$ and $t \notin S$, in which case $\Delta(S) = \tau$, so $(S, V \setminus S)$ is an (s,t)-cut with $c_G(\vec{\partial}S) < \tau$.

To identify the set $S = V_{>x}$, it suffices to iterate through the cuts corresponding to all $O(n)$ prefixes $S = V_{>x}$, keeping track of $\Delta(S)$ and $c_G(\vec{\partial}S)$. We may start with $|S| = 1$, which includes the vertex v with the highest potential ϕ_v, identify edges in the cut, and compute the value of $c_G(\vec{\partial}S)$; then we keep adding vertices to S, one at a time, in the order of decreasing potentials, and for each added vertex, we re-compute the set of edges in the new cut, then compute the associated capacity. Each update after adding some vertex v requires iterating through $\deg(v)$ many edges, and since $\sum_v \deg(v) = 2m$, the overall time complexity to find the minimum threshold cut is $O(m)$ (after the initial sorting by ϕ_v in $O(n \log n)$). □

Using Lemmas 13 and 14, we finally prove Theorem 5:

Proof. To convert a single commodity flow into a right stochastic matrix problem, we consider the "empty demand" as another type of commodity. Specifically, we fix $k = 2$ and $X = [f, f_\emptyset]$, where f is the vector of a flow of congestion 1, with its entries indicating congestion on edges, and $f_\emptyset$ is the vector indicating remaining congestion, defined as $f_\emptyset := \mathbb{1} - f$; we also define $A := D_{G'}C_{G'}$, and $B := [d, d_\emptyset]$, where $d_\emptyset := D_{G'}C_{G'}\mathbb{1} - d$ is the empty demand vector. Then $AX = B$ encodes a solution for the single commodity problem.

Next, we cite Theorem 4.4 from Sherman [30] to compute an α-congestion approximator R of G with $\alpha = (\log n)^{O(1)}$ in $\tilde{O}(m)$ time with high probability; apply Lemma 12 and we obtain

$$nnz(RD_{G'}C_{G'}) \leq O(m \log n)$$

Since G' is a residual graph of G, we have $G' \leq 2G$, and applying Lemma 13 on subgraph $G'/2$ (and then scaling up by factor 2) gives $\|RD_{G'}C_{G'}\|_{\infty\to\infty} \leq 4$. Now define $R' := R/4$ and $A_2 := \begin{bmatrix} R'A \\ -R'A \end{bmatrix}$ and $B_2 := \begin{bmatrix} R'B \\ -R'B \end{bmatrix}$, so that

$$\|A_2\|_{\infty\to\infty} = \|R'A\|_{\infty\to\infty} = \|RA/4\|_{\infty\to\infty} = \|RD_{G'}C_{G'}/4\|_{\infty\to\infty} \leq 1$$

and we invoke Theorem 9 to obtain, in time $\tilde{O}(k \operatorname{nnz}(A_2)\epsilon^{-1}) \leq \tilde{O}(k \operatorname{nnz}(RD_{G'}C_{G'})\epsilon^{-1}) \leq \tilde{O}(m\epsilon^{-1})$, either

1. some X and some $S_1, S_2 \in \Delta_2^n$ such that

$$\begin{cases} R'AX - R'B \leq \epsilon S_1 \\ R'(-A)X - R'(-B) \leq \epsilon S_2 \end{cases}$$

2. some $Y = [y_1, y_2] > 0$ such that

$$\operatorname{tr}(Y(A_2X - B_2)) \geq 0$$

for all feasible flows $X = [f, f_\emptyset]$

In the first case, combining the two inequalities gives

$$-\epsilon S_2 \leq R'(AX - B) \leq \epsilon S_1$$

thus each row $r_i \in \mathbb{R}^2$ of $R'(AX - B)$ satisfies $\|r_i\|_\infty \leq \epsilon$, so that $\|r_i\|_1 \leq 2\epsilon$, and

$$\begin{aligned}
\|R'(Af - d)\|_{\infty\to\infty} &= \max_{\|v\|_\infty=1} \|R'(Af - d)v\|_\infty \\
&= \max_{\|v\|_\infty=1} \max_i |(R'(Af - d)v)_i| \\
&= \max_i \max_{\|v\|_\infty=1} |(R'(Af - d)v)_i| \\
&= \max_i \|(R'(Af - d))_i\|_1 \\
&\leq \max_i \|(R'(AX - B))_i\|_1 \\
&\leq 2\epsilon
\end{aligned}$$

and it follows from definition of R that if the algorithm returns some flow $X = [f, f_\emptyset]$, then the residual demand $Af - d$ can be routed with congestion $\leq O(\epsilon)$ with respect to the undirected graph G. We may retroactively set ϵ a constant factor smaller so that the congestion is $\leq \epsilon$.

In the second case, it equivalently holds for all feasible flows $X = [f, f_\emptyset]$ that

$$y_1^\top \left(\begin{bmatrix} A \\ -A \end{bmatrix} f - \begin{bmatrix} d \\ -d \end{bmatrix} \right) + y_2^\top \left(\begin{bmatrix} A \\ -A \end{bmatrix} f_\emptyset - \begin{bmatrix} d_\emptyset \\ -d_\emptyset \end{bmatrix} \right) > 0$$

By construction of duplicated rows, we have $y_i = \begin{bmatrix} w_i \\ z_i \end{bmatrix}$, and either one of

(i) $w_1^\top (Af - d) + w_2^\top (Af_\emptyset - d_\emptyset) > 0$
(ii) $z_1^\top (Af - d) + z_2^\top (Af_\emptyset - d_\emptyset) < 0$

is true. We further notice that

$$A(f + f_\emptyset) = d + d_\emptyset \iff Af - d = -(Af_\emptyset - d_\emptyset)$$

so we substitute and either one of

(i) $(w_2 - w_1)^\top (d - Af) > 0$
(ii) $(z_1 - z_2)^\top (d - Af) > 0$

is true; either of which, according to Lemma 14, implies the existence of some (s, t)-cut $(S, V \backslash S)$ with $c_G(\vec{\partial} S) < \tau$ that can be computed in $O(m + n \log n)$ time. □

Chapter 3

Continuous Graham Scan and its Application for Constructing GAM Convex Relaxations

3.1 Background Information on McCormick Envelope and sBB

Deterministic global optimization of general nonlinear programs have long been a difficult computational problem; yet such optimization problems have a variety of uses, for example chemical and process engineering [10, 18], certification [25], and power systems [21]. Since such problems are NP-hard, general purpose efficient algorithms do not exist. Despite that, researchers have designed various computational schemes, to hopefully tackle certain instances of such problems.

Perhaps the most famous family of such algorithms is the spatial Branch-and-Bound (sBB) paradigm, first seen in the work of Garth P. McCormick [24] in his 1976 Mathematical Programming paper. With a wide variety of implementations, as seen in the references above, the general structure remains the same: consider the following optimization problem

$$\min f(x = [x_1, ..., x_k]), \quad x \in S \cap B, \quad B := \prod_i [x_i^L, x_i^U]$$

where S is the feasible region specified by the constraints, and each x_i^L, x_i^U are the lower and upper bounds of variable x_i. The sBB algorithms incrementally approach the global optima by apply the following steps repeatedly:

(i) Partition the hyper-rectangle B into many smaller hyper-rectangles $B_1, B_2, ...$

(ii) For each B_i, use local optimization methods to solve for some p_i, such that $f(x^{(i)}) = p_i$ for some $x^{(i)} \in S \cap B_i$. Such p_i is an upper estimate of the global optima within partition $S \cap B_i$.

(iii) For each B_i, we compute some convex set $C_i \supset S \cap B_i$.

(iv) For each B_i, we compute some convex function g_i, such that $g_i(x) \leq f(x)$ for all $x \in C_i$.

(v) For each B_i, we use convex optimization techniques (e.g. interior point method) to find d_i,

such that $g_i(x^{(i)}) = d_i$ for some $x^{(i)} \in C_i$. Such d_i is a lower estimate of the global optima within partition $S \cap B_i$.

(vi) Compare the intervals $[p_i, d_i]$ across all i; discard a partition B_i if it provably does not contain the overall global optima (specifically, there exists some j such that $p_j < d_i$).

(vii) Repeat (i) through (vi) for each of the remaining B_i

Such routine is proven in [24] to eventually converge to the global optima as more partitions are generated. However, we note that step (iii) and step (iv) require constructing convex relaxations of functions. In particular, [24] gives a recursive routine for computing convex relaxations of factorable expressions. We use f_{cv}, f_{cc} to denote the convex under-estimator and concave over-estimator (also called convex and concave relaxations) of some function f. The recursive routine is presented as follows:

(a) $(u + v)_{cv} = u_{cv} + v_{cv}$

(b) $(uv)_{cv} = \inf(u)v_{cv} + u_{cv}\inf(v) - \inf(u)\inf(v)$

(c) $(f(u))_{cv} = e_I f(\text{mid}(f_{cv}(u), f_{cc}(u), \text{argmin}_I f))$

where f is a univariate function with known properties (which we call *intrinsic functions*), $e_I f$ is the tight convex relaxation of f within interval $I \subset \mathbb{R}$, and mid indicates point-wise medium.

McCormick's recursive envelope is in general not tight; yet we show that for a large subclass of Generalized Additive Models (GAM), McCormick's recursive envelope matches the original expression at their global minima when paired with our continuous version of Graham Scan.

3.2 Introduction

Spatial Branch-and-Bound (sBB) have seen wide uses in solving a variety of continuous and nonconvex optimization problems, particularly when an approximate global optima is desired and the problems otherwise lack specific structures to exploit. The popular approaches like [3, 14, 24] all involve partitioning the variable space, finding convex relaxations of each partition, optimizing the original and relaxed problems in each partition to obtain an interval containing the true global optima, and then pruning the provably suboptimal branches to reduce computation cost. Although being an optimization method, sBB have also seen uses for verification of certain mathematical models [25].

Traditional convex relaxation methods like the McCormick Relaxation [24], which recursively computes the convex envelope of factorable expressions, tend to produce very lose envelopes for nested expressions of even one variable, as we will see in [add section], which does not facilitate branch pruning. Although such an issue is generally unavoidable for high-dimensional functions, tight convex envelopes of univariate polynomials are relatively efficient to compute, and incorporate them into nonlinear programs with many univariate expressions may yield promising results.

One type of non-linear expression with many univariate polynomial terms is a class of nonlinear surrogate models called the Generalized Additive Mode (GAM) [15, 16], which is based on the Kolmogorov-Arnold approximation Theorem [cite]. Specifically, we consider optimizing the following objective:

$$\min_{x \in \mathbb{R}^n} \Phi \left(\sum_{i=1}^{n} \phi_i(x_i) \right) \tag{3.1}$$

where the function Φ is called a *link function* and introduces nonlinearity, while each of the ϕ_i are called *functional forms*, which can be obtained by back-fitting algorithms like mentioned in [16]. Such formulations appear in several contexts, such as the architectures of Kolmogorov-Arnold Networks (KAN) [23], and as a surrogate modeling technique [9, 11]. In particular, we focus on the setting where each ϕ_i is a univariate polynomial (e.g., a spline function), as seen in [33].

However, gaps still exist, and there are two questions:

1. While convex envelopes of univariate polynomials (and univariate functions in general) already exist, they have to be re-computed if we alter the domain of the function, as it frequently occurs in any sBB routine.
2. If we are able to construct tight convex envelopes of each univariate function in Equation (3.1), then how good of a convex envelope can we obtain?

We summarize our contributions on two points:

1. We propose‘ a continuous version of the Graham-Scan algorithm, which was first proposed in [13], with a bitangent computing subroutine that relies on root-finding, and show that it is indeed more efficient than existing methods for updating the subsequent convex envelopes as the variable scopes change during sBB.
2. We further prove that given tight convex envelopes computed by such methods, the recursive McCormick relaxation yields a convex relaxation of the entire GAM that is tight at the global optima, which the authors believe have promising implications in scenarios where

GAMs are used as components when formulating large hierarchal optimization problems, as in such cases, high quality convex envelopes of such models shall lead to tighter relaxations overall.

It came to our notice that even outside of the context of optimization, convex hull algorithms for functions (or more generally, parametric curves) do exist, and there are two popular families of approaches: the first type is to compute convex hulls of curves by solving polynomial equations, as seen in [17]; while such approach is very general, the complexity seems high at $\tilde{O}_B(n^7 + n^6\tau)$, partly due to the algebraic nature of the algorithm; and even if a numerical alternative approach exists, it is unclear how efficiently can it update the convex hull when the domain of the parametric curve (function) changes. Another type is by computing the collection of bitangents via Hough transformation [32]; while popular for computer vision purposes, this approach seem to generally require discretizations of the curves, which means achieving machine precision is impractical; and still, it is unclear if it can be modified to efficiently update the convex hull if the domain changes.

On the other hand, our setting focuses on functions instead of parametric curves, which enables us to use an expensive root finding step to pre-compute the segments of the functional forms that are convex; all subsequent updates caused by change in variable domain during sBB are easy to implement, as it only involves updating a stack-like data structure and finding unique solutions of bitangents, as opposed to the more general settings addressed in [17, 32], where multiple solutions might exist.

3.2.1 Assumption, Background Information and Notations

Assumption: Throughout the paper, we assume that real number arithmetic is precise up to b many bits; in other words, approximations can at most obtain a relative error of 2^{-b}. In other words, barring catastrophic cancellation, approximations within a precision of 2^{-b} is assumed to be numerically accurate.

Notations: We further clarify some non-standard notations:

(1) We use $\mathscr{P}_n$ to denote the collection of univariate polynomials up to degree n.

(2) Given a function $f : \mathbb{R} \to \mathbb{R}$ and a set $S \subset \mathbb{R}$, we use $f(S)$ to denote $\{f(x) \,:\, x \in S\}$, and use $\mathcal{G}_f(S)$ to denote the graph of f restricted to S, and use $\text{Epi}_f(S)$ to denote the epigraph of f restricted to S.

(3) For lists M, N and element l in algorithms, we use $M :: N$ to indicate concatenating lists M with N, and $N :: l$ to indicate appending element l to the end of list M.

(4) For a function $f : \mathbb{R} \to \mathbb{R}$, some subset $S \subset \mathbb{R}$, such that f is invertible over S, we use $f^{-1}|_S : f(S) \to S$ to denote the inverse of restriction of f over S.

(5) Given $S, T \subset \mathbb{R}$, we use $S \leq T$ to denote that $s \leq t$ for all $(s, t) \in S \times T$.

(6) Given a vector space V and a differentiable function $f : V \to \mathbb{R}$, we define its associated *Bregman Divergence* $D_f(\cdot, \cdot) : V \times V \to \mathbb{R}$ as

$$D_f(y, x) := f(y) - f(x) - \nabla f(x)^\top (y - x)$$

Intuitively, it is a measurement of distance, and can be viewed as a generalization of p-norms and KL-divergence.

In particular, we make the following remarks regarding Bregman Divergence:

Remark 15. *For a differentiable function $f : V \to \mathbb{R}$, the followings hold about Bregman Divergence:*

(i) x_0 is a root of $D_f(y, \cdot)$ iff the tangent line of f at x_0 goes through $(y,\ f(y))$. In particular, y is a root of $D_f(y, \cdot)$.

(ii) $D_f(y, \cdot)$ has no roots other than y if $f : [a, b] \to \mathbb{R}$ is strictly convex or strictly concave.

We further note that property (ii) is a strict version of Bregman Divergence's non-negativity.

3.2.2 Examples illustrating McCormick Envelope's Drawback

To illustrate that McCormick Envelopes defined in [24] may perform poorly even for univariate expressions, we present the following example:

Fix some $k \in \mathbb{N}$, and define the following function

$$f(x) := x^{2k} - x^{2k+2}$$

with $0 \leq x \leq 1$.

We proceed to compare two possible convex / concave envelopes of this function, namely McCormick Envelope versus Bernstein Inequality + Interval Arithmetic.

(1) For McCormick Envelopes, we treat all powers of x as intrinsic functions over x for maximum tightness, which leads to the pair of convex / concave envelopes

$$\begin{cases} f_{cv} = x^{2k} - 1 \\ f_{cc} = 1 - x^{2k+2} \end{cases}$$

(2) For Bernstein Inequality, we first convert f into Bernstein Basis, so that

$$f(x) = \frac{1}{(k+1)(2k+1)} B_{2k,2k+2} + \frac{1}{k+1} B_{2k+1,2k+2}$$

where $B_{i,n} := \binom{n}{i} x^i (1-x)^{n-1}$ is the standard definition of Bernstein Basis. Then we apply first derivative tests, and conclude by Extreme Value Theorem that

$$\text{argmax}_x\ B_{i,n} = \frac{i}{n}$$

which leads to an interval around each Bernstein Basis Function

$$0 \leq B_{i,n} \leq \binom{n}{i} \left(\frac{i}{n}\right)^i \left(\frac{n-i}{n}\right)^{n-i}$$
$$0 \leq B_{i,n} \leq 1$$

where the last line is obtained by expanding the binomial via definition, expanding the fractional terms, and use the fact that

$$\frac{n^n}{n!} \geq \frac{i^i}{i!} \cdot \frac{(n-i)^{n-i}}{(n-i)!}$$

which is easily obtained by term-wise comparison.

Substitute the intervals back into $f(x)$ and we obtain

$$0 \leq f(x) \leq \frac{1}{(k+1)(2k+1)} + \frac{1}{k+1} = \frac{2k+2}{(k+1)(2k+1)} = \frac{2}{2k+1}$$

so we obtain

$$\begin{cases} f_{cv} = 0 \\ f_{cc} = \frac{2}{2k+1} \end{cases}$$

Now we compare the area-between-envelopes for these two cases, and see that

$$\begin{cases} \text{Area}_{\text{McCormick}} = \displaystyle\int_0^1 (2 - x^{2k} - x^{2k+2}) dx \sim O(1) \\ \\ \text{Area}_{\text{Bernstein}} = \dfrac{2}{2k+1} \sim O\left(\dfrac{1}{k}\right) \end{cases}$$

which means that as k gets large, recursive McCormick Envelopes perform arbitrarily worse than even not-necessarily-optimal convex and concave envelopes, like the ones provided by the Bernstein inequalities.

3.3 Continuous Graham Scan for Univariate Convex Envelope

We begin with two remarks about convex envelopes of univariate functions:

Remark 16. *Let $f : [x_L, x_R] \to \mathbb{R}$ be any polynomial, and f_{cv} is its convex envelope over $[x_L, x_R]$, then there exists points $x_L = x_0, x_1, ..., x_n = x_R$, such that $f_{cv} = \sum_{i=0}^{n-1} \mathbb{1}_{[x_i, x_{i+1}]} g_i$, where $g_i = f_i$ or g_i is a affine function.*

Remark 17. *If any of the afore-mentioned g_i is affine, then it is tangent to the graph of f at 2 points.*

These remarks motivate an algorithm for computing tight convex envelopes for a univariate polynomial p; On a high level, given a polynomial p, our algorithm first computes the set of intervals on which p is convex, then computes bitangent lines across these intervals, and combines the bitangent lines via mimicking the famous Grahan Scan algorithm of [13] for convex hull of 2D points.

Definition 1. *For any degree n univariate polynomial p over the domain $\mathcal{I} := [x_L, x_U] \subset \mathbb{R}$, we define the associated Convex Intervals $CI_{\mathcal{I}}(p)$ to be the set of intervals $\{I_1, I_2, ... \subset \mathcal{I}\}$, such that $\frac{d^2}{dx^2} p(x) \geq 0$ for all $x \in I_i, I_i \in CI_{\mathcal{I}}(p)$.*

For convenience of notation, we treat degenerate intervals located at the boundary of the domain, namely $[x_L, x_L]$ and $[x_U, x_U]$, as convex intervals.

Definition 2. *Given any degree n univariate polynomial p , and two convex intervals $I_1, I_2 \in CI(p)$, we define $\texttt{bitan}_p(I_1, I_2)$ as the unique affine function $g(x)$, such that $g(I_1 \cup I_2) \leq p(I_1 \cup I_2)$, and $p - g$ has exactly one root in each of I_1 and I_2.*

Computation of convex intervals require finding all roots of a polynomial, which can be approximated to a relative error of 2^{-b} (ie. up to machine precision, according to our assumption) in $O(n^3 + n \log^2 n \log b)$ time by solving the eigen-problem for the companion matrix of p, according to Pan and Chen's work in [27].

We encode such root finding procedure as `all_real_roots`, and present the following subroutine for computing the convex intervals $CI(p)$ of p:

Algorithm 1 Convex Interval Computation

Input: $p \in \mathscr{P}_n, \mathcal{I} = [x_L, x_U] \subset \mathbb{R}$
$R \leftarrow \texttt{all\_real\_roots}(D^2 p)$
$R \leftarrow (R \cap \mathcal{I}) \cup \{a, b\}$
sort R in ascending order
$C \leftarrow []$
for adjacent pairs $(r_i, r_{i+1}) \in R$ **do**
  if $\frac{d^2}{dx^2} p(\frac{r_i + r_{i+1}}{2}) \geq 0$ **then**
    $C \leftarrow C :: [r_i, r_{i+1}]$
  end if
end for
return $[x_L, x_L] :: C :: [x_U, x_U]$

To construct the convex envelope of some degree n univariate polynomial p, we need to

compute two-point tangent lines that connect suitable pairs of intervals in $CI(p)$. We will use the following theorem, the proof of which we defer to Section 3.4 and the algorithm of which we present at Algorithm 3.

Theorem 18. *Given a univariate polynomial p, convex intervals I_l, I_r on p, and floating point precision ϵ, there exists an algorithm that takes $O(b^2)$ and computes $\texttt{bitan}_p(I_l, I_r)$ up to arithmetic precision.*

Using the above theorem and Remark 4, we mimic Graham Scan and design a general algorithm for constructing the convex envelope of any one-variable polynomial.

Theorem 19. *There exists an algorithm that, given the convex intervals of any univariate polynomial p of degree n, $O(n)$ queries to bitangent computation routine (ie. Algorithm 3) and an additional $O(n)$ time, computes the tight convex envelope of p within any interval $[x_L, x_U]$.*

In particular, given the $O(n^3 + n \log^2 n \log b)$ complexity for finding roots of a polynomial, as well as the $O(b^2)$ complexity of computing unique bitangents via Algorithm 3, the overall convex envelope algorithm takes

$$O(n^3 + n \log^2 n \log b + nb^2)$$

to run initially, and

$$O(nb^2)$$

for subsequent updates as the domain changes, as recomputation of roots are not needed.

Proof. We present the following algorithm, which is a continuous analog of the famous Graham Scan

Algorithm 2 Continuous Graham Scan

Input: $p \in \mathscr{P}_n, \mathcal{I} = [x_L, x_U] \subset \mathbb{R}$
$S \leftarrow []$
for $c_1, c_2, ...$ in $CI_{\mathcal{I}}(p)$ **do**
  $l \leftarrow \texttt{bitan}_p(c_i, c_{i+1})$
  while S not empty and $\texttt{top}(S)$ has slope greater than that of l **do**
    $\texttt{bitan}_p(c_j, c_i) := \texttt{top}(S)$
    $S \leftarrow S[:-1]$
    $l \leftarrow \texttt{bitan}_p(c_j, c_{i+1})$
  end while
  $S \leftarrow S :: l$
end for
return S

where $\texttt{bitan}_p(\cdot, \cdot)$ is defined at Definition 2.

For runtime of this algorithm, note that each left-side convex interval only enters and leaves the stack S at most once each; since there are $O(d)$ convex intervals, there are $O(d)$ stack operations; and since there is one bitangent computation per stack operations, it holds that the overall runtime is $O(d)$ queries to a bitangent algorithm, plus an additional $O(d)$.

For correctness of the algorithm, we start with a technical lemma:

Lemma 20. *When Algorithm 2 recomputes the bitangent (ie. updates the variable l) within the* `while` *loop, the new bitangent has a slope no less than the old bitangent.*

Proof. In a particular iteration of the `while` loop, we denote the convex intervals c_j, c_i, c_{i+1} as I_1, I_2, I_3. Due to their indexing, we have $I_1 \leq I_2 \leq I_3$; see the introduction section for ordered comparison between sets.

Since the algorithm did not exit the while loop, it must hold that the stack S is nonempty, and has $\texttt{bitan}_p(I_1, I_2)$ at its top, where the temporary variable l equals to some $\texttt{bitan}_p(I_2, I_3)$, and that

$$\frac{d}{dx}\texttt{bitan}_p(I_1, I_2) > \frac{d}{dx}\texttt{bitan}_p(I_2, I_3) \tag{3.2}$$

where $\frac{d}{dx}$ indicates taking the slope of the bitangents. In other words, the top of the stack S is a bitangent with a smaller slope than the bitangent below it in S. Since the newly updated bitangent is $\texttt{bitan}_p(I_1, I_3)$, it is equivalent to show

$$\frac{d}{dx}\texttt{bitan}_p(I_1, I_3) \geq \frac{d}{dx}\texttt{bitan}_p(I_2, I_3)$$

By definition of bitangents, the function $p - \texttt{bitan}_p(I_1, I_2)$ has a root in I_1 and a root in I_2; we call these roots x_1, x_2; similarly, the two roots of $p - \texttt{bitan}_p(I_2, I_3)$ are called y_2, y_3. In particular, this means $x_1 \in I_1$, $x_2, y_1 \in I_2$, and $y_2 \in I_3$. We note that

$$\texttt{bitan}_p(I_1, I_2)(y_1) \leq p(y_1) = \texttt{bitan}_p(I_2, I_3)(y_1)$$

where the inequality is from tangents being under-estimators of convex functions (in this case, p is convex over I_2) and the equality is from $\texttt{bitan}_p(I_1, I_2)$ tangent to p at x_2.

Apply Equation (3.2) to the above equation, and we obtain

$$\texttt{bitan}_p(I_1, I_2)(t) < \texttt{bitan}_p(I_2, I_3)(t) \qquad \forall t < y_1$$

In particular, since $I_1 < I_2 < I_3$, we have $x_1 \leq \{x_2, y_1\}$, and

$$\texttt{bitan}_p(I_1, I_2)(x_1) < \texttt{bitan}_p(I_2, I_3)(x_1)$$

By definition of the bitangents, we substitute the tangent-points with values of $p(\cdot)$, and have

$$p(x_1) < \texttt{bitan}_p(I_2, I_3)(x_1)$$

so that $\texttt{bitan}_p(I_2, I_3)$ is strictly above p at x_1. Now if $\texttt{bitan}_p(I_2, I_3)$ intersects p at two points within I_1, then we use Mean Value Theorem and obtain some $x_0 \in I_1$, such that $p'(x_0) = \frac{d}{dx}\texttt{bitan}_p(I_2, I_3)$; since evidently $p(x_0) \leq \texttt{bitan}_p(I_2, I_3)(x_0)$, it holds that

$$D_p(x_0, y_2) < 0$$

by applying Lemma 21 on I_1, I_3, we obtain

$$k = \frac{d}{dx}\texttt{bitan}_p(I_2, I_3) < \frac{d}{dx}\texttt{bitan}_p(I_1, I_3) = k^*$$

and we are done.

If otherwise $\texttt{bitan}_p(I_2, I_3)$ does not intersect p at two points, then let (x^*, y^*) be the two tangent points of $\texttt{bitan}_p(I_1, I_3)$; we add a large enough quadratic function centered at x^* to $p|_{I_1}$, so that $\texttt{bitan}_p(I_1, I_3), \texttt{bitan}_p(I_2, I_3)$ remains the same, but now $\texttt{bitan}_p(I_2, I_3)$ does intersect p at two points within I_1, so we conclude with the same logic as above. □

Now we proceed to prove the correctness of algorithm. In particular, the bitangents S computed by Algorithm 2 naturally yields the following convex relaxation of p:

$$C(x) := \begin{cases} \texttt{bitan}_p(c_i, c_j)(x), & \text{if } \sup c_i \le x \le \inf c_j \text{ and } \texttt{bitan}_p(c_i, c_j) \in S \\ p(x), & \text{otherwise} \end{cases}$$

We first note that the slopes of all computed bitangents in S follow a monotonically increasing order, so $C(x)$ is convex over the $\mathcal{I}$ by first order convexity; we next note that any $\texttt{bitan}_p(c_i, c_j) \in S$ indeed under-estimates p: suppose otherwise that there is some $\texttt{bitan}_p(c_i, c_j) \in S$, where $\texttt{bitan}_p(c_i, c_j)(r) > p(r)$ for some r, and c_i is on left side of c_j. Then by smoothness of p, there must be another convex interval c_k of p sandwiched between c_i, c_j. However this state is unachievable, because $\texttt{bitan}_p(c_k, c_j)$ would necessarily have more positive slope than $\texttt{bitan}_p(c_i, c_j)$, contradicting Lemma 20. It thus follows that $C(x)$ encodes some convex relaxation of $p(x)$.

For tightness of the convex relaxation, we use $c_1, ..., c_k$ to denote the convex intervals of p sorted from left to right, and let h'_t be the half space above $tx + b_t$, where

$$b_t := \max\{b \in \mathbb{R} \mid tx + b \le p(x) \forall x \in [x_L, x_R]\}$$

we further let $G := \text{Epi}_C([x_L, x_R])$ be the epigraph of the convex envelope $C(x)$ within $[x_L, x_R]$. Now it suffices to show

$$G \subseteq \bigcap_{t \in \mathbb{R}} h'_t$$

In particular, for any $t \in \mathbb{R}$, one of the following holds:

(a) when $\frac{d}{dx}\texttt{bitan}_p(c_i, c_j) \le t \le \frac{d}{dx}\texttt{bitan}_p(c_j, c_k)$ and $\texttt{bitan}_p(c_i, c_j), \texttt{bitan}_p(c_j, c_k) \in S$. Then there is some unique $u \in c_j$ where $C'(u) = t$, so $tx + b_t$ is tangent to $C(x)$ at u, and since $C(x)$ convex, we have $tx + b_t$ under-estimate $C(x)$, therefore $G \subset h'_t$.

(b) When $t < \frac{d}{dx}\texttt{bitan}_p(c_1, c_2)$ or $t > \frac{d}{dx}\texttt{bitan}_p(c_{k-1}, c_k)$, it is evident that $tx+b_t$ intersects with the graph of p at a unique point within either c_1 or c_k, which is part of graph of C, so by convexity of $C(x)$, we again have $tx + b_t$ under-estimate $C(x)$, therefore $G \subset h'_t$.

In either case, it follows that $G \subset h'_t$, so $C(x)$ is indeed the tightest convex relaxation of p.

□

3.4 Computation of Unique Bitangents for Polynomials

Similar to the exact convex hull algorithm of rational parametric curves from INRIA [17], we compute bitangent lines via solving for polynomial equations; the key idea, however, is that since the outline of our algorithm mimics that of Graham Scan, we only need to compute bi-tangents with respect to specific pairs of convex intervals; since each pair of convex intervals admit a unique bitangent, the polynomial equation shall have a unique root associated with each pair of convex intervals.

We discuss the two possible types of bitangents separately:

1. **With one fixed point**
   Given a polynomial p restricted to domain $[x_L, x_U]$, a bitangent of graph of p may intersect with one of x_L or x_U. Assume that $a = (x_L, f(x_L))$. It suffices to find some point $b = (x_b, f(x_b))$, where the tangent line of f at x_b passes through a. From Remark 15 1 item (i), we may equivalently solve for

   $$D_p(x_L, x) := p(x_L) - p(x) - p'(x)(x_L - x) = 0$$

   which is trivially obtained using bisection method.
2. **With no fixed point**
   In particular, computing the bi-tangent with respect to a pair of disjoint convex intervals I_x, I_y involves solving the polynomial equations

   $$D_p(x, y) = D_p(y, x) = 0$$

   subject to constraint $(x, y) \in I_l \times I_r$. But such equations become

   $$\begin{cases} p(y) - p(x) - p'(x)(y - x) = 0 \\ p(x) - p(y) - p'(y)(x - y) = 0 \end{cases}$$

   Rearrange and we have $(p'(x) - p'(y)) \cdot (x - y) = 0$, which means $p'(x) = p'(y)$ must hold at the true solution, since $I_x \cap I_y = \varnothing$ and therefore $x \neq y$ must hold. On the other hand, given a pair of x, y such that $p'(x) = p'(y)$, if such x, y does not equal the true solution (x^*, y^*) of the equation, then their Bregman Divergence $D_p(y, x)$ can be used to determine whether $p'(x) \leq p'(x^*)$ or otherwise.

Inspired by the above observation, we set up a simple algorithm that solves for such $p'(x^*)$ essentially using bisection method:

Algorithm 3 Bisection Algorithm for Computing Two-Point Tangent Line

$k_{\min}, k_{\max} \leftarrow p''(I_l) \cap p''(I_r)$
for $t = 1, ..., T = \lceil \log(1/\epsilon) \rceil$ **do**
  $k \leftarrow (k_{\min} + k_{\max})/2$
  $x_l, x_r \leftarrow (p')^{-1}|_{I_l}(k), (p')^{-1}|_{I_r}(k)$
  if $D_p(x_l, x_r) > 0$ **then**
    $k_{\max} \leftarrow k$
  else
    $k_{\min} \leftarrow k$
  end if
end for
return $(p')^{-1}|_{I_l}(k), (p')^{-1}|_{I_r}(k)$

To prove the correctness of this algorithm, we start with a technical lemma:

Lemma 21. *Let I_l, I_r be disjoint convex intervals of p such that $I_l < I_r$, and let x_l, x_r be the unique values in I_l, I_r such that $p'(x_l) = p'(x_r) = k$, and let $p'(x^*) = p'(y^*) = k^*$ be the unique solution to the above polynomial equation with $x^* \in I_l$, then $k - k^*$ and $D_p(x_l, x_r)$ have the same sign.*

Proof. We discuss two cases:

(1) if $k < k^*$, then by strict convexity of p in I_l, we have

$$x_l = (p')^{-1}|_{I_l}(k) < (p')^{-1}|_{I_l}(k^*) = x^*$$

We further consider two tangent lines of p, namely

$$\begin{cases} \ell_{x_l}(x) = p(x_l) + p'(x_l)(x - x_l) \\ \ell_{x^*}(x) = p(x^*) + p'(x^*)(x - x^*) = p(y^*) + p'(y^*)(x - y^*) \end{cases}$$

Both of which are tangent to p within I_l, so by convexity, we have for all $x > x^*$

$$\begin{aligned} \ell_{x^*}(x) &= p(x^*) + p'(x^*)(x - x^*) \\ &\geq \ell_{x_l}(x^*) + p'(x^*)(x - x^*) \\ &> \ell_{x_l}(x^*) + p'(x_l)(x - x^*) \\ &= \ell_{x_l}(x) \end{aligned}$$

where the first inequality is from ℓ_{x_l} being a tangent of p, the function p being convex in I_l, and that $x_l, x^* \in I_l$; the second inequality is from $p'(x^*) = k' > k = p'(x_l)$. In particular, we substitute $x = x_r$ and have

$$\ell_{x^*}(x_r) > \ell_{x_l}(x_r)$$

which holds because $x_r \in I_r \geq I_l \ni x^*$, so we expand and have

$$p(x^*) + p'(x^*)(x_r - x^*) > p(x_l) + p'(x_l)(x_r - x_l)$$

Note that $D_p(y^*, x^*) = 0$, so $p(y^*) - p(x^*) - p'(x^*)(y^* - x^*) = 0$; add this to the left hand side and we have

$$p(y^*) + p'(y^*)(x_r - y^*) > p(x_l) + p'(x_l)(x_r - x_l)$$

rearrange and we have

$$p'(x_l)(x_r - x_l) < p(y^*) + p'(y^*)(x_r - y^*) - p(x_l)$$

by convexity of p in I_l, we have

$$p(y^*) + p'(y^*)(x_r - y^*) \leq p(x_r)$$

so combined, we have

$$p'(x_l)(x_r - x_l) < p(x_r) - p(x_l)$$
$$p'(x_r)(x_l - x_r) > p(x_l) - p(x_r)$$

and it follows that $D_p(x_l, x_r) < 0$.

(2) If $k > k^*$, then a similar argument follows, and we have $D_p(x_l, x_r) > 0$.

□

Now we prove the correctness of Algorithm 3, the bisection algorithm:

Lemma 22. *The above algorithm is correct, if the unique solution (x^*, y^*) is in the interior of $I_x \times I_y$.*

Proof. Define $k^* = p'(x^*)$. Observe that

$$((p')^{-1}|_{I_l}(k^*), (p')^{-1}|_{I_r}(k^*)) = (x^*, y^*)$$

so it suffices to show that the binary search converges to a correct $k = k^*$. We discuss two cases:

(1) if $k < k^*$, then by Lemma 21, we have $D_p(x_l, x_r) < 0$
(2) If $k > k^*$, then by Lemma 21, we have $D_p(x_l, x_r) > 0$

Therefore in either case, the bisection algorithm always picks the interval that contains k^*, which completes the proof. □

Note that when such unique solution is not in the interior of $I_x \times I_y$, the problem is reduced to the "One fixed point" subcase.

Since the algorithm takes $O(\log(1/\epsilon))$ bisection rounds, each requiring root finding subroutines for computing $p^{-1}(...)|_{...}$, which also takes up to $O(\log(1/\epsilon))$, we obtain the following:

Theorem 18. *Given a univariate polynomial p, convex intervals I_l, I_r on p, and floating point precision ϵ, there exists an algorithm that takes $O(b^2)$ and computes $\texttt{bitan}_p(I_l, I_r)$ up to arithmetic precision.*

3.5 Good Convex Relaxation of Polynomial GAM

We show that given a Generalized Additive Model $M(x) = \phi\left(\sum_i p_i(x_i)\right)$ composed of many univariate polynomials p_i and a differentiable monotone link function $\phi : \mathbb{R} \to \mathbb{R}$, and given the tight convex envelopes of the composing univariate polynomials, which can be computed by Algorithm 2, the well-know recursive convex relaxation scheme due to Garth P. McCormick [24], combined with bounds-tightening at the link function, yields a convex relaxation M' of M, such that $\min_x c_M(x) = \min_x M(x)$, and $\text{argmin}_x c_M(x) = \text{argmin}_x M(x)$.

Theorem 23. *Let hyper-rectangle $B = \prod_{i=1}^d [x_i^L, x_i^U] \subset \mathbb{R}^d$ be the permissible domain of all variables x_i; We define a Generalized Additive Model $M : B \to \mathbb{R}$, such that*

$$M(x) = \Phi(P(x)) = \Phi\left(\sum_i^d p_i(x_i)\right)$$

where the link function $\Phi : \mathbb{R} \to \mathbb{R}$ is monotonic and either increasing or decreasing; for any function $f : S \to \mathbb{R}$, we further use $e_s f, E_s f$ to denote the tight convex and concave envelopes of f with respect to domain S. We further let $M' : B \to \mathbb{R}$ be the recursive convex envelope of M constructed similar to McCormick's procedure in [24], such that

(i) $P^- := \sum_i e_{[x_i^L, x_i^U]}(p_i),\ P^+ := \sum_i E_{[x_i^L, x_i^U]}(p_i)$

(ii) $I := [\inf_B P^-, \sup_B P^+]$*, which is easily computed via component-wise minimization.*

(iii) $M'(x) := e_I \Phi(P^-(x))$ *if Φ is increasing, and* $M'(x) := e_I \Phi(P^+(x))$ *otherwise*

Then the following properties hold:

(a) *M' is convex relaxation of M: specifically, M' is convex and $M'(x) \le M(x) \quad \forall x \in B$.*

(b) $\min_x M'(x) = \min_x M(x)$*, and argmin$_x M'(x)$ = argmin$_x M(x)$*

(c) *Let c_M be the recursive McCormick relaxation of M as defined in [24], except that the convex envelopes c_{p_i} of each p_i is tight, so $c_{p_i} = e_{[x_i^L, x_i^U]} p_i$, and similarly for C_{p_i}. Then $c_M(x) = M'(x)$ for all x. In essence, such procedure indeed is the same as McCormick relaxation using the best known univariate convex envelopes.*

Proof. For (a), we specifically focus on the case where Φ is monotonically increasing: for any $v, w \in \mathbb{R}^d$, we define $q_{v,w}(t) := P^-(v + tw)$; intuitively, $q_{v,w}$ is the restriction of P^- onto a one-dimensional affine subset of $\mathbb{R}^d$, parameterized by t; since P^- is a sum of convex functions and therefore convex, we have $q_{v,w}$ convex. Since both $e_I\Phi$ and $q_{v,w}$ are convex, and e_I non-decreasing, we have $e_I\Phi \circ q_{v,w}$ convex in t. Since v, w arbitrary, it follows from zeroth order convexity that $M' := e_I\Phi \circ P^-$ is convex in B. Furthermore, for any $x \in B$, we have

$$P^-(x) = \sum_i e_{[x_i^L, x_i^U]} p_i(x_i) \le \sum_i p_i(x_i) = P(x)$$

where the inequality follows from definition of convex envelope, so

$$M'(x) = e_I\Phi(P^-(x)) \le \Phi(P^-(x)) \le \Phi(P(x))$$

where the first inequality is due to $e_I\Phi$ being the tight convex envelope of Φ within I, which is within the possible range of values of P^-; and the second inequality is due to $P^- \le P$ and that Φ is monotonically increasing.

In case where Φ monotonically decreases, we follow an analogous proof. For convexity, we perform a restriction and notice that $f \circ g$ is convex if f is convex and decreasing, and g is concave. For underestimation, we notice that $P \leq P^+$ and use the fact that Φ is monotone decreasing to derive

$$M'(x) = e_I \Phi(P^+(x)) \leq \Phi(P^+(x)) \leq \Phi(P(x))$$

□

For (b), assume without loss of generality that Φ is monotonically increasing, then note that $e_I\Phi$ is also monotonically increasing: $e_I\Phi$ is a piecewise function made up of sections of Φ and bitangent lines to the graph of Φ, so $e_I\Phi$ is differentiable; if $e_I\Phi'(x) < 0$ for some x, then either by definition or by Mean Value Theorem, we can find some z such that $\Phi'(z) < 0$, which is a contradiction. Now since $e_I\Phi$ monotonically increasing, we have

$$\begin{aligned}\text{argmin}_{x\in B} M'(x) &= \text{argmin}_{x\in B} P^-(x) = [\text{argmin}_{x_i \in [x_i^L, x_i^U]} \ e_{[x_i^L, x_i^U]} p_i(x_i)]^\top \\ &= [\text{argmin}_{x_i \in [x_i^L, x_i^U]} \ p_i(x_i)]^\top = \text{argmin}_{x\in B} P(x) = \text{argmin}_{x\in B} M(x)\end{aligned}$$

where the first equality is by monotonicity of $e_I\Phi$, the second and fourth equality is by the fact that each component of the summation is independent, the third equality is by definition of tight convex envelope, and the last equality is by monotonicity of Φ.

Now fix $x^* := \text{argmin}_{x\in B} M(x)$. Then by definition of tight convex envelope, we have

$$P^-(x^*) = \sum_i e_{[x_i^L, x_i^U]} p_i(x_i^*) = \sum_i p_i(x_i^*) = P(x^*)$$

so we have $P^-(x^*) = P(x^*) = \inf_B P^-$; due to the monotonicity of Φ, we have $e_I\Phi(\inf_B P^-) = \Phi(\inf_B P^-)$, so

$$\min_{x\in B} M'(x) = M'(x^*) = e_I\Phi(P^-(x^*)) = \Phi(P(x^*)) = M(x^*) = \min_{x\in B} M(x)$$

and we are done with the case for Φ being monotonically increasing; for the other case, we follow a similar proof, with the only difference being taking the argmax of P^+ instead. □

For (c), we note that P^-, P^+ are exactly the McCormick envelopes of P, so it suffices to consider the relaxation at Φ. If Φ is monotonically increasing, then for any $x \in B$, we have

$$e_I\Phi(\text{mid}(c_P(x), C_P(x), z_{\min}^\Phi)) = e_I\Phi(\text{mid}(P^-(x), P^+(x), \inf_B P^-)) = e_I\Phi(P^-(x)) = M'(x)$$

where the first inequality is due to Φ being monotonically increasing and therefore minimizes at $\inf_B P^-$; the second equality is by definition. Note that the left hand side of the equality is precisely the McCormick's convex envelope of $\Phi(P(x))$.

If otherwise Φ is monotonically decreasing, then we have

$$e_I\Phi(\text{mid}(c_P(x), C_P(x), z_{\min}^\Phi)) = e_I\Phi(\text{mid}(P^-(x), P^+(x), \sup_B P^+)) = e_I\Phi(P^+(x)) = M'(x)$$

where the first inequality is due to Φ being monotonically decreasing and therefore maximizes at $\sup P^+$. Note that the left hand side of the equality is once again precisely the McCormick's convex envelope of $\Phi(P(x))$, so we are done.

□

Bibliography

[1] Amir Abboud, Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. All-pairs max-flow is no harder than single-pair max-flow: Gomory-hu trees in almost-linear time. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2204–2212. IEEE, 2023.

[2] Arpit Agarwal, Sanjeev Khanna, Huan Li, Prathamesh Patil, Chen Wang, Nathan White, and Peilin Zhong. Parallel approximate maximum flows in near-linear work and polylogarithmic depth. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3997–4061. SIAM, 2024.

[3] Ioannis P Androulakis, Costas D Maranas, and Christodoulos A Floudas. αbb: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7:337–363, 1995.

[4] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. March 2022.

[5] Paul Christiano, Jonathan A Kelner, Aleksander Madry, Daniel A Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 273–282, 2011.

[6] Michael B Cohen, Aaron Sidford, and Kevin Tian. Relative lipschitzness in extragradient methods and a direct recipe for acceleration. *arXiv preprint arXiv:2011.06572*, 2020.

[7] Yefim Dinitz. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Doklady*, 11:1227–1280, 1970.

[8] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

[9] Tracey S Frescino, Thomas C Edwards Jr, and Gretchen G Moisen. Modeling spatially explicit forest structural attributes using generalized additive models. *Journal of vegetation science*, 12(1):15–26, 2001.

[10] Chao-Yang Gau and Mark A Stadtherr. Parallel branch-and-bound for chemical engineering applications: Load balancing and scheduling issues. In *VECPAR*, pages 273–300, 2000.

[11] Jason N Goetz, Richard H Guthrie, and Alexander Brenning. Integrating physical and empirical landslide susceptibility models using generalized additive models. *Geomorphology*,

129(3-4):376–386, 2011.

[12] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45 (5):783–797, 1998. doi: 10.1145/290179.290181. URL http://doi.acm.org/10.1145/290179.290181.

[13] Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Info. Proc. Lett.*, 1:132–133, 1972.

[14] Yannis A Guzman, MM Faruque Hasan, and Christodoulos A Floudas. Performance of convex underestimators in a branch-and-bound framework. *Optimization Letters*, 10(2): 283–308, 2016.

[15] Trevor Hastie and Robert Tibshirani. Generalized additive models. *Statistical science*, 1 (3):297–310, 1986.

[16] Trevor J Hastie. Generalized additive models. *Statistical models in S*, pages 249–307, 2017.

[17] Christina Katsamaki, Fabrice Rouillier, and Elias Tsigaridas. Exact convex hull computation for plane and space parametric curves. 2023.

[18] Kwok-Fu Lee, AH Masso, and DF Rudd. Branch and bound synthesis of integrated process designs. *Industrial & Engineering Chemistry Fundamentals*, 9(1):48–58, 1970.

[19] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE Computer Society, 2020.

[20] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, and Thatchaphol Saranurak. Near-linear time approximations for cut problems via fair cuts. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 240–275. SIAM, 2023.

[21] Pengxiang Liu, Zhi Wu, Wei Gu, and Yuping Lu. An improved spatial branch-and-bound algorithm for non-convex optimal electricity-gas flow. *IEEE Transactions on Power Systems*, 37(2):1326–1339, 2021.

[22] Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 803–814. ACM, 2020.

[23] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024.

[24] Garth P McCormick. Computability of global solutions to factorable nonconvex programs: Part i—convex underestimating problems. *Mathematical programming*, 10(1):147–175, 1976.

[25] César Munoz and Anthony Narkawicz. Formalization of bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning*, 51:151–196, 2013.

[26] Arkadi Nemirovski. Prox-method with rate of convergence o (1/t) for variational inequali-

ties with lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004.

[27] Victor Y Pan and Zhao Q Chen. The complexity of the matrix eigenproblem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 507–516, 1999.

[28] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. 2019. To appear in SODA'19.

[29] Jonah Sherman. Nearly maximum flows in nearly linear time. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 263–269. IEEE, 2013.

[30] Jonah Sherman. Area-convexity, l_∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 452–460, 2017.

[31] Jan Van Den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514. IEEE, 2023.

[32] M Wright, A Fitzgibbon, Peter J Giblin, and Robert B Fisher. Convex hulls, occluding contours, aspect graphs and the hough transform. *Image and Vision Computing*, 14(8): 627–634, 1996.

[33] Lan Xue and Hua Liang. Polynomial spline estimation for a generalized additive coefficient model. *Scandinavian Journal of Statistics*, 37(1):26–46, 2010.