Subadditivity of Piecewise Linear Functions

By

Jiawei Wang

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

MATHEMATICS

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Matthias Köppe (Chair)

Jesús De Loera

Yueyue Fan

Committee in Charge

2020

Contents

Abstract

An optimization problem is trying to minimize or maximize a real objective function such that the input variables satisfies certain constraints. The simplest optimization problem is the linear program (LP), which has been proved to be in the P class and are usually solved by the simplex method or interior point methods. However, imposing integral constraints to an optimization problem can make the problem difficult to solve, and the mixed integer program (MIP) belongs to the NP-hard class.

Mixed integer optimization problems have a large number of applications in various fields, such as operations research and machine learning. Although MIP are typically hard to solve, the good news is that researchers are making substantial progress on solving problems more efficiently using better computation powers and more robust algorithms. Cutting plane algorithms, which were first proposed by Gomory and Johnson in the 1960s, are widely used in the state-of-the-art solvers. In the cutting plane algorithms, a family of real functions (so called cut-generating functions) are used to provide valid constraints to the optimization problem so that they can be solved faster. Cut-generating functions are typically piecewise linear functions and satisfy subadditivity. In this dissertation, we study the subadditivity of piecewise linear functions and use a software to verify subadditivity more efficiently.

It is important to verify subadditivity of a cut-generating function before applying it to optimization problems. As the structure of the cut-generating function gets complicated, the existing algorithm can take a very long time to verify subadditivity. We develop a spatial branch and bound algorithm to prove or disprove subadditivity of a given piecewise linear function. We use a benchmark work to show that the new algorithm works better for functions with complicated structures. We also address the reproducibility of the benchmark work, and we provide a open sourced repository so that other interested researchers can reproduce the experiment.

Dual-feasible functions are in the scope of superadditive duality theory, and they are an important family of functions which have been used in certain combinatorial optimization problems. We provide a new characterization of strong dual-feasible functions and we relate them to cut-generating functions. Inspired by results on cut-generating functions, we discover new results on dual-feasible functions. Software has been used to study properties of dual-feasible functions, based on which new dual-feasible functions can be found.

Acknowledgments

This thesis would certainly not exist without the guidance of my advisor, Prof. Matthias Köppe. I feel honored to work with him on interesting theoretical and computational projects. I would like to express my thanks to him for introducing me to the optimization world and providing extensive helps on mathematical and computer science aspects. His guidance has undoubtedly made me a better researcher.

I would like to thank Prof. Yueyue Fan and Prof. Jesús De Loera for serving on my thesis committee and providing valuable comments and suggestions. I would also like to thank Prof. Yuan Zhou, who is a former student of Prof. Matthias Köppe. The codes she wrote are very helpful and gave me a good start of my projects.

Many thanks to all my friends at Davis. I really enjoy the life with you in this small and quiet town.

Finally, I want to thank my parents and family. Without the support of them, I'm sure that I would not be the person I am today. I am especially grateful to my wife Qiongyu Li, who has been giving me care, support and happiness.

CHAPTER 1

Introduction

An integer linear program is a classic mathematical model, which tries to optimize a linear function subject to linear constraints and integral variables. It is a well-known result that integer linear program belongs to the NP-hard class, so integer linear programs are very hard to solve in theory. Although integer linear programs are also hard to solve in practice, it is possible that some large scale optimization problems can be solved in a reasonable time, due to the rapid evolution of computation power and extensive research on optimization algorithms.

An general integer linear program has the following form:

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && Ax \geq b \\ & && x \in \mathbb{Z}^n. \end{aligned} \tag{1.1}$$

Integer programs have various applications including transportation, economics and machine learning. Those problems in practice are first formulated by a integer program, then the program is fed into a state-of-art optimization solver to obtain optimal solutions. Note that there is usually more than one formulation for a problem, and a stronger formulation will typically make the solving faster. Optimization solvers are actively studied by academic researchers and developed by commercial companies, which is aimed at solving general optimization problems fast. Due to the strong modeling power and accessibility to optimization solvers, integer programs have been widely used in practice.

Here is one illustrative integer program formulation for a combinatorial optimization problem, called the bin packing problem.

EXAMPLE 1.0.1. *Suppose there are* 10 *items A with weight* 0.3 *each and* 20 *items B with weight* 0.4 *each. The goal is to use a minimum number of bins to pack all items such that the weight of each bin is at most 1.*

The idea of the formulation is to generate all optimal packing patterns such that no more item can be packed into the bin. It is not hard to list all three packing patterns: 2 *item B, or* 2 *item A and* 1 *item B, or* 3 *item A. We introduce three integer variables to denote how many bins used for each pattern. In the following formulation, the total number of bins are minimized and at least* 10 *item A and* 20 *item B should be packed. The reason to use inequalities is that more items could be packed because optimal packing patterns are used.*

$$
\begin{aligned}
&\textit{minimize} && x_1 + x_2 + x_3 \\
&\textit{subject to} && 0x_1 + 2x_2 + 3x_3 \geq 10 \\
&&& 2x_1 + x_2 + 0x_3 \geq 20 \\
&&& x_1, x_2, x_3 \in \mathbb{Z}_+
\end{aligned}
\tag{1.2}
$$

The strength of an integer program (IP) formulation depends on its linear program (LP) relaxation, which is obtained by removing integral constraints from the original formulation. For instance, the following program is the LP relaxation of the integer problem (1.1).

$$
\begin{aligned}
&\text{minimize} && c^T x \\
&\text{subject to} && Ax \geq b \\
&&& x \in \mathbb{R}^n.
\end{aligned}
\tag{1.3}
$$

It is well known that linear programs belong to the class P, and they can usually be solved efficiently by using the simplex method or the interior point method. If the optimal solution x^* of the LP relaxation satisfies all integral constraints, then the solution is also the optimal solution of the original IP. Otherwise, x^* doesn't satisfies some integral constraints of the IP. The cutting plane algorithm used in the state-of-the-art solvers is trying to find a linear constraint $\alpha^T x \geq \beta$, which is satisfied by all feasible solutions to the original IP and is violated by the optimal solution x^* of the LP relaxation. The type of inequality $\alpha^T x \geq \beta$ is called *valid inequality* or *cut*. Adding cuts to the LP relaxation will make its feasible region "smaller" (closer to the convex hull of all feasible solutions of IP), thus making the formulation stronger and the solving process faster.

A natural question to ask is: how to generate cuts? It would be best to have a method to generate cuts fast and applicable to general integer programs. Cut-generating functions can map coefficients in the optimal dictionary of the LP relaxation to coefficients in valid inequalities. The first generation of cut-generating functions appeared in the so-called master finite group relaxation which was introduced by Gomory in 1969 [**Gom69a**]. Later in 1972, Gomory and Johnson [**GJ72a**, **GJ72b**] extended the notion to infinite group relaxation. In the recent years, researchers have studied cut-generating functions in the infinite group relaxation problems in terms of diversity and effectiveness. They are trying to discover cut-generating functions with interesting properties and hope to find effective cuts applicable to the state-of-the-art solvers. Another trending topic nowadays is the so-called multi-row cutting planes, which are expected to add cuts more efficiently. For example, the paper [**BHKM13**] introduced a multi-dimensional cut-generating functions which can be used to generate "strong" valid inequalities (facets).

Although cut-generating functions do not have to be a "piecewise linear function" (we will give formal definitions later), almost all those that are used in practice are. The software [**KZHW20**] implements the single row Gomory–Johnson cut-generating functions, and it can be used to verify whether a given piecewise linear function is "strong". In the verification process, one key component is to check whether the function is subadditive. As the structure of a piecewise linear function gets complicated, the previous subadditivity test could take a long time to finish. We develop a new version of the subadditivity test using the spatial branch and bound technique, and we expect the new version has better performance as the structure of functions gets more complex. The spatial branch and bound algorithm may also be the key to the computational approaches to studying subadditivity in the multi-row cutting plane theory.

To compare the performance of different algorithms in practice, the standard way is to do a benchmark experiment. In the optimization community, there is extensive research on benchmarking solvers solving different types of optimization problems, for example [**mip18**]. In both natural science and computer science fields, researchers have been evaluating benchmark works in terms of reproducibility [**Bak16**, **CP16**]. Reproducibility is a key component to identify whether a benchmark work is convincing. There is also a trend in the academic field toward emphasizing reproducible benchmark works [**VK12**, **RHGM18**]. We design a benchmark experiment to compare different subadditivity test algorithms. The repository of the experiment is open-sourced, and the entire experiment can be reproduced from instances generation to data analysis.

The duality theory of integer linear optimization is a multifaceted research topic that connects to cutting plane theory and the theory of value functions of parametric optimization problems. The central objects on the dual side of this theory are *superadditive functionals*. Those superadditive functionals have different concrete forms, including dual-feasible functions [**ACdCR16**]. Dual-feasible functions provide strong dual bounds in a branch-and-bound algorithm and strong valid inequalities in a cutting-plane procedure. They have been applied to this effect in particular for combinatorial optimization problems that benefit from a column-generation approach, such as bin-packing or cutting-stock problems [**Lue83**, **Van00**].

Dual-feasible functions (DFFs) have been considered an important technique to combinatorial optimization problems alone, which possibly limits their applicability. In fact, they have a deep connection with cut-generating functions in term of subadditivity. The recent discovery of the relation between dual-feasible functions and cut-generating functions makes it possible for dual-feasible functions to be applied to a broader field and foster general integer programming. On the other hand, we would like to make the powerful results for cut-generating functions — and the rich toolsets that were used to obtain them — available for spaces of functions that arise from the study of broader algorithmic frameworks. This includes algorithms for large-scale problems based on decomposition techniques. The context in which the study of DFFs arose, combinatorial problems with Dantzig–Wolfe decomposition, is one such setting.

In this dissertation, we focus on single-row cut-generating functions for infinite relaxation problems and two types of dual-feasible functions, and we study the subadditivity (superadditivity) condition in the characterizations of those piecewise linear functions.

1.1. Cut-generating functions

Cut-generating functions are closely tied to the setting of tableau cuts in a simplex-based cutting plane procedure. They play an essential role in generating valid inequalities which cut off the *current fractional basic solution* in the LP relaxation. In this way, they explain and generalize the Gomory fractional cut and Gomory mixed-integer cut, the workhorses of state-of-art integer programming solvers. The cut-generating functions in the Gomory–Johnson model [**GJ72a**, **GJ72b**] are related to Gomory's corner relaxation [**Gom69b**], which is obtained by relaxing the non-negativity of all basic variables in the tableau. Thus, basic integer variables are allowed to take any value in $\mathbb{Z}$ in the relaxations. (Cut-generating functions for stronger relaxations have been studied too: In the

model of Yıldız and Cornuéjols [**YC16**], basic variables are constrained to some set $S \subset \mathbb{R}$ with suitable properties. We will come back to this model later.)

We first introduce the Gomory–Johnson cut-generating functions of the infinite relaxation model; details can be found in [**BHK16a**, **BHK16b**]. Consider the single-row Gomory–Johnson model, which takes the following form:

$$x + \sum_{r \in \mathbb{R}} r\, y(r) = b, \quad b \notin \mathbb{Z}, b > 0, \tag{1.4}$$

$$x \in \mathbb{Z},\ y : \mathbb{R} \to \mathbb{Z}_+, \text{ and } y \text{ has finite support.}$$

Let $\pi \colon \mathbb{R} \to \mathbb{R}$ be a nonnegative function, and π is called a *valid* function if $\sum_{r \in \mathbb{R}} \pi(r)\, y(r) \geq 1$ holds for any feasible solution (x, y) to (1.4). There is a hierarchy of valid functions regarding their strength. The *minimal* functions are those that are pointwise non-dominating. As we will see later, they are the ones that have a characterization involving subadditivity. Among the minimal functions, a function is said to be *extreme* if it can not be written as a convex combination of two other valid functions. Minimal/extreme functions are precisely those that generate all non redundant, nontrivial, facet-defining valid inequalities in all possible corner relxations. Minimal functions are characterized by subadditivity and several other properties as follows.

THEOREM 1.1.1 ([**GJ72a**]). *A non-negative function $\pi \colon \mathbb{R} \to \mathbb{R}$ is minimal to (1.4) if and only if*

(i) $\pi(x) = 0$ for all $x \in \mathbb{Z}$;

(ii) π is subadditive in the sense that $\pi(x) + \pi(y) \geq \pi(x + y)$ for all $x, y \in \mathbb{R}$;

(iii) π satisfies the symmetry condition in the sense that $\pi(x) + \pi(b - x) = 1$ for all $x, y \in \mathbb{R}$.

Note that a minimal function π is periodic modulo 1 ($\mathbb{Z}$-periodic) and the functions values are bounded between 0 and 1. Therefore, only the restriction $\pi|_{[0,1]}$ is of interest.

1.2. Subadditivity of piecewise linear functions

We begin to introduce the definitions of piecewise linear functions in the Gomory–Johnson setting using polyhedral complexes; more details can be found in [**BHK14**, section 2.1] and [**HKZ18c**, **BHK16a**].

Let $0 = a_0 < a_1 < \cdots < a_{n-1} < a_n = 1$, and denote by $B = \{a_0, a_1, \ldots, a_{n-1}, a_n\}$ the set of all possible *breakpoints*. The 0-dimensional faces are defined to be the singletons, $\{a_i\}$, $a_i \in B$, and the 1-dimensional faces are the closed intervals, $[a_i, a_{i+1}]$, $i = 0, \ldots, n-1$. Together they form $\mathcal{P} = \mathcal{P}_B$, a finite polyhedral complex. We call a function $\pi\colon [0,1] \to \mathbb{R}$ *piecewise linear* over $\mathcal{P}_B$ if for each face $I \in \mathcal{P}_B$, there is an affine linear function $\pi_I\colon \mathbb{R} \to \mathbb{R}$, $\pi_I(x) = c_I x + b_I$ such that $\pi(x) = \pi_I(x)$ for all $x \in \operatorname{rel\,int}(I)$. Under this definition, piecewise linear functions can be discontinuous. Let $I = [a_i, a_{i+1}]$. The function π can be determined on the open intervals $\operatorname{int}(I) = (a_i, a_{i+1})$ by linear interpolation of the limits $\pi(a_i^+) = \lim_{x \to a_i, x > a_i} \phi(x) = \pi_I(a_i)$ and $\pi(a_{i+1}^-) = \lim_{x \to a_{i+1}, x < a_{i+1}} \pi(x) = \pi_I(a_{i+1})$. We say the function π is continuous piecewise linear over $\mathcal{P}_B$ if it is affine over each of the cells of $\mathcal{P}_B$ (thus automatically imposing continuity).

Note that we assume that a piecewise linear function has finitely many breakpoints, which is also convenient to implement the function in the software.

Following the notation in [**BHK14**, **BHK16a**, **BHK16b**, **BHKM13**, **HKZ18a**], we introduce the function $\Delta\pi\colon \mathbb{R}\times\mathbb{R} \to \mathbb{R}$, $\Delta\pi(x,y) = \pi(x)+\pi(y)-\pi(x+y)$. The function $\Delta\pi$ measures the slack in the subadditivity condition. Observe that the piecewise linearity of π induces piecewise linearity of $\Delta\pi$. In order to express the domains of linearity of $\Delta\pi(x,y)$, and thus domains of additivity and strict subadditivity, we introduce the two-dimensional polyhedral complex $\Delta\mathcal{P} = \Delta\mathcal{P}_B$. The faces F of the complex are defined as follows. Let $I, J \in \mathcal{P}_B$ and $K \in \mathcal{P}_B \cup (\mathcal{P}_B + 1)$, where $\mathcal{P}_B + 1$ is the periodic extension of $\mathcal{P}_B$. So each of I, J, K is either a breakpoint of ϕ (or its periodic extension to $[1,2]$) or a closed interval delimited by two consecutive breakpoints. Then $F = F(I,J,K) = \{\, (x,y) \in \mathbb{R} \times \mathbb{R} : x \in I,\ y \in J,\ x + y \in K \,\}$. Note that π is actually defined for all real numbers and for simplicity only the domain $[0,1]$ is implemented. Assume both x, y are in the range $[0,1]$, then the range of $x + y$ is actually $[0,2]$, which is the reason to include periodic extension for the K component. The projections $p_1, p_2, p_3\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ are defined as $p_1(x,y) = x$, $p_2(x,y) = y$, $p_3(x,y) = x + y$. Let $F \in \Delta\mathcal{P}$ and let $(u,v) \in F$. We define

$$\Delta\pi_F(u,v) = \lim_{\substack{(x,y)\to(u,v)\\(x,y)\in\operatorname{rel\,int}(F)}} \Delta\pi(x,y),$$

which allows us to conveniently express limits to boundary points of F, in particular to vertices of F, along paths within $\operatorname{rel\,int}(F)$. It is clear that $\Delta\pi_F(u,v)$ is affine over F, and $\Delta\pi(u,v) = \Delta\pi_F(u,v)$ for all $(u,v) \in \operatorname{rel\,int}(F)$. We will use $\operatorname{vert}(F)$ to denote the set of vertices of the face F.

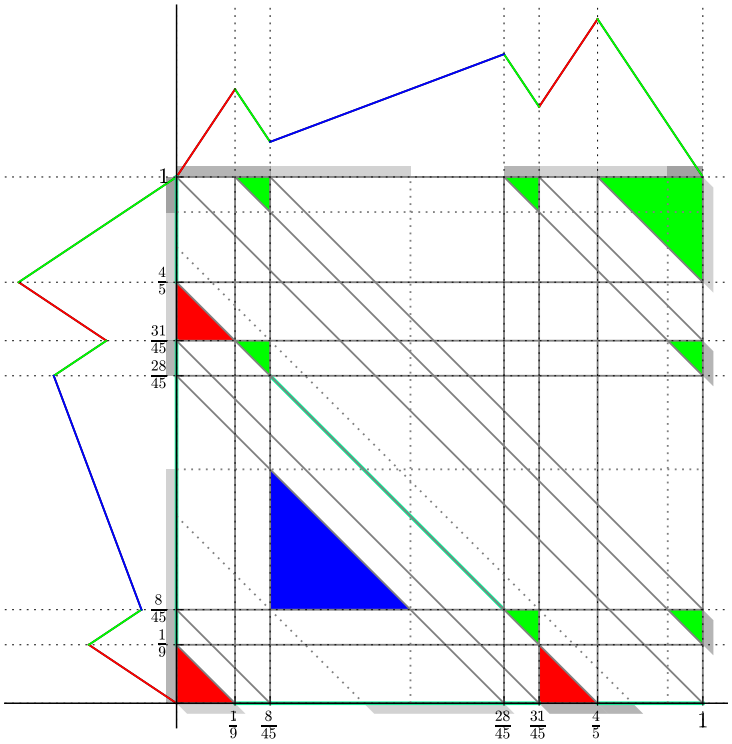

FIGURE 1.1. Diagram of an extreme function and the corresponding painting on the two-dimensional polyhedral complex $\Delta\mathcal{P}$ (gray solid lines). The plot can be regenerated by the command `plot_2d_diagram(pi,colorful=True)`, where `pi = gj_forward_3_slope()`. The extreme function `pi` is plotted on the top and left borders. The heavy diagonal green lines $x+y=\frac{4}{5}$ and $x+y=\frac{9}{5}$ correspond to the symmetry condition. Additive faces in $\Delta\mathcal{P}$ are painted with three colors. At the borders, the projections $p_i(F)$ of two-dimensional additive faces are shown as gray shadows: $p_1(F)$ at the top border, $p_2(F)$ at the left border, $p_3(F)$ at the bottom and the right borders. Different colors on the function and on the additive faces represent different "covered components" used in the extremality test.

We now define the *additive faces* of the two-dimensional polyhedral complex $\Delta\mathcal{P}$ of π. When π is continuous, we say that a face $F \in \Delta\mathcal{P}$ is additive if $\Delta\pi = 0$ over all F. Notice that $\Delta\pi$ is affine over F, the condition is equivalent to $\Delta\pi(u,v) = 0$ for any $(u,v) \in \text{vert}(F)$. When π is discontinuous, following [**KZHW20**], we say that a face $F \in \Delta\mathcal{P}$ is additive if F is contained in a face $F' \in \Delta\mathcal{P}$ such that $\Delta\pi_{F'}(x,y) = 0$ for any $(x,y) \in F$. Since $\Delta\pi$ is affine in the relative interiors of each face of $\Delta\mathcal{P}$, the last condition is equivalent to $\Delta\pi_{F'}(u,v) = 0$ for any $(u,v) \in \text{vert}(F)$.

We include Figure 1.1 [1] here as an example of the two dimensional polyhedral complex of an extreme cut-generating function together with additive faces.

[1]The plot is generated by our software [**KZHW20**], which is written in Python, using the framework of SageMath [**S$^+$16**], a comprehensive Python-based open source computer algebra system. In this dissertation, a function name shown in typewriter font refers to the `cutgeneratingfunctionology` module of our SageMath program.

After defining the two-dimensional polyhedral complex, we are ready to introduce a classical algorithm for testing minimality for piecewise linear functions. We do not emphasize the extremality of cut-generating functions in this dissertation, and we refer interested readers to [**HKZ18b**]. The major part of the minimality test is based on the two real variable function $\Delta\pi$ defined in the two-dimensional polyhedral complex. The subadditivity is equivalent to $\Delta\pi \geq 0$, which can be verified by enumerating vertices (together with limiting cones in the discontinuous case) of all faces in $\Delta\mathcal{P}$. Roughly speaking, there are quadratically many vertices in $\Delta\mathcal{P}$ in terms of the total number of breakpoints, thus the classical algorithm has at least quadratic time complexity (without considering the complexity of the function evaluation). The algorithm is quite straightforward and easy to implement, and it works well if the total number of breakpoints is not large. However, as more cut-generating functions with complex structure are being discovered, the classical algorithm can take a long time to finish. The observation inspires us to pursue a more efficient method for the subadditivity test, especially for complicated functions.

We develop a spatial branch and bound (sBB) algorithm for studying the subadditivity slack of piecewise linear functions. The motivation of using the sBB algorithm is that we hope to prove subadditivity in a region containing a large number of vertices fast by constructing a strong convex relaxation.

The sBB algorithms [**SP99**, **TSS02**] are the extension of traditional Branch-and-Bound (BB) algorithms to continuous solution space. In the sBB algorithms, the solution space is successively partitioned into smaller and smaller regions. The problem is solved recursively by closing the gap of upper and lower bounds to the objective function value. The sBB algorithms are usually applied to smooth nonlinear programming with bounded feasible region and twice differentiable objective function. The idea of sBB algorithms has been implemented in several solvers including BARON [**TS05**], SCIP [**BGG$^+$12**], α-BB [**AAF98**].

In regards to sBB algorithms, the key step is also how to construct a convex relaxation of the original nonconvex problem. In a global minimization problem, the convex relaxation should be easy to solve and the solution should provide a lower bound for the objective in the original problem. At each iteration of the algorithm, consider a convex relaxation of the original problem restricted to a sub-region and solve the relaxation to get a local lower bound and a global upper bound. If the current local lower bound is larger than the best global upper bound so far, then it is impossible for the current region to contain the optimal solution, therefore no branching is

needed. Eventually, the solution space is explored exhaustively and the best global upper bound is the optimal objective function value. Note that the efficiency of the sBB algorithms depends on the selection rule of the sub-regions and the construction of convex relaxations.

The key idea in an sBB algorithm is to prune some regions which cannot contain any better solutions based on the current best solution. It is crucial to construct tight convex relaxations which can produce good and fast bounds therefore prune some regions efficiently. One way to generate a convex relaxation makes use of linearizing all nonconvex terms and then replacing each nonconvex definition constraint with the upper concave and lower convex envelopes [**SP99**]. However, it is not always easy to find the tightest envelopes, so slacker estimators are usually used instead. Various common underestimators have been studied in the literature, including convex univariate, concave univariate terms [**SP99**] and piecewise convex and concave terms [**LP03**].

In regards to the subadditivity test setting, we use the affine functions as the upper and lower estimators. One reason to use affine estimators is that computing such estimators and solving the respective convex relaxation is relatively efficient. Although it is not clear what is the tightest affine underestimator of $\Delta\pi$ on some region, we can show that the estimator which has the best lower bound can be computed by solving an LP. Here is a tradeoff which is analogous to the cutting plane algorithm in a branch and bound tree. Adding cutting planes generally reduces the size of the branch and bound tree, but it also adds more computations for cuts generation. Similarly, construction of strong convex relaxations leads to early pruning, but the construction in each node can be burdensome. In our sBB tree, the strongest convex relaxation results from solving an LP. Although solving an individual LP is typically easy, it is not ideal to solve a large number of LPs.

Due to the use of branch and bound technique and LP solvers, it is difficult to analyze the time complexity of the sBB algorithm for the subadditivity test. Generally speaking, a branch and bound algorithm does not have a proven time complexity, but there exists analysis of the algorithm applied to specific problems. For example, certain knapsack problems have a proven exponential time complexity using a particular type of branch and bound algorithm [**Chv80**, **JS11**]. In our subadditivity study, we observe that the sBB algorithm can outperform the classical subadditivity test in some hard instances. The sBB algorithm has other potential applications for cut-generating functions, including generating additive faces and verifying objective limit. It may also be an efficient technique for studying subadditivity of multi-row cut-generating functions.

1.3. Reproducibility

Extensive academic research breakthroughs are grounded on reproducible works, which provide more reliability than non-reproducible ones. Due to the "reproducibility crisis" being discovered in various fields, the scientific community nowadays is emphasizing more on reproducibility in a research work [**VK12**]. In this dissertation, we focus on the reproducibility of a benchmark work in a resource sharing platform.

Under the term "reproducibility", there are various aspects in a benchmark work. The authors in [**Fei06**] suggested that the main objective of reproducibility is to gain insights and make progress based on other researchers' work. They pointed out that one benefit of a reproducibility study is that, by conducting the same experiment in a slightly different way (for example, in a different platform or different configuration parameters), it will bring the original experiment into a broader and more general theory. Even if some discrepancy is discovered, it is also a valuable context which leads to further necessary research. It was also suggested in [**Fei06**] that standardized experimental design is preferred since similar methodology like a lab manual is widely used in natural science field [**GS12**]. Repetitions together with statistical analysis are an important aspect in a benchmark work. Due to randomness, repetitions are necessary in order to generate statistically valid results, like the confidence interval. It is also crucial to minimize the cost of total experiment time while maintaining the statistically rigorous results. Sequential stopping rules can be applied to determine when there is sufficient data to draw a rigorous conclusion and the collection of data can stop [**Sta66**, **Sin14**, **CW20**]. Another important aspect is whether the benchmark work is made open-sourced, including instances generation, scripts to run the experiment and statistical analysis. The paper [**VK12**] also suggested that the computer science society should encourage researchers to publish reproducibility study of others' work, and the journal editors should accept such work no matter whether the reproducibility study supports the original paper or not. The authors in [**VK12**] believed that such encouragement and culture change can foster the reproducibility study and on the other hand lead researchers towards publishing reproducible papers.

In the optimization field, the well known benchmark MIPLIB library has been actively studied in the mixed integer programming community. MIPLIB is a library of mixed integer programming test cases which are tested on multiple optimization solvers, including both open-sourced and commercial ones. The latest version MIPLIB 2017 [**mip18**] is the sixth generation of MIPLIB

library which was firstly created in 1992. The website of MIPLIB 2017 is consistently updated to indicate improvements including less solving time, first feasible solution, better feasible solution, provable optimal solution, or provable infeasibility. The source code of the MIPLIB 2017 is also provided in the website for other researchers to reproduce the benchmark work and perform data analysis. Another important component of MIPLIB 2017 is the "benchmarking set", containing mixed integer problems to which optimization solvers are applied. The benchmarking set has been carefully selected so that it can best represent today's mixed integer problems. The shifted geometric mean has been used as the "standard" performance metric in computational MIPs. However, the choice of the shift value is not standard, depending on whether the authors want to emphasize more on easy or hard problems. In terms of the solving time, the shifts of 1s [**GEG$^+$17**], 10s [**KB16**], 60s [**Ach07**] have been used. Use of other performance metrics, like performance profiles [**DM02**] or PAR10 [**KMS$^+$11**], is also not uncommon.

Ideally, measuring performance metric, for example execution times, should be done assuming that the program is the only one running on the platform. However, if the computation is done in a time/resource sharing hardware such as a high performance cluster (HPC), it is impossible to control the resource competition caused by other programs of other users. We can still use the computational results to see how the competitions of resources, like CPU or memory, can influence the measurements. In the high performance cluster we use, it is possible to assign a priority to the jobs we submitted. Since the computations with higher priority will be more competitive for the resource, it is also possible to test the influence of different priorities to execution times (or wall clock time).

We design a benchmark experiment in a high performance cluster (HPC) to make comparisons among different algorithms in the subadditivity study. The experiment also serves as a demo for the reproducibility framework. The overall computation time would be too long if the experiment was run on a personal computer, so we use the computation power and parallelism in a HPC to finish the entire computations in a reasonable amount of time. In our benchmark work, we are aimed at providing an open-sourced repository which can be used to reproduce the complete experiment from generating test instances to reporting data analysis results. We use git as a version control system for the repository and use the git submodule technique to store raw data. Different branches of the submodule store computational results for different computational tasks, which makes the structure of the repository clean to view. An Jupyter Notebook is used to perform interactive data

analysis which can be done on a personal laptop. Our notebook is aimed at maximizing flexibility of data analysis so that other researchers can use the notebook as a template and do their own analysis.

We study statistical properties of our experiment results. The normality assumption is checked on the collected data. Then we fit data into various distribution family using maximum likelihood estimation. The goodness of fit is verified by the Kolmogorov-Smirnov test, which is a hypothesis test to determine whether samples are drawn from a given distribution. The goal of the distribution fitting is to show that there exist better distribution candidates for our experiment results other than the normal distribution. We investigate two sequential stopping rules based on confidence interval procedure, and robustness and efficiency are tested by a Monte Carlo simulation. Since the experiment is on a resource shared platform, the wall clock time might be more variant than the CPU time. We use a hypothesis test to compare the variation of CPU time and wall clock time, and we find that in general there is not enough evidence to say they do not have equal variance.

1.4. Dual-feasible functions

Within the framework of superadditive duality theory, the recent monograph [**ACdCR16**] has studied characterizations and applications of dual-feasible functions. In the monograph, two simple and fundamental settings in which superadditive functionals of a single real variable appear, are defined as follows. *Classical dual-feasible functions (cDFFs)* are functions $\phi\colon D \to D$ such that

$$\sum_{i\in I} x_i \leq 1 \quad \Rightarrow \quad \sum_{i\in I} \phi(x_i) \leq 1 \tag{1.5}$$

holds for any family $\{x_i\}_{i\in I} \subseteq D$ indexed by a finite index set I, where $D = [0,1]$. In [**ACdCR16**, Chapters 2 and 3], these functions are studied alongside with *general dual-feasible functions (gDFFs)*, which satisfy the same property (1.5) for the extended domain $D = \mathbb{R}$. (There is an equivalent definition of these functions in terms of valid inequalities for infinite-dimensional integer programming models, which we suppress until Section 4.4.2.)

Example 1.4.1. *We use the previous example Example 1.0.1 to illustrate how dual-feasible functions are used to generate fast dual bounds in combinatorial optimization problems.*

The LP relaxation of the original IP formulation is obtained by removing the integral constraints. The optimal objective of the following LP relaxation is an lower bound of the original IP.

$$\begin{aligned} \textit{minimize} \quad & x_1 + x_2 + x_3 \\ \textit{subject to} \quad & 0x_1 + 2x_2 + 3x_3 \geq 10 \\ & 2x_1 + x_2 + 0x_3 \geq 20 \\ & x_1, x_2, x_3 \in \mathbb{R}_+ \end{aligned} \tag{1.6}$$

Based on strong duality theory in linear programming, the dual problem of the above (feasible) linear program is also a linear program, and they have the same optimal objective. Thus, every feasible solution to the following dual problem can generate a lower bound of the original IP.

$$\begin{aligned} \textit{maximize} \quad & 10u_1 + 20u_2 \\ \textit{subject to} \quad & 0u_1 + 2u_2 \leq 1 \\ & 2u_1 + 1u_2 \leq 1 \\ & 3u_1 + 0u_2 \leq 1 \\ & u_1, u_2 \in \mathbb{R}_+ \end{aligned} \tag{1.7}$$

The dual-feasible functions then can be applied to generate feasible solutions to the above dual problem. In this example, the claim is that $(\phi(0.3), \phi(0.4))$ *is a feasible solution for any (classical or general) dual-feasible function* ϕ*, and then* $\lceil 10\phi(0.3) + 20\phi(0.4) \rceil$ *is a lower bound of the original IP problem. The claim is not hard to prove by using the definition of dual-feasible functions and the notion of "optimal" packing patterns. In this way, fast dual bounds of an IP problem can be generated by plugging numbers to dual-feasible functions, without the need of solving any optimization problems.*

For each of the two settings, classical and general, there is a hierarchy of DFFs regarding their strength, following the same logic as cut-generating functions. The *maximal* DFFs are those that are pointwise non-dominated. As we will see later, they are the ones that have a characterization involving superadditivity. Among the maximal DFFs, a DFF is said to be *extreme* if it can not be written as a convex combination of two other maximal DFFs.

THEOREM 1.4.1 (Characterization of maximal cDFFs, [**ACdCR16**, Theorem 2.1]). *A function $\phi\colon [0,1] \to [0,1]$ is a maximal cDFF if and only if the following conditions hold:*

(i) $\phi(0) = 0$*;*

(ii) ϕ *is superadditive in the sense that* $\phi(x) + \phi(y) \leq \phi(x+y)$*;*

(iii) $\phi(x) + \phi(1-x) = 1$ *for all* $x \in [0,1]$.

THEOREM 1.4.2 (Characterization for maximal gDFFs, [**KW19**, Theorem 4.3]). *A function $\phi\colon \mathbb{R} \to \mathbb{R}$ is a restricted maximal gDFF if and only if the following conditions hold:*

(i) $\phi(0) = 0$*;*

(ii) ϕ *is superadditive in the sense that* $\phi(x) + \phi(y) \leq \phi(x+y)$*;*

(iii) $\phi(x) \geq 0$ *for all* $x \in \mathbb{R}_+$*;*

(iv) $\phi(x) + \phi(1-x) = 1$ *for all* $x \in \mathbb{R}$ *or* $\phi(x) = cx$ *for some constant* $c \in [0,1]$.

These features of the theory are remarkably similar to the ones in the study of the valid inequalities for the Gomory–Johnson infinite group problem [**GJ72a**, **GJ72b**], and the broader context of cut-generating functions [**CCD$^+$13**, **YC16**]. We attempt to determine the precise relation between DFFs and cut-generating functions, and to transfer recent advances in the study of the latter to the DFF setting, in the hope that they will prove useful there.

Like the majority of the development in [**ACdCR16**, Chapter 2], we consider piecewise linear functions that are allowed to be discontinuous at the breakpoints. Piecewise linear DFFs can be defined analogously using polyhedral complex and we omit the formal definition here. Figure 1.2 is an example of the $\Delta\mathcal{P}$ of a maximal cDFF. In contrast to Gomory–Johnson cut-generating functions, the two dimensional polyhedral complex of cDFFs only contains the lower triangle since the domain of cDFFs is $[0,1]$.

Inspired by the implementation of the single-row Gomory–Johnson model, we transfer and extend recent algorithmic techniques [**BHK14**, **HKZ18c**, **Zho17**, **HKZ18a**, **HKZ18b**] to piecewise linear DFFs. The algorithms are implemented in the current version of the software [**KZHW20**] alongside with the Gomory–Johnson cut-generating functions. Similar to [**KZ15**], we provide an electronic compendium of the known extreme DFFs from [**ACdCR16**]. The extremality of the functions from this library is proved in [**ACdCR16**] by studying analytical properties of extreme DFFs.

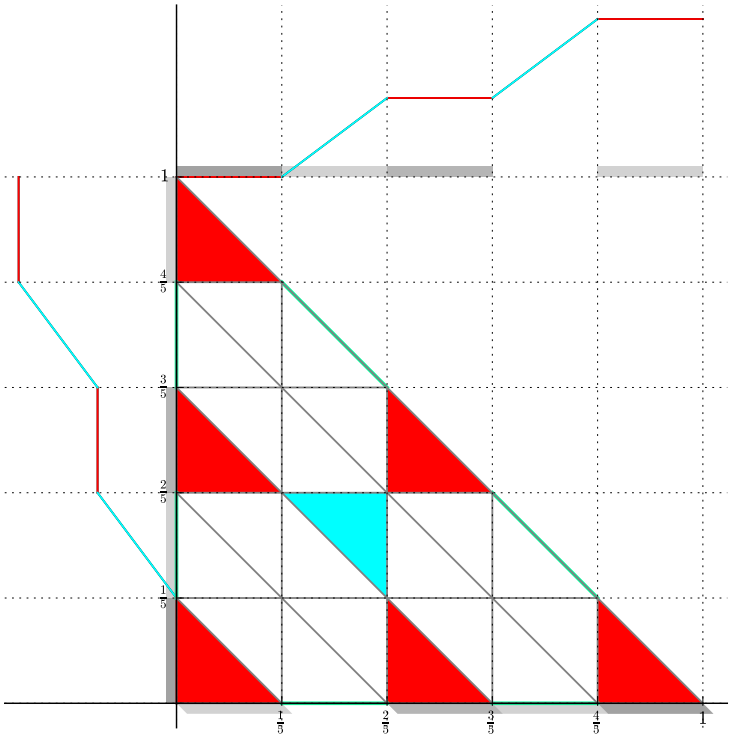

FIGURE 1.2. Maximal cDFF $\phi_{BJ,1}(x;C) = \frac{\lfloor Cx \rfloor + \max(0, \frac{\{Cx\}-\{C\}}{1-\{C\}})}{\lfloor C \rfloor}$ for $C = \frac{5}{2}$, and its two dimensional polyhedral complex. The plot can be regenerated by the command `plot_2d_diagram_dff_no_lable(phi, colorful=True)`, where `phi = phi_bj_1(c=5/2)`. The function `phi` is plotted on the top and left borders. The diagonal blue lines $x + y = 1$ corresponds to the symmetry condition. Additive faces are painted colorful. At the borders, the projections $p_i(F)$ of two-dimensional additive faces are shown as gray shadows: $p_1(F)$ at the top border, $p_2(F)$ at the left border, $p_3(F)$ at the bottom and the right borders. Different colors on the function and on the additive faces represent different "covered components" used in the extremality test.

We complement this by our algorithmic techniques, leading to automatic maximality and extremality tests for cDFFs. They are based on the methods of polyhedral complexes and functional equations from [**BHK14**, **HKZ18c**] and the inverse semigroup techniques from [**HKZ18a**, **HKZ18b**]. On the basis of the automatic maximality and extremality test, we use a computer-based search technique based on polyhedral computation and filtering to find new extreme DFFs. Our search reveals that the classic dual-feasible functions are much richer than what is represented by the families of functions described in the literature. We hope that our software facilitates experimentation and further study.

In terms of the gDFFs, they turn out to have a very close relation to a model studied by Jeroslow [**Jer79**], Blair [**Bla78**] and Bachem et al. [**BJS82**], which we refer to by $Y_{=1}$ and a certain relaxation of this model, which we denote by $Y_{\leq 1}$. Both $Y_{=1}$ and $Y_{\leq 1}$ can be studied in the Yıldız–Cornuéjols model [**YC16**] with various sets S. GDFFs generate valid inequalities for the Yıldız–Cornuéjols model with $S = (-\infty, 0]$, and cut-generating functions generate valid inequalities for the Jeroslow model where $S = \{0\}$. These two families of functions are then connected by an operation known as "tilting" [**AEGJ03**].

The Gomory–Johnson model is well-studied and the literature provides a large library of functions with interesting structures. For example, there exist extreme functions with arbitrary number of slopes. Perhaps the most famous result of Gomory and Johnson's masterpiece [**GJ72a**, **GJ72b**] is the 2-slope theorem, showing that every continuous piecewise linear minimal valid function that has only two different slope values is an extreme function. Another interesting result is about approximation theory. Basu et al. [**BHM16**] proved that the 2-slope extreme Gomory–Johnson cut-generating functions are dense in the set of continuous minimal functions.

We introduce a conversion from Gomory–Johnson functions to DFFs, which under some conditions generates maximal or extreme cDFFs and gDFFs. This work is also a possible starting point for constructing new parametric families of DFFs with special properties. We also prove the analogous two slope theorem and an approximation result for gDFFs. In our proof of the approximation theorem, unlike the 2-slope fill-in procedure that Basu et al. [**BHM16**] used, we always use 0 as one slope value in our fill-in procedure, which is necessary since the 2-slope theorem of gDFFs requires 0 to be one slope value.

THEOREM 1.4.3 (reworded Theorem 4.5.1). *Let ϕ be a continuous piecewise linear maximal gDFF with only 2 slope values and one slope is* 0*, then ϕ is extreme.*

THEOREM 1.4.4 (reworded Theorem 4.6.1). *Let ϕ be a continuous piecewise linear maximal gDFF, if ϕ is not a linear function, then for any $\epsilon > 0$, there exists an extreme gDFF ϕ_{ext} such that $\|\phi - \phi_{\text{ext}}\|_\infty < \epsilon$.*

In contrast, these analogous theorems do not directly apply to cDFFs. We show that for cDFFs, there cannot exist a 2-slope theorem. We find a counterexample, a maximal cDFF with 2 slopes and 3 "connected covered components" (a concept from the algorithmic extremality test) that is

not extreme. It remains an open question whether maximal cDFFs can also be approximated in the same way by extreme functions.

1.5. Outline of the dissertation

Chapter 2. Subadditivity Test

This chapter studies the subadditivity slack $\Delta\pi$ of the single-row Gomory-Johnson cut-generating functions. We introduce three main algorithms for computing the minimum of $\Delta\pi$. The classical (naive) method explores every vertex in the two dimensional polyhedral complex of a piecewise linear function π. The spatial branch and bound algorithm utilizes affine estimators to construct a convex relaxation in every node of the branch and bound tree. The computation of the minimum of $\Delta\pi$ can also be formulated as a mixed integer program, then an optimization solver can be applied to solve the minimization problem. We design an experiment to compare the performance of different algorithms applied to various computation tasks related to subadditivity.

First, in Section 2.1, we explain how subadditive or nearly subadditive piecewise linear functions can be used to generate valid inequalities, even if those functions are not valid functions. Near-subadditivity means that the minimum of $\Delta\pi$ is a negative number but close to 0. The naive subadditivity test is explained in Section 2.2. The algorithm makes use of affine linearity of $\Delta\pi$ over every face in the two dimensional polyhedral complex $\Delta\mathcal{P}$, then $\Delta\pi$ is fully characterized by its values at all vertices in all faces of $\Delta\mathcal{P}$. In Section 2.3, we introduce several mixed integer programs to formulate piecewise linear functions. However, those MIP formulations have a large number of binary variables if the number of breakpoints is large. We explain the spatial branch and bound (SBB) algorithm and its variations in Section 2.4. Different constructions of the convex relaxation vary in strength, thus lead to different branch and bound tree size. In the spatial branch and bound algorithm, a linear program needs to be solved in order to get the strongest convex relaxation. We also explore different traversing strategies like BFS, DFS and the "best bound" strategy. The spatial branch and bound algorithm has other applications besides computing the minimum of $\Delta\pi$, including verifying an objective limit and generating additive faces. In Section 2.5, from our benchmark experiment, we include the computational results of comparing the naive algorithm, the sBB algorithm (its variations), and mixed integer formulations. In terms of the performance metric, performance profile is widely used to compare different algorithms, but in the MIP community the shifted geometric mean is a standard metric to use. In our benchmark

experiment, both performance profile and shifted geometric mean are reported. More details on the benchmark experiment regarding reproducibility are explained in Chapter 3.

Chapter 3. Benchmarking on High Performance Clusters

Chapter 3 focuses on performing a reproducible benchmark work on high performance clusters. In this chapter, we do not include technical optimization knowledge, and we mainly focus on presenting a benchmark experiment which is convenient for other researchers to reproduce. The reason to use HPC is that the overall computation would be too long if run on a laptop.

In Section 3.1, we first summarize literature on reproducibility, especially in the computer science field. To make the benchmark work reproducible, clear experiment design, statistically rigorous data analysis and non-proprietary work are key components. In Section 3.2, we briefly review some computational experiments which were run in a time/resource sharing hardware, like high performance clusters. We also propose several criteria for reproducible benchmark works in a HPC. Based on the criteria we propose, we evaluate the MIPLIB library in the optimization community regarding the reproducibility in Section 3.3. The MIPLIB library is a well known optimization benchmark set with several generations, and the latest version MIPLIB 2017 has been a good example of reproducible benchmark work in the MIP community. We review some basic statistical knowledge in Section 3.4 and study the sequential analysis in Section 3.5. In Section 3.6, we explain our open-sourced repository[2] for running a benchmark work in HPC. We use git as a version control system for the repository. The git submodule technique is used to store raw data, which will make the repository clean to view and easy to rerun the experiment. In terms of data analysis, we use a Jupyter Notebook to report results like running time and memory. The notebook also serves as a template to present the data interactively, and it is convenient for other researchers to do their own analysis. We also study some statistical properties of the data in Section 3.7.

Chapter 4. Dual feasible functions.

First, we briefly review key results for DFFs in Section 4.1. In Section 4.2, we introduce the implementation of cDFFs in the current version of the software [**KZHW20**], including automatic maximality and extremality tests. Based on the automatic maximality and extremality test, we use a computer-based search technique based on polyhedral computation and filtering to find new extreme cDFFs.

[2]Github repository: https://github.com/mkoeppe/jiawei-computations

Then, in Section 4.3, we turn to the study of gDFFs. Here we transfer techniques used by Yıldız and Cornuéjols for the study of their previously mentioned model of cut-generating functions, which generalizes the Gomory–Johnson framework. Inspired by the characterization of minimal Yıldız–Cornuéjols cut-generating functions and using similar techniques, we give a full characterization of maximal gDFFs.

In Section 4.4, we investigate the relation between cDFFs and gDFFs and various cut-generating functions. Motivated by the operation known as "tilting", we discover a conversion from cut-generating functions to DFFs. From our conversion, we obtain new parametric families of DFFs with interesting structures.

In Section 4.5, we show an analogous two-slope theorem for gDFFs, following the same proving process of the two slope theorem in the Gomory–Johnson model. CDFFs have a bounded domain, which leads to non-existence of a similar two slope theorem for cDFFs. Finally, in Section 4.6, we turn to the approximation theory of gDFFs. We prove an approximation theorem, which indicates that almost all continuous maximal gDFFs can be approximated by extreme (2-slope) gDFFs as close as we desire.

CHAPTER 2

Subadditivity Test

We study methods for testing (near) subadditivity of continuous piecewise linear functions (Gomory–Johnson cut-generating functions). If the function is assumed to be piecewise linear and periodic, then verifying subadditivity is a finite test [**KZHW20**]. Suppose $\pi\colon \mathbb{R} \to \mathbb{R}$ is a continuous piecewise linear and $\mathbb{Z}$-periodic function, we define $\Delta\pi(x, y) = \pi(x) + \pi(y) - \pi(x + y)$, which represents the subadditive slack. Suppose π is $\mathbb{Z}$-periodic, then $\Delta\pi$ is $\mathbb{Z}^2$-periodic and we only need to consider $\Delta\pi$ over $[0, 1]^2$. In fact, due to the continuity of π, $\Delta\pi$ is also a continuous piecewise linear function. One task related to the subadditivity test is computing the minimum of $\Delta\pi$. It is clear that π is subadditive if and only if the minimum of $\Delta\pi$ is nonnegative. Therefore, subadditivity can be verified by minimizing a piecewise linear nonconvex function ($\Delta\pi$) over a convex domain ($[0, 1]^2$). This program has a very simple feasible region however the objective function is not easy to write out explicitly, especially when π has a large number of affine pieces. Unlike the traditional smooth nonlinear programming [**SP99**], the objective function in the piecewise linear setting is not everywhere twice differentiable.

The classical (naive) algorithm for the subadditivity test presented in [**KZHW20**] explores every linear piece of the function $\Delta\pi$. The naive algorithm has also been used in other subadditivity-related applications, including generating additive faces. The naive algorithm is straightforward and works well if the function π has a relatively small number of breakpoints.

In this chapter, we introduce other two types of algorithms which can be used to study subadditivity of piecewise linear functions. One is using a mixed integer formulation and the other is a customized branch and bound algorithm.

Given an optimization problem, besides heuristic approaches, mixed integer formulation is usually a good starting point of understanding the problem itself. After the formulation, it is clear to know the statistic of the problem, for example what is the objective function, what is the size of problem and how many integer variables are used. Due to the size of the mixed integer formulation and the NP-hardness of solving it, theoretically the problem is unlikely to be solved in a reasonable

time. However, it is shown that optimization solvers have become 2,000,000 times faster in the period 1990-2019 [**Mit20**, **Nem13**, **Bix12**, **BB19**], and the more speedup can be achieved with hardware enhancements. So the mixed integer formulation can at least be tried on easily accessible modern optimization solvers, especially when combining good initial solutions (from heuristics) and callback functionality. Sometimes even the most powerful solver cannot solve the problem to optimality in a reasonable time, yet it is still possible to obtain some useful information like the difficulty of the problem, the quality of solutions from heuristics and the best solution after a given amount of time. In terms of computational performance, mixed integer formulation can also be used as a baseline to compare with newly proposed algorithms [**BBM04**, **MGR12**].

In the context of piecewise linear functions, mixed integer programs have been widely used to formulate non-convex functions; for example [**Mey76**]. For a piecewise linear function, its graph can be considered as a union of polyhedra, which can be formulated using mixed integer programs. We refer interested readers to [**JL84**] for MIP representability, which characterizes sets that can be modeled as MIP problems. The idea of using MIP to formulate a piecewise linear function π is to introduce binary variables to represent different affine pieces of π. We focus on the so-called "disaggregated logarithmic" (DLog) formulation [**Vie15**] which uses the "binary encoding" technique, trying to reduce the number of binary variables.

Another promising approach to study subadditivity is a customized branch and bound algorithm. Customized branch and bound algorithms have been widely used in various optimization problems, for example job scheduling [**BBM04**, **AC91**], covering [**WHG18**], knapsack [**MGR12**, **KL10**], facility location [**BK17**] and sparse principle component analysis [**MWA06**, **BB19**]. The branch and bound technique used in a "general purpose" optimization solver usually tightens a relaxation by branching on native variables of the MIP formulation, regardless of actual meaning of integer variables in the original problem. In a customized branch and bound algorithm, branching criterion is genetically problem-specific and has a concrete practical meaning in the original problem [**WHG18**]. The flexibility of branching decision makes customized branch and bound algorithms potentially efficient. Another merit of customized branch and bound algorithm is the ability to use known powerful (problem-specific) technique while exploring the branch and bound tree. Good heuristics can be used to provide fast and good solution in a tree node [**WHG18**], which generally speed up the solving process. Bollapragada et.al [**VAB$^+$97**] combined a heuristic and the column generation technique in every node of the branch and bound tree. In sparse principle

component analysis, Berk et.al [**BB19**] takes the advantage of algebraic structure while computing upper bounds, which are not part of the bounds generation in the compared optimization solver.

Traditional branch and bound algorithms are usually applied to solving the mixed integer linear programs, which have discontinuous solution spaces. In the branching step, subproblems are generated by restricting some of the integer variables, thus yielding a problem with smaller size. In the bounding step, a convex relaxation (usually a LP relaxation) of a subproblem is solved and the optimal objective gives a dual bound for subproblem. After comparing the dual bound with the current global primal bound, a branching or pruning decision is made. After exhaustively exploring the solution space in the branch and bound tree, the optimal solution can be found.

We introduce a spatial branch and bound algorithm for studying subadditivity-related features of a piecewise linear function. In regards to one spatial branch and bound algorithm, the key step is also how to construct a convex relaxation of the original nonconvex problem. Suppose we are considering a global minimization problem, and the feasible region is convex. At each iteration of the algorithm, consider a convex relaxation of the original problem restricted to a (convex) sub-region and solve the relaxation to get a local lower bound. If the current local lower bound is larger than the best global upper bound so far, then it is impossible for the current region to contain the optimal solution, therefore no branching is needed. Otherwise, the current region can be further divided in the branching step.

Note that the efficiency of the spatial branch and bound algorithms depends on the selection rule of the sub-regions and the construction of the convex relaxation. In our sBB algorithm, the selection of sub-regions is based on the breakpoints so that all subregions form a "coarser refinement" of the two dimensional polyhedral complex $\Delta\pi$. The way to construct a convex relaxation makes use of affine functions as the under and lower estimators of the function π. One reason to use affine estimators is that computing such estimators and solving the respective convex relaxation is relatively efficient. The spatial branch and bound algorithm relies on early pruning, meaning the local lower bound is no smaller than the current global upper bound. Therefore, the tight convex relaxation (local lower bound) and early good feasible solution (global upper bound) play important roles in the algorithm. In our subadditivity test setting, a good feasible solution is easy to find. Suppose the given function is actually subadditive and $\pi(0) = 0$, then $\Delta\pi(0,0) = 0$. So $(0,0)$ is an optimal solution and no better solution will be found in the branch and bound search

tree. In terms of the convex relaxation, we can show that the best local lower bound using affine estimators can be found by solving an LP.

We implement a spatial branch and bound algorithm with several variations in our software [**KZHW20**]. Those variations use different strategies to select branching nodes and to construct convex relaxations. The spatial branch and bound algorithm can also be applied to other applications related to subadditivity, including generation of additive faces. After trying the spatial branch and bound algorithm on some cut-generating functions, we observe that it can outperform the naive algorithm and the MIP formulation on some functions with complicated structure. The main reason of the spatial branch and bound algorithm can work well is that subadditivity can be proven using the convex relaxations in certain large regions where many affine pieces of $\Delta\pi$ are contained.

2.1. Subadditive Functions

We first briefly review the single-row Gomory–Johnson model, which takes the following form:

$$x + \sum_{r \in \mathbb{R}} r\, y(r) = b, \quad b \notin \mathbb{Z}, b > 0 \tag{2.1}$$

$$x \in \mathbb{Z},\ y : \mathbb{R} \to \mathbb{Z}_+, \text{ and } y \text{ has finite support.}$$

Let $\pi\colon \mathbb{R} \to \mathbb{R}$ be a nonnegative function. Then by definition π is a valid Gomory–Johnson function if $\sum_{r\in\mathbb{R}} \pi(r)\, y(r) \geq 1$ holds for any feasible solution (x, y). Pointwise minimal $\mathbb{Z}$-periodic functions are characterized by the following conditions: (i) $\pi(0) = 0$, (ii) $\pi(x) \geq 0$, (iii) $\pi(x)+\pi(y) \geq \pi(x+y)$ (subadditivity), (iv) $\pi(x) + \pi(b - x) = 1$ (symmetry).

Subadditivity is a necessary condition of minimality, thus subadditivity of a given function π needs to be verified before applying it to generate the valid inequality $\sum_{r\in\mathbb{R}} \pi(r)\, y(r) \geq 1$. Even if the generated cut is theoretically valid, the cutting plane algorithm can still have poor computational performance due to floating-point arithmetic. For example, it has been shown that Mixed Integer Rounding (MIR) procedure can frequently generate infeasible cuts [**Mar09**]. There are also studies on how to "safely" generate valid MIR cuts or their variants by utilizing the bounds on variables [**NS04**, **BS08**, **DGL10**, **DGG10**].

We introduce the notion *near-subadditivity*. Nearly subadditive functions are those functions satisfying $\pi(x) + \pi(y) - \pi(x + y) \geq -\epsilon$ for some "small" $\epsilon > 0$. Here we makes no claims on

the effectiveness of nearly subadditive functions in computational practice. Instead, we just briefly discuss a potential application of nearly subadditive functions and we focus on how to verify near-subadditivity for piecewise linear functions.

One nearly subadditive function can be considered as a perturbed cut generating function, and the perturbation could mimic the "floating-point arithmetic error". Then the following theorem claims that nearly subadditive functions can generate valid inequalities, which also cut off the "current fractional solution".

PROPOSITION 2.1.1. *Let $\pi\colon \mathbb{R} \to \mathbb{R}$ be a $\mathbb{Z}$-periodic function, and satisfying: (i) $\pi(0) = 0$, (ii) $\pi(b) = 1$, (iii) $\pi(x) + \pi(y) - \pi(x+y) \geq -\epsilon$, for some $\epsilon > 0$. Define $\pi_\epsilon(x) = \pi(x) + \epsilon$ for all $x \in \mathbb{R}$. Then the following inequality is valid for any feasible solution (x, y) to equation* (1).

$$\sum_{r \in \mathbb{R}} \pi_\epsilon(r)\, y(r) \geq 1$$

PROOF. From condition (iii) we know that π_ϵ is a subadditive function. For any solution (x, y) of Equation (2.1), it holds that

$$\begin{aligned}
1 &= \pi(b) \\
&= \pi_\epsilon(b) - \epsilon \\
&= \pi_\epsilon(x + \sum_{r \in \mathbb{R}} r\, y(r)) - \epsilon \\
&\leq \pi_\epsilon(x) - \epsilon + \sum_{r \in \mathbb{R}} \pi_\epsilon(r)\, y(r) \\
&= \pi(x) + \sum_{r \in \mathbb{R}} \pi_\epsilon(r)\, y(r) \\
&= \sum_{r \in \mathbb{R}} \pi_\epsilon(r)\, y(r).
\end{aligned}$$

The inequality is implied by the subadditivity of π_ϵ, and the last step is derived from $\pi(0) = 0$ and π is $\mathbb{Z}$-periodic. □

Note that near-subadditivity implies subadditivity, so it is "easier" to verify the former. As our experiment results in section 2.5 indicate, verifying near-subadditivity can have much shorter running time than verifying (strict) subadditivity using certain algorithms. Then it might be more time-efficient to apply near-subadditivity to generate fast cuts. To put it in a concrete case, let

π be a piecewise linear function with potentially large number of breakpoints. Suppose verifying subadditivity of π takes long time but verifying near-subadditivity is relatively efficient, then π can still be used to generate cuts based on Proposition 2.1.1.

In the remaining of the chapter, we restrict ourselves to $\mathbb{Z}$-periodic continuous functions and how to efficiently check (near) subadditivity.

2.2. Naive Subadditivity Test

We review the naive method to check the minimality (thus subdditivity) of a given piecewise linear function; see the function `minimality_test` and `subadditivity_test` in our software [**KZHW20**]. The function `minimality_test(`π`)` implements a fully automatic test to check whether a given piecewise linear function π is minimal. As the most important component in the function `minimality_test(`π`)`, the function `subadditivity_test(`π`)` checks subadditivity of π, by using the affine linearity in faces of $\Delta\mathcal{P}$. As we will see later, the time complexity of the minimality test mainly depends on the number of breakpoints. Using the same notation, we assume that the set of breakpoints in $[0,1]$ is $B = \{0 = a_0, a_1, \ldots, a_{n-1}, a_n = 1\}$.

To discuss the time complexity of the algorithm, we first explain the time complexity of the function evaluation $\pi(x)$ given an input x in our implementation. Since there are n affine pieces defined on $[0,1]$, we need to find out the affine piece containing the input x, which can be done by a binary search. After that, plugging x into the corresponding affine function gives the function value $\pi(x)$. Therefore, the time complexity of the function evaluation is $\Theta(\log n)$. Note that the caching technique can speedup certain function evaluations, for example, caching all function values at the breakpoints (which is exactly our implementation). But in general, function evaluation needs to go through a binary search.

In order to test minimality, we need to first check that the range of the function stays in $[0,1]$ and $\pi(0) = 0$. Since we assume the function is continuous and piecewise linear with finitely many breakpoints, only function values at the breakpoints (i.e. $\pi(a_i)$) need to be checked. In regards to the symmetry condition, it suffices to show that the function $h(x) \coloneqq \pi(x) + \pi(b - x) = 1$ for any real number x. Note that $h(x)$ is also a $\mathbb{Z}$-periodic continuous piecewise linear function, and it is also symmetric about $x = \frac{b}{2}$. Then, the condition $h(x) = 1$ only needs to be checked on the set of breakpoints of π, namely B. Therefore, other than subadditivity test, other conditions of minimality can be verified in $\Theta(n \log n)$ time.

In terms of subadditivity, we only consider $\Delta\pi$ over $[0,1]^2$ (two dimensional polyhedral complex $\Delta\mathcal{P}$). Due to the affine linearity of $\Delta\pi$ in every face F in $\Delta\mathcal{P}$, it suffices to check $\Delta\pi(u,v) \geq 0$ for every $(u,v) \in \text{vert}(F)$. As for the diagram of $\Delta\mathcal{P}$, we start with a square complex $I = J = [0,1], K = [0,2]$, which is exactly $[0,1]^2$, and then refine I, J, K based on the set of breakpoints B and $B \cup (B+1)$. Based on the refinement of I, J using breakpoints B, there are $\Theta(n^2)$ vertices. Similarly, the refinements of I, K and J, K both give $\Theta(n^2)$ vertices. Some of the vertices might be duplicated, and using the symmetry of $\Delta\pi$ can reduce the number of vertices in half. Overall, it is not hard to see that there are $\Theta(n^2)$ vertices on which $\Delta\pi$ needs to be evaluated. Thus, the naive subadditivity test has time complexity $\Theta(n^2 \log n)$.

To summarize, since the subadditivity test is the dominant part, the time complexity of the implementation of `minimality_test` is $\Theta(n^2 \log n)$. In fact, the naive algorithm has been used in other applications related to the subadditivity slack. The function `generate_maximal_additive_faces` generates maximal additive faces, defined as those which are not contained in any higher dimensional additive faces. The maximal additive faces then will be used in `extremality_test` to verify whether the function is extreme.

2.3. Mixed Integer Program Formulation

In this section, we introduce alternative methods to compute the minimum of $\Delta\pi$ by solving mixed integer programs.

First we introduce notations of parameters used in mixed integer program formulations. Following notations defined in the previous chapter, we denote $B = [a_0, a_1, \ldots, a_n]$ as a list containing all breakpoints. We use "list" instead of "set" because mixed integer formulations rely on the order of breakpoints. Define the list $V = [v_0, v_1, \ldots, v_n]$ to be the function values on B. Specifically, $v_i = \pi(a_i)$ for all $i \in \{0, 1, \ldots, n\}$. From continuity of π we know that the sets B and V uniquely define the function π, therefore the minimum of $\Delta\pi$ is uniquely determined by B and V. To conveniently express mixed integer formulations, we also need to define lists of extended breakpoints and function values for $x+y \in [0,2]$. Define $B' = [a_0, a_1, \ldots, a_n, 1+a_1, 1+a_2, \ldots, 1+a_n] = [a'_0, \ldots, a'_{2n}]$ and $V' = [\pi(a_0), \pi(a_1), \ldots, \pi(a_n), \pi(1+a_1), \pi(1+a_2), \ldots, \pi(1+a_n)] = [v'_0, \ldots, v'_{2n}]$. We use B, B' and V, V' as parameters to formulate mixed integer programs.

We review three formulation techniques (namely Convex Combination, Multiple Choice, Disaggregated Logarithmic [**Vie15**]), which are used to model univariate piecewise linear functions.

Then we apply them to solve our $\Delta\pi$ minimization problem. The idea of all three formulations is using the convex combination of B or B' to express x, y or $x+y$, and adding binary variables to ensure that $\pi(x), \pi(y), \pi(x+y)$ can also be expressed as the convex combination of V or V'.

2.3.1. Convex Combination Formulation.

$$\textbf{minimize} \quad vx + vy - vz \tag{2.2}$$

$$\textbf{subject to} \quad x + y = z \tag{2.3}$$

$$\sum_{i=0}^{n} \lambda_i^x a_i = x, \qquad \sum_{i=0}^{n} \lambda_i^y a_i = y, \qquad \sum_{j=0}^{2n} \lambda_j^z a'_i = z \tag{2.4}$$

$$\sum_{i=0}^{n} \lambda_i^x v_i = vx, \qquad \sum_{i=0}^{n} \lambda_i^y v_i = vy, \qquad \sum_{j=0}^{2n} \lambda_j^z v'_i = vz \tag{2.5}$$

$$\sum_{i=0}^{n} \lambda_i^x = \sum_{i=0}^{n} \lambda_i^y = \sum_{j=0}^{2n} \lambda_j^z = 1 \tag{2.6}$$

$$\sum_{i=1}^{n} b_i^x = \sum_{i=1}^{n} b_i^y = \sum_{j=1}^{2n} b_j^z = 1 \tag{2.7}$$

$$\lambda_{i-1}^x + \lambda_i^x \geq b_i^x \quad i = 1, 2, \ldots, n \tag{2.8}$$

$$\lambda_{i-1}^y + \lambda_i^y \geq b_i^y \quad i = 1, 2, \ldots, n \tag{2.9}$$

$$\lambda_{j-1}^z + \lambda_j^z \geq b_j^z \quad j = 1, 2, \ldots, 2n \tag{2.10}$$

$$x, y, z, vx, vy, vz \in \mathbb{R} \tag{2.11}$$

$$\lambda_i^x, \lambda_i^y, \lambda_j^z \in \mathbb{R}_+ \quad i = 0, 1, \ldots, n \quad j = 0, 1, \ldots, 2n \tag{2.12}$$

$$b_i^x, b_i^y, b_j^z \in \{0, 1\} \quad i = 1, 2, \ldots, n \quad j = 1, 2, \ldots, 2n \tag{2.13}$$

In the above formulation, we use vx, vy, vz to represent the function values $\pi(x), \pi(y), \pi(x+y)$. The objective function, as the subadditivity slack, is minimized. Using convex combinations of breakpoints B, B' and function values V, V' to express x, y, z and vx, vy, vz is written as constraints (2.4), (2.5), (2.6). The binary variables b_i^x, b_i^y, b_j^z are introduced to ensure that the convex combinations are valid. We explain the reason for variable x, and the variables y, z follow the same logic. Based on constraint (2.7), exactly one variable $b_{i_0}^x$ is 1, representing $x \in [a_{i_0-1}, a_{i_0}]$. If $i \neq i_0$, meaning $b_i^x = 0$, then the constraint (2.8) is redundant due to the non-negativity of λ^x. If $i = i_0$, then constraints (2.6) and (2.8) force $\lambda_{i-1}^x + \lambda_i^x = 1$, which also implies that all other

λ^x variables are fixed to 0. Then, x can be expressed as a convex combination of breakpoints a_{i_0-1} and a_{i_0} given $x \in [a_{i_0-1}, a_{i_0}]$: $x = \lambda^x_{i_0-1} a_{i_0-1} + \lambda^x_{i_0} a_{i_0}$. The function π is an affine function over $[a_{i_0-1}, a_{i_0}]$, so we can also express vx as a conic combination of v_{i_0-1} and v_{i_0} using the same coefficients: $vx = \lambda^x_{i_0-1} v_{i_0-1} + \lambda^x_{i_0} v_{i_0}$. Note that the convex combination is not unique if $x \in B$.

This convex combination formulation of minimizing $\Delta\pi$ has approximately $4n$ continuous variables, $4n$ binary variables and $4n$ constraints, given the function π has $n+1$ breakpoints.

2.3.2. Multiple Choice Formulation.

$$\textbf{minimize} \quad vx + vy - vz \tag{2.14}$$

$$\textbf{subject to} \quad x + y = z \tag{2.15}$$

$$\sum_{i=0}^{n} \lambda^x_i a_i = x, \qquad \sum_{i=0}^{n} \lambda^y_i a_i = y, \qquad \sum_{j=0}^{2n} \lambda^z_j a'_i = z \tag{2.16}$$

$$\sum_{i=0}^{n} \lambda^x_i v_i = vx, \qquad \sum_{i=0}^{n} \lambda^y_i v_i = vy, \qquad \sum_{j=0}^{2n} \lambda^z_j v'_i = vz \tag{2.17}$$

$$\sum_{i=0}^{n} \lambda^x_i = \sum_{i=0}^{n} \lambda^y_i = \sum_{j=0}^{2n} \lambda^z_j = 1 \tag{2.18}$$

$$\sum_{i=1}^{n} b^x_i = \sum_{i=1}^{n} b^y_i = \sum_{j=1}^{2n} b^z_j = 1 \tag{2.19}$$

$$\lambda^x_i = \gamma^x_{i,i} + \gamma^x_{i,i+1}, \quad b^x_i = \gamma^x_{i-1,i} + \gamma^x_{i,i} \quad i = 1, \ldots, n \tag{2.20}$$

$$\lambda^y_i = \gamma^y_{i,i} + \gamma^y_{i,i+1}, \quad b^y_i = \gamma^y_{i-1,i} + \gamma^y_{i,i} \quad i = 1, \ldots, n \tag{2.21}$$

$$\lambda^z_j = \gamma^z_{j,j} + \gamma^z_{j,j+1}, \quad b^z_j = \gamma^z_{j-1,j} + \gamma^z_{j,j} \quad j = 1, \ldots, 2n \tag{2.22}$$

$$\gamma^x_{0,0} = \gamma^y_{0,0} = \gamma^z_{0,0} = \gamma^x_{n,n+1} = \gamma^y_{n,n+1} = \gamma^z_{2n,2n} = 0 \tag{2.23}$$

$$x, y, z, vx, vy, vz \in \mathbb{R} \tag{2.24}$$

$$\lambda^x_i, \lambda^y_i, \lambda^z_j \in \mathbb{R}_+ \quad i = 0, 1, \ldots, n \quad j = 0, 1, \ldots, 2n \tag{2.25}$$

$$\gamma^x_{i,i}, \gamma^x_{i,i+1}, \gamma^y_{i,i}, \gamma^y_{i,i+1}, \gamma^z_{j,j}, \gamma^z_{j,j+1}, \in \mathbb{R}_+ \quad i = 0, 1, \ldots, n \quad j = 0, 1, \ldots, 2n \tag{2.26}$$

$$b^x_i, b^y_i, b^z_j \in \{0, 1\} \quad i = 1, 2, \ldots, n \quad j = 1, 2, \ldots, 2n \tag{2.27}$$

The multiple choice formulation is based on the convex combination formulation. The auxiliary γ variables are introduced to be associated with binary variables b in every affine linear piece.

After replacing all λ variables with γ variables, there are approximately $8n$ continuous variables, $4n$ binary variables and $8n$ constraints in the multiple choice formulation. It can be proven that the multiple choice formulation is stronger than the convex combination formulation.

2.3.3. Disaggregated Logarithmic Formulation.

$$\textbf{minimize} \quad vx + vy - vz \tag{2.28}$$

$$\textbf{subject to} \quad x + y = z \tag{2.29}$$

$$\sum_{i=0}^{n} \lambda_i^x a_i = x, \quad \sum_{i=0}^{n} \lambda_i^y a_i = y, \quad \sum_{j=0}^{2n} \lambda_j^z a_i' = z \tag{2.30}$$

$$\sum_{i=0}^{n} \lambda_i^x v_i = vx, \quad \sum_{i=0}^{n} \lambda_i^y v_i = vy, \quad \sum_{j=0}^{2n} \lambda_j^z v_i' = vz \tag{2.31}$$

$$\sum_{i=0}^{n} \lambda_i^x = \sum_{i=0}^{n} \lambda_i^y = \sum_{j=0}^{2n} \lambda_j^z = 1 \tag{2.32}$$

$$\lambda_i^x = \gamma_{i,i}^x + \gamma_{i,i+1}^x, \quad \lambda_i^y = \gamma_{i,i}^y + \gamma_{i,i+1}^y, \quad i = 0, 1, \ldots, n \tag{2.33}$$

$$\lambda_j^z = \gamma_{j,j}^z + \gamma_{j,j+1}^z, \quad j = 0, 1, \ldots, 2n \tag{2.34}$$

$$\gamma_{0,0}^x = \gamma_{0,0}^y = \gamma_{0,0}^z = \gamma_{n,n+1}^x = \gamma_{n,n+1}^y = \gamma_{2n,2n}^z = 0 \tag{2.35}$$

$$\sum_{i=1}^{n} (\gamma_{i-1,i}^x + \gamma_{i,i}^x) f(i)_k = s_k^x, \quad k = 1, 2, \ldots, m \tag{2.36}$$

$$\sum_{i=1}^{n} (\gamma_{i-1,i}^y + \gamma_{i,i}^y) f(i)_k = s_k^y, \quad k = 1, 2, \ldots, m \tag{2.37}$$

$$\sum_{j=1}^{2n} (\gamma_{j-1,j}^z + \gamma_{j,j}^z) f(j)_{k'} = s_{k'}^z, \quad k' = 1, 2, \ldots, m' \tag{2.38}$$

$$x, y, z, vx, vy, vz \in \mathbb{R} \tag{2.39}$$

$$\lambda_i^x, \lambda_i^y, \lambda_j^z \in \mathbb{R}_+, \quad i = 0, 1, \ldots, n, \quad j = 0, 1, \ldots, 2n \tag{2.40}$$

$$\gamma_{i,i}^x, \gamma_{i,i+1}^x, \gamma_{i,i}^y, \gamma_{i,i+1}^y \in \mathbb{R}_+, \quad i = 0, 1, \ldots, n \tag{2.41}$$

$$\gamma_{j,j}^z, \gamma_{j,j+1}^z, \in \mathbb{R}_+, \quad j = 0, 1, \ldots, 2n \tag{2.42}$$

$$s_k^x, s_k^y, s_{k'}^z \in \{0, 1\}, \quad k = 1, 2, \ldots, m, \quad k' = 1, 2, \ldots, m' \tag{2.43}$$

The disaggregated logarithmic formulation uses the idea of "binary encoding" to reduce the number of binary variables. Let $m = \lceil \log_2 n \rceil$, $m' = \lceil \log_2 2n \rceil = 1 + m$, $f(i)$ be the binary encoding of i, and $f(i)_k$ be the kth digit of $f(i)$. The λ and γ variables follow the same logic as in the multiple choice formulation. In terms of the binary variables s, we only study the case in terms of x. The key constraints (2.36) mean that the convex combination of binary vector (i.e. $f(i)$) is also a binary vector (i.e. s^x), which implies that exactly one $(\gamma^x_{i_0-1,i_0} + \gamma^x_{i_0,i_0})$ is 1 and all others are 0. Therefore, $\lambda_{i_0-1} = \gamma_{i_0-1,i_0}$, $\lambda_{i_0} = \gamma_{i_0,i_0}$, $\lambda_{i_0-1} + \lambda_{i_0} = 1$ and all other λ and γ variables are zero.

After replacing all γ variables, the disaggregated logarithmic formulation has approximately $8n$ continuous variables, $3 \log_2 n$ binary variables and $3 \log_2 n$ constraints. Note that the disaggregated logarithmic formulation is stronger than the other two formulations. It also has fewer binary variables, which typically lead to shorter running time using a MIP solver. In regards to our computation of the $\Delta\pi$ minimum, we only implement the disaggregated logarithmic formulation and solve it using a commercial solver in order to get the best solving time.

The mixed integer formulation has its own limitations. First, it uses floating point computation, and the solved optimal objective is an "estimate" of the actual optimal objective within a small tolerance. It is possible that the reported optimal objective is a negative value which is very close to zero, then rigorously speaking, the subadditivity cannot be proven and near-subadditivity should be used instead. To make the result more precise, it is possible to tighten certain tolerances in the solver. However, tightening tolerances will usually make the solving time longer and it cannot resolve the precision caveat completely. As we will see later, the mixed integer formulation has limited applications in other subadditivity related studies. For example, unlike the naive algorithm, the MIP formulation cannot be used to generate additive faces nor to verify extremality.

Despite the limitations of mixed integer formulations, it still worths to consider them (at least the strongest one) as the baseline algorithm, which can be solved "out-of-the-box". As the development of modern optimization solvers and the enhancement of hardware are expected in the future, the MIP formulation in subadditivity study can also be considered as a test instance to measure speedups. Simply formulating the $\Delta\pi$ minimization problem can help us understand the size/difficulty of the problem, and filter certain optimization techniques quickly. For instance, the number of variables and the number of constraints are not exponential, thus techniques like column generation are unlikely to be applicable. In our computational results, we only compare the disaggregated logarithmic formulation with naive algorithm and spatial branch and bound algorithm.

2.4. Spatial Branch and Bound Method

In this section, we explain in detail the spatial branch and bound algorithm, especially how to construct convex relaxations. The idea of the spatial branch and bound algorithm is proving subadditivity fast in certain large regions using convex relaxations. Given a $\mathbb{Z}$-periodic continuous piecewise linear function π, the algorithm can be used to compute the minimum of $\Delta\pi$, verifying whether π is (nearly) subadditive, generating additive vertices and additive faces.

Suppose $I, J \subset [0,1]$ and $K \subset [0,2]$, and follow the definition of the complex $\Delta\mathcal{P}$, we define $F = F(I,J,K) = \{\, (x,y) \in \mathbb{R} \times \mathbb{R} : x \in I,\, y \in J,\, x+y \in K \,\}$ and $\mathrm{vert}(F)$ to be the vertices of F. In our sBB algorithm, I, J, K can be any subintervals and F does not have be a face of $\Delta\mathcal{P}$. To prove a function π is subadditive, if suffices to show that $\Delta\pi(x,y) \geq 0$ for all $(x,y) \in F([0,1],[0,1],[0,2])$.

2.4.1. Lower and Upper Estimators. We first introduce the definitions of lower and upper estimators which will be used in the spatial branch and bound algorithm.

Definition 2.4.1. *Given a function π defined over the interval $I = [a,b]$, we call a function $\underline{\pi}^I$ a lower estimator of π if $\pi(x) \geq \underline{\pi}^I(x)$ for all $x \in I$. We call a function $\overline{\pi}^I$ an upper estimator of π if $\pi(x) \leq \overline{\pi}^I(x)$ for all $x \in I$.*

Given $I, J \subset [0,1]$ and $K \subset [0,2]$, suppose we know that $\underline{\pi}^I, \underline{\pi}^J$ are lower estimators of π over the interval I, J, and $\overline{\pi}^K$ is an upper estimator of π over the interval K. It is clear that $\Delta\pi(x,y) \geq \underline{\pi}^I(x) + \underline{\pi}^J(y) - \overline{\pi}^K(x+y)$ for all $(x,y) \in F(I,J,K)$. If the minimum of $\underline{\pi}^I(x) + \underline{\pi}^J(y) - \overline{\pi}^K(x+y)$ is already nonnegative, then we can conclude π is subadditive in $F(I,J,K)$. For each (convex) region $F(I,J,K)$, instead of computing the minimum of $\Delta\pi$, we consider the relaxation:

$$\begin{aligned} \textbf{minimize} \quad & \underline{\pi}^I(x) + \underline{\pi}^J(y) - \overline{\pi}^K(x+y) \\ \textbf{subject to} \quad & (x,y) \in F(I,J,K) \end{aligned}$$

If the estimators are carefully chosen, then the above relaxation is convex. One natural question to ask is how to choose the estimators. In general they should be "easy" to construct and solving the above relaxation should also be "fast". In our spatial branch and bound algorithm, we choose estimators of π such that the optimal solution of the convex relaxation occurs in $\mathrm{vert}(F(I,J,K))$.

2.4.2. Constant Estimators. The simplest lower (upper) estimators are the constant lower (upper) bounds. It is not hard to show the following lemma.

LEMMA 2.4.1. *Let $\pi\colon \mathbb{R} \to \mathbb{R}$ be a continuous piecewise linear $\mathbb{Z}$-periodic function, and $I, J \subset [0,1]$ and $K \subset [0,2]$. Suppose there exist $\alpha_I, \alpha_J, \beta_K \in \mathbb{R}$ such that $\pi(x) \geq \alpha_I$ for $x \in I$, $\pi(x) \geq \alpha_J$ for $x \in J$, and $\pi(x) \leq \beta_K$ for $x \in K$. Then $\Delta\pi(x,y) \geq \alpha_I + \alpha_J - \beta_K$ for $(x,y) \in F(I,J,K)$.*

We briefly discuss the time complexity of constructing the constant estimators. Following the previous notation, we assume the set of breakpoints of π is $B = \{a_0, ..., a_n\}$. Given subintervals I, J, K and the region $F(I,J,K)$, constructing the constant upper and lower estimators is straightforward. Due to continuity and piecewise linearity of π, it suffices to loop through the function value at every breakpoint (and two endpoints) contained in the intervals I, J, K. Assuming function evaluation takes $\log n$ time, then constructing constant estimators together with solving the corresponding convex relaxation has near-linear time complexity $O(n \log n)$. On the other hand, it is possible that there are quadratically many vertices (of the complex $\Delta\mathcal{P}$) contained in $F(I,J,K)$. Ideally, to prove the minimum of $\Delta\pi$ is nonnegative over a certain region $F(I,J,K)$, using constant estimators could be faster than computing values of $\Delta\pi$ of every vertex contained in $F(I,J,K)$.

2.4.3. Affine Estimators with Fixed Slope Values. Computing constant estimators is fast and straightforward, and they can be generalized to affine estimators.

DEFINITION 2.4.2. *Given a function π defined over the interval $I = [a,b]$, and a fixed slope value s, we call an affine function $\underline{\pi}^I$ (or $\overline{\pi}^I$) a slope-s affine lower/upper estimator if it is a lower/upper estimator of π and has slope value s.*

Observe that constant estimators are exactly slope-0 affine estimators. If the slope value s is fixed, the "best" slope-s affine lower/upper estimator is trivial to construct. It suffices to choose the largest/smallest intercepts while maintaining valid lower/upper estimators. In regards to the complexity of constructing slope-s affine estimators, we also need to loop through every breakpoint (and two endpoints) contained in each interval, which takes $O(n \log n)$ as well.

In terms of the strength of the convex relaxation using affine estimators, it is not hard show that the optimal solution occurs in $\mathrm{vert}(F(I,J,K))$ as Lemma 2.4.2.

LEMMA 2.4.2. *Let $\pi\colon \mathbb{R} \to \mathbb{R}$ be a continuous piecewise linear $\mathbb{Z}$-periodic function, and $I, J \subset [0,1]$ and $K \subset [0,2]$. Suppose there exist affine estimators $\underline{\pi}^I, \underline{\pi}^J, \overline{\pi}^K$. Then for all $(x,y) \in F(I,J,K)$ it holds that*

$$\begin{aligned}\Delta\pi(x,y) &\geq \min_{(x,y)\in F(I,J,K)} \underline{\pi}^I(x) + \underline{\pi}^J(y) - \overline{\pi}^K(x+y) \\ &= \min_{(x,y)\in \mathrm{vert}(F(I,J,K))} \underline{\pi}^I(x) + \underline{\pi}^J(y) - \overline{\pi}^K(x+y)\end{aligned}$$

PROOF. Note that the affine linearity of $\underline{\pi}^I, \underline{\pi}^J, \overline{\pi}^K$ implies the affine linearity of $\underline{\pi}^I(x)+\underline{\pi}^J(y)-\overline{\pi}^K(x+y)$ on $F(I,J,K)$. Therefore, the minimum of $\underline{\pi}^I(x) + \underline{\pi}^J(y) - \overline{\pi}^K(x+y)$ only depends on the values at $\mathrm{vert}(F(I,J,K))$. Based on the definition of lower and upper estimators, we conclude the desired result. □

Heuristically, we choose the fixed slope value to be the slope of the line segment connecting two endpoints of the interval I or J or K.

2.4.4. Affine Estimators. To construct the strongest convex relaxation, three affine estimators should not be constructed separately. We can compute all three affine estimators simultaneously which can produce the best lower bound for $\Delta\pi$ on $F(I,J,K)$, by solving some linear program.

Given a continuous piecewise linear function π, and the breakpoint set B. Let $B' = B \cup (B+1)$ be the extended breakpoint set. Fix $I, J \subset [0,1]$ and $K \subset [0,2]$ and we assume all endpoints of I, J, K belong to B'. We make this assumption so that the linear program can be written cleanly, and the assumption is easy to fulfill by the branching criterion that we will explain later.

Suppose $\underline{\pi}^I, \underline{\pi}^J, \overline{\pi}^K$ are affine estimators defined in Definition 2.4.1, and let $\underline{\pi}^I(x) = s_I x + t_I$, $\underline{\pi}^J(x) = s_J x + t_J$ and $\overline{\pi}^K(x) = s_K x + t_K$. Define the inputs as the breakpoints $B \cap I$, $B \cap J$ and $B' \cap K$, values at those breakpoints, and $\mathrm{vert}(F(I,J,K))$. The s variables and t variables used in the linear program represent slope values and intercepts of affine estimators respectively. We use variable m as the minimum of $\underline{\pi}^I(x) + \underline{\pi}^J(y) - \overline{\pi}^K(x+y)$ and this can be taken care of by maximizing m. We write the linear program as follows, and solving it can produce the best lower bound of $\Delta\pi$ and three corresponding affine estimators.

$$\text{maximize} \quad m \tag{2.44}$$

$$\text{subject to} \quad m \le s_I x + t_I + s_J y + t_J - s_K(x+y) - t_K \ \forall (x,y) \in \text{vert}(F) \tag{2.45}$$

$$\pi(b_I) \ge s_I b_I + t_I \quad \forall\, b_I \in B \cap I \tag{2.46}$$

$$\pi(b_J) \ge s_J b_J + t_J \quad \forall\, b_J \in B \cap J \tag{2.47}$$

$$\pi(b_K) \le s_K b_K + t_K \quad \forall\, b_K \in B' \cap K \tag{2.48}$$

$$s_I, s_J, s_K, t_I, t_J, t_K, m \in \mathbb{R} \tag{2.49}$$

THEOREM 2.4.1. *Let $\pi\colon \mathbb{R} \to \mathbb{R}$ be a continuous piecewise linear $\mathbb{Z}$-periodic function, and $I, J \subset [0,1]$ and $K \subset [0,2]$. Then the linear program (2.44-2.49) is always feasible and bounded, and the optimal objective function value m^* is a lower bound of $\Delta\pi$ on $F(I,J,K)$. Furthermore, for any three affine estimators $\underline{\pi}^I, \underline{\pi}^J, \overline{\pi}^K$, it holds that*

$$m^* \ge \min_{(x,y)\in F(I,J,K)} \underline{\pi}^I(x) + \underline{\pi}^J(y) - \overline{\pi}^K(x+y)$$

PROOF. Since π is a continuous piecewise linear function defined on a bounded interval, then $s_I = s_J = s_K = 0$, $t_I = \min_{x\in I} \pi(x)$, $t_J = \min_{x\in J} \pi(x)$, and $t_K = \max_{x\in K} \pi(x)$ is a feasible solution. On the other hand, $m \le \max_{x\in I} \pi(x) + \max_{x\in J} \pi(x) - \min_{x\in K} \pi(x)$, therefore the program is bounded. The optimal objective function value m^* is a lower bound of $\Delta\pi$ on $F(I,J,K)$ is due to the fact that $s_I x + t_I, s_J x + t_J$ are lower estimators of π over I, J and $s_K x + t_K$ is an upper estimator of π over K. Based on the formulation of the linear program, we know that m^* is the best lower bound generated by affine estimators of π. □

The linear program has 7 free variables and the number of constraints depends on how many breakpoints are contained in I, J, K. Let $M = M(I,J,K)$ be total number of breakpoints contained in I, J, K. Note that M is linear in n. If n is large, then a large-scale linear program needs to be solved. Since solving an linear program is needed, it is unclear what the time complexity is. In the spatial branch and bound algorithm described in the next section, we need to solve a large number of such LPs if the best lower bounds are computed. In general, solving a large number of large-scale LPs can be time-consuming. So we tend to use affine estimators with fixed values if M is large to avoid solving too many LPs.

We have in total three types of estimators for generating the lower bound of $\Delta\pi$, the constant estimators, affine estimators by solving an LP and affine estimators with fixed slope values. Affine estimators by solving an LP can produce the best lower bound of $\Delta\pi$ but it is also the most expensive one to construct. Computing constant estimators and computing affine estimators with fixed slope values have the same time complexity. Note that using affine estimators with fixed slope values together with constant estimators can produce better bounds than using constant estimators alone.

2.4.5. Spatial Branch and Bound Tree. We introduce the spatial branch and bound algorithm, which can be used to compute the minimum of $\Delta\pi$, therefore prove or disprove the subadditivity of π. The inputs are breakpoints of π: $B = [0 = a_0, a_1, \dots, a_n = 1]$ and the corresponding function values: $V = [\pi(a_0), \pi(a_1), \dots, \pi(a_n)]$. The output is the minimum of $\Delta\pi$. We use B', V' to denote the extended breakpoints and function values the same way as in Section 2.3.

In the branch and bound tree, every node $N = N(I, J, K)$ is associated with one region $F = F(I, J, K)$ where $I, J \subset [0, 1]$ and $K \subset [0, 2]$. The root node corresponds to the region $F(I, J, K)$ defined by $I = J = [0, 1]$ and $K = [0, 2]$. For every node $N(I, J, K)$ we consider the following:

- **Estimators**: Use affine estimators $\underline{\pi}^I, \underline{\pi}^J, \overline{\pi}^K$ introduced in previous section.
- **Local lower bound**: Using affine estimators implies that $\underline{\pi}^I(x) + \underline{\pi}^J(y) - \overline{\pi}^K(x + y)$ is affine on $F(I, J, K)$, therefore $\min_{(x,y)\in \mathrm{vert}(F(I,J,K))} \underline{\pi}^I(x) + \underline{\pi}^J(y) - \overline{\pi}^K(x + y)$ is the local lower bound of $\Delta\pi$ on $F(I, J, K)$.
- **Global upper bound**: $\min_{(x,y)\in \mathrm{vert}(F(I,J,K))} \Delta\pi(x, y)$ is one global upper bound of $\Delta\pi$. There are at most 6 vertices of $F(I, J, K)$, therefore computing global upper bound in one node is relatively cheap.
- **Pruned by bounds**: If the local lower bound is no less than the current global upper bound, then no branching is needed since no better solutions can be found within $F(I, J, K)$.
- **Pruned by indivisibility**: If there is no breakpoint contained in the interior of I, J, K, then $F(I, J, K)$ is actually one face of the complex $\Delta\mathcal{P}$; see Remark 2.4.1. In this case, $\Delta\pi$ is an affine function on $F(I, J, K)$ and it is uniquely determined by the function values at $\mathrm{vert}(F(x, y, z))$ and no branching is needed.

- **Branching**: If the current node cannot be pruned by bounds or indivisibility, choose a breakpoint b contained in the interior of I or J or K, and divide the region $F(I, J, K)$ by the line $x = b$, or $y = b$, or $x + y = b$. We make sure that the chosen breakpoint b belongs to B (or B' if dividing line is $x + y = b$). For example, without loss of generality, suppose $b = a_i \in I = [a_s, a_t]$ where $s < i < t$. Then the left child and right child of the current nodes are $N([a_s, a_i], J, K)$ and $N([a_i, a_t], J, K)$.
- **Node selection**: Three selection strategies: DFS, BFS and Best Bound.

REMARK 2.4.1. *Based on the branching principle, all endpoints of I, J, K are breakpoints of π for every node in the tree, thus the region $F(I, J, K)$ of every node is always a union of some faces in $\Delta\mathcal{P}$. If the region is indivisible, then $F(I, J, K)$ is one face of the complex $\Delta\mathcal{P}$. Compared to all faces in $\Delta\mathcal{P}$, regions of all leaf nodes can be considered as a "coarser refinement" of the square $[0, 1]^2$; see example in Figure 2.1.*

Note that, in the branching step, each time only one interval of I, J, K will be divided into two subintervals and other two intervals will pass to children. Therefore the spatial branch and bound tree is a binary tree. In our implementation, the eligible interval, which contains at least one breakpoint in the interior, with largest length is selected, and the mid-indexed breakpoint is selected as the dividing endpoint. Specifically, if $I = [a_s, a_t]$ where $t - s > 1$ is selected as the interval to be divided, then it will be divided into subintervals $[a_s, a_{\lfloor (s+t)/2 \rfloor}]$ and $[a_{\lfloor (s+t)/2 \rfloor}, a_t]$.

Next, we introduce the formal spatial branch and bound algorithm for computing the minimum of $\Delta\pi$.

initialization:

set global upper bound $U \leftarrow \infty$

set unvisited node list $S \leftarrow \{N([0, 1], [0, 1], [0, 2])\}$

while S is not empty **do**

pick one node $N(I, J, K) \in S$ based on traversing strategy and remove the node from S.

compute the upper bound $u_N = \min_{(x,y)\in \mathrm{vert}(F(I,J,K))} \Delta\pi(x, y)$.

if $u_N < U$ **then**

Update $U \leftarrow u_N$.

end if

if $N(I, J, K)$ is not divisible **then**

Pruned by indivisibility. Continue.

end if

compute the local lower bound l_N from affine estimators.

if $l_N \geq U$ **then**

Pruned by bounds. Continue.

else

Branching: put left and right child nodes N_l, N_r to S.

end if

end while

Return the minimum of $\Delta\pi = U$.

There are some variants of the algorithm. In the while loop, each time we pick one node from the unvisited node list. We can choose the depth first search, breadth first search or the best bound principle. As mentioned in the previous section, we have three choices of affine estimators: constant estimators, affine estimators with fixed slope values and affine estimators by solving an LP. It is also possible to use a combination of these estimators. For example, one algorithm in our computational experiment uses 30 as a threshold for the maximum LP size, meaning if there are more than 30 constraints in the LP, then we don't solve it and use the constant estimators and affine estimators with fixed slope values instead. Since the branch and bound technique as well as optimization solver is used, it is hard to analyze the theoretical complexity of our proposed algorithms. Instead, we perform a benchmark experiment and report the execution time and memory usage of our proposed algorithm in Section 2.5.5.

We use an example to illustrate the spatial branch and bound tree. Figure 2.1 shows the diagrams of leaf nodes in the branch and bound tree of an extreme function. We color the leaf node: red, if the the leaf node is pruned by bounds; yellow, if the region is indivisible. The diagram on the left uses constant estimators and the one on the right uses affine estimators by solving LPs. After carefully examining these two diagrams, we can see that the red regions by affine estimators are "bigger" than the red regions by constant estimators. Equivalently, the diagram using constant estimators gives a "finer" refinement. The reason is that affine estimators by solving LPs always produce the best lower bounds, therefore the corresponding branch and bound tree always has the smallest size.

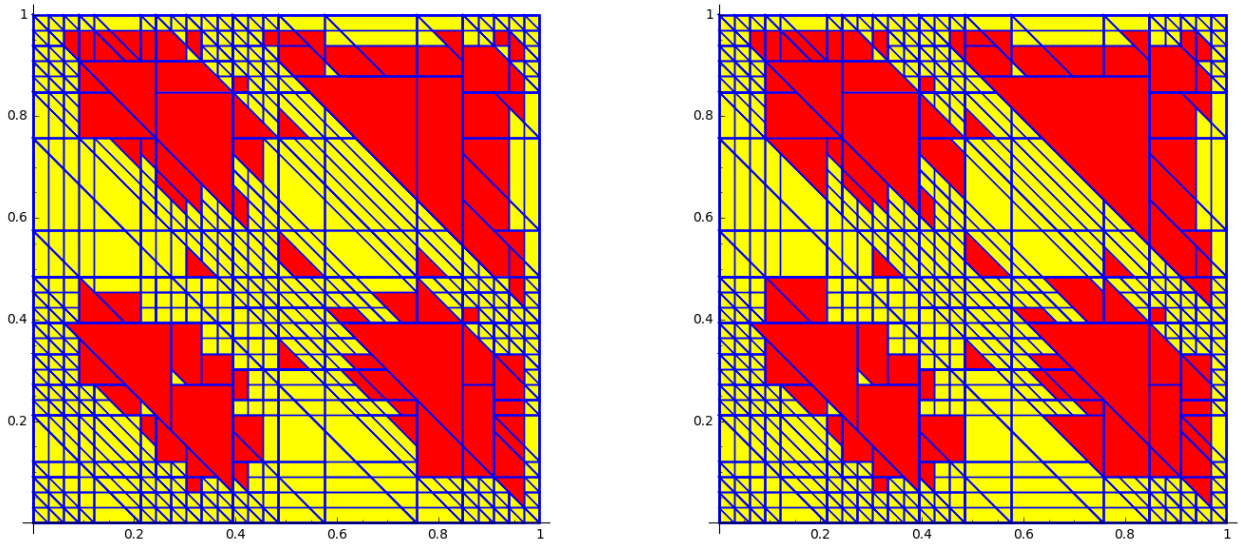

FIGURE 2.1. The diagrams correspond to a 7 slope extreme function `kzh_7_slope_1()` showing the leaf nodes in the branch and bound tree for computing the minimum of $\Delta\pi$. The yellow regions are pruned by indivisibility and the red regions are pruned by bounds. The diagram on the left uses constant estimators and the one on the right uses affine estimators by solving LPs. Both diagrams use best bound rule as the node selection strategy.

2.4.6. Other applications. Besides computing the minimum of $\Delta\pi$ and verifying whether π is subadditive, there are other important applications of the sBB algorithm.

The objective cutoff can be considered as a constraint which specifies the worst expected objective, and this feature has been implemented in most modern solvers. For a given objective cutoff, the solver will terminate early if it proves that there is no solution with better objective than the cutoff, without actually solving the problem to optimality. From the objective cutoff perspective, we can use sBB algorithms to check whether $\Delta\pi$ can reach the preset objective cutoff (for example 0 or $-\epsilon$). Solutions which produce the objective cutoff or better objectives can also be cached. If we set the objective cutoff to 0, then we can cache additive vertices and nonsubadditive vertices. To verify near-subadditivity, we can set the objective cutoff to $-\epsilon$. Then we are able to verify near-subadditivity more efficiently since we do not have to solve the $\Delta\pi$ minimization problem to optimality.

Figure 2.2 shows the diagrams of branch and bound trees for verifying (near) subadditivity. As we compare the two diagrams using different objective cutoffs, we can see that the diagram of verifying near-subadditivity has "larger" red regions. The "larger" red regions are those ones on which near-subadditivity can be proven using estimators but subadditivity cannot. Thus those

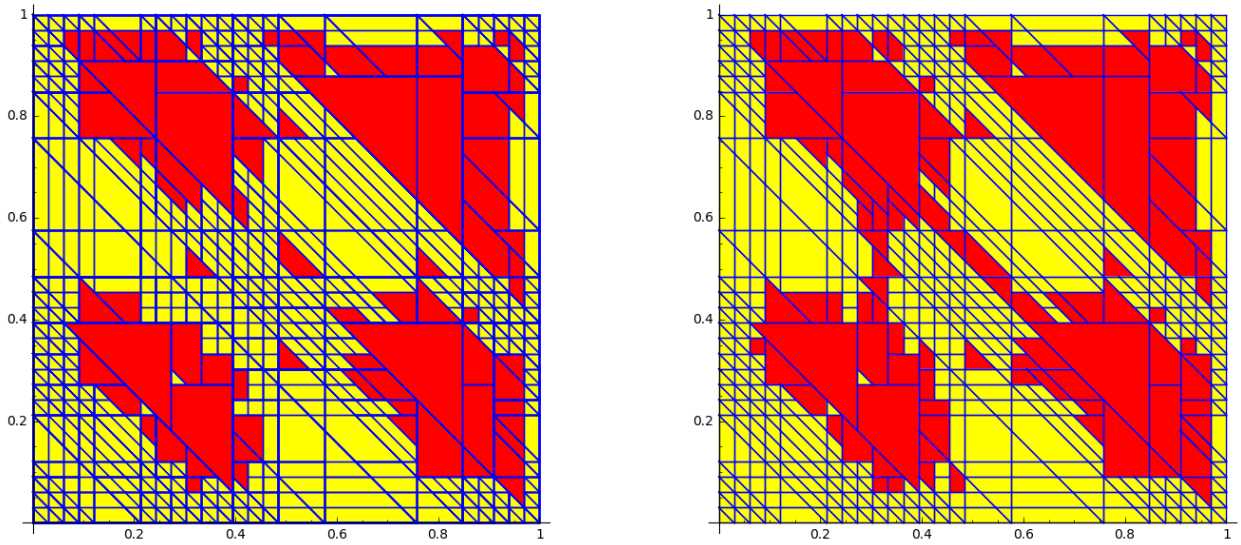

FIGURE 2.2. The diagrams correspond to a 7 slope extreme function `kzh_7_slope_1()` showing the leaf nodes in the branch and bound tree for verifying (near) subadditivity. The yellow regions are pruned by indivisibility and the red regions are pruned by bounds. The diagram on the left uses 0 as the objective cutoff (subadditivity) and the one on the right uses -0.01 as the objective cutoff (near-subadditivity). Both diagrams use constant estimators, and the best bound rule as the node selection strategy.

"larger" red regions can be pruned early in verifying near-subadditivity, which leads to a smaller tree size.

In the current extremality test for cut-generating functions, one key step is to compute the covered components from the additive faces. Our sBB algorithms can also be applied to find (inclusion) maximal additive faces.

2.5. Computational Results

In this section, we explain our benchmark experiments and the computational results. Our goal is to compare the performance of different algorithms applied to various subadditivity-related computational tasks. The benchmark experiment was conducted in SageMath on the Peloton cluster[1], where each node has 64GB RAM, 2 sockets, 16 cores, and 32 threads running Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz. We compare the time performance using selected algorithms applied to four computational tasks. The maximum solving time is 1 hour and maximum memory usage is 8GB. The outliers are removed by Tukey's rule. Given a set of measurements $\{x_1, x_2, ..., x_n\}$, let $Q1$ and $Q3$ be the first and third quantile. The interquantile, denoted as IQR, is defined as $Q3 - Q1$.

[1]https://wiki.cse.ucdavis.edu/support/systems/peloton

The values outside of the range $[Q1 - 1.5\,IQR,\ Q3 + 1.5\,IQR]$ are identified as outliers. The source code and instructions of how to reproduce the experiment are made open sourced[2]. This project is licensed under the terms of the MIT license.

2.5.1. Benchmark set. We first explain how we choose the benchmark set. There are in total 150 continuous piecewise linear functions (test instances). Fifty of them are extreme functions, including known extreme functions in the literature and their two slope approximations [**BHM16**]. Every minimal function can be arbitrarily approximated by a two slope minimal (thus extreme) function in the infinity norm. The approximation functions are generated by the "two slope fill-in" procedure and the two slope values are the limiting slopes at 0. The idea of the fill-in procedure is approximating each affine linear piece by many "zigzags", which are affine linear pieces with the fixed two slope values. The bound on approximation error can force the number of "zigzags" to be large, thus those approximation functions can have potentially many breakpoints. Based on the 50 extreme functions, we perturb them by using convex combination with "Gomory Mixed Integer Cut" to obtain 50 minimal but non-extreme functions. Similarly, we perturb 50 extreme functions using convex combination with simple non-subadditive functions to obtain 50 non-subadditive functions. By carefully choosing the coefficient in the convex combination, we make sure that the 50 non-subadditive functions all satisfy the property that $\Delta\pi \geq -0.01$.

We include Figure 2.3 as a histogram of the distribution of the number of breakpoints of all extreme functions in the benchmark set. Since we do convex perturbations to generate non-extreme and non-subadditive functions, the histograms of other functions look similar. By our intuition, the time complexity (in all algorithms) heavily depends on the number of breakpoints. Roughly speaking, the histogram represents the "difficulty distribution" of the benchmark set. As we see in the histogram, the most "difficult" test instance has around 10000 breakpoints.

2.5.2. Computational tasks. There are four computational tasks. Note that in different tasks, we use different subsets of the benchmark set and the algorithm set. We explain the four computational tasks as follows.

(1) Computation of the minimum of $\Delta\pi$ includes the whole benchmark set. There are three main algorithms which can be applied to solve the minimization problem: naive algorithm, MIP algorithm and sBB algorithm.

[2]Github repository: https://github.com/mkoeppe/jiawei-computations

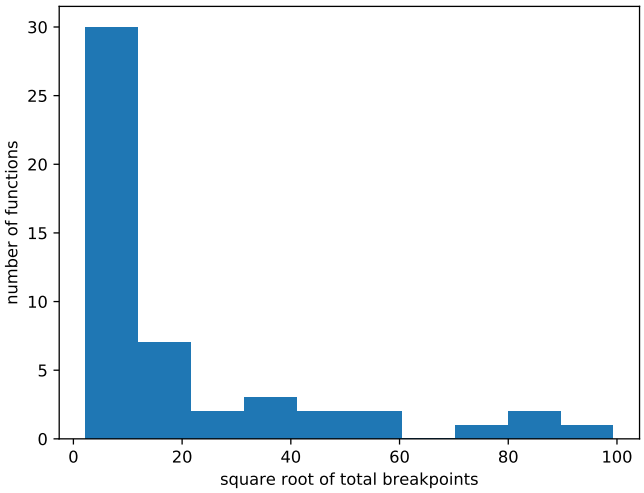

FIGURE 2.3. The histogram of the distribution of the number of breakpoints of all 50 extreme functions. The x-axis is the square root of the number of breakpoints. The y-axis is the number of extreme functions in each bin.

(2) Generation of additive faces is not applied to non-subadditive functions. The additive faces are usually used in the extremality test, so it is not of interest to generate additive faces of non-subadditive functions. We only use the naive algorithm and the sBB algorithm in this task since the MIP formulation cannot be used here.

(3) Verification of objective cutoff 0 is not applied to non-subadditive functions. The task here is to verify there is no better objective than 0 in the minimization problem (i.e. subadditivity). We use all three algorithms. In the MIP formulation, we use 0 as the "cutoff" parameter.

(4) Verification of objective cutoff -0.01 includes the whole benchmark set. The task here is to verify there is no better objective than -0.01 in the minimization problem (i.e. near-subadditivity). We carefully choose perturbation functions and coefficients in the convex combination so that every function in the benchmark set is nearly subadditive (i.e. $\Delta\pi \geq -0.01$). We use all three algorithms. In the MIP formulation, we use -0.01 as the "cutoff" parameter.

2.5.3. Algorithms. Next, we explain the algorithm set. As being first implemented in our software, the naive algorithm is set to be the baseline algorithm. In regards to the MIP formulation, we use the Dlog formulation explained in subsection 2.3.3, and we use Cplex as the MIP solver to solve the mixed-integer program. The default setting is used in Cplex, and the solver parameter "Upper Cut" is set to be the objective cutoff in two computational tasks. In regards to the sBB

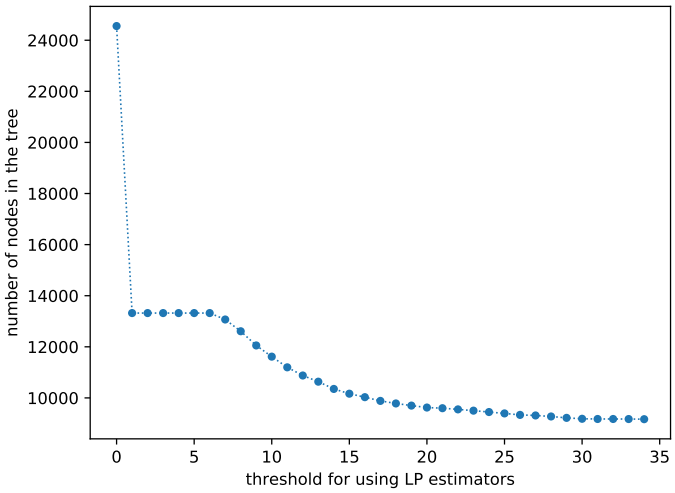

FIGURE 2.4. The graph shows the number of total nodes in the branch and bound tree using different types of estimators. The function is a 10 slope extreme function `kzh_10_slope_1()` and the task is computing the minimum of $\Delta\pi$. Best Bound is used as the node selection strategy.

algorithms, there are 12 variations. We use three traversing strategies: DFS, BFS and Best Bound (BB), and we use four different ways to construct the local lower bound of $\Delta\pi$: constant bound, fast bound, mixed bound and LP bound. The constant bound is obtained from constant estimators. Fast bound is the better one between the constant bound and the bound from affine estimators with fixed slope values. The LP bound is obtained by solving an LP in the corresponding sBB tree node. In terms of the mixed bound, we use 30 as the threshold. Specifically, if the LP size (the number of constraints) is larger than the threshold, then the fast bound is used. Otherwise, the LP bound is used. Therefore, the algorithm set consists of in total 14 algorithms.

We use an example to show how different estimator constructions can impact the size of the branch and bound tree. In Figure 2.4, the leftmost point corresponds to constant bound, and the second leftmost point (the big drop) corresponds to fast bound, and other points correspond to different thresholds in the mixed bound. We can consider the rightmost point as using the LP bound. The convex relaxation becomes stronger from left to right in Figure 2.4. Observe that using a stronger convex relaxation results in smaller tree size.

Note that the function $\Delta\pi$ is symmetric about $y = x$. The naive algorithm and sBB algorithms can utilize the symmetry to speedup the computation. We use the speedup technique in our experiment to save computational resources.

2.5.4. Results. We report the time performance in two different ways: shifted geometric mean and performance profile.

In the MIP community, the shifted geometric mean is mostly used for measuring the solving time of an optimization problem, such as [**LT13**, **AW13**]. Given a set of measurements of solving times $\{x_1, x_2, ..., x_n\}$, the metric $\sqrt[n]{\prod_{i=1}^{n}(x_i + s)} - s$ is called the shifted geometric mean of $\{x_1, x_2, ..., x_n\}$ with shift s. The shifted geometric mean is known to be robust to the interference of very large outliers (compared to arithmetic mean) and very small outliers (compared to geometric mean) [**Ach07**]. Besides solving time, the shifted geometric mean can also be applied to other optimization metric including the number of nodes explored in the branch and bound tree and the number of iterations of the simplex algorithm. The choice of the shift s is not standard, in terms of the solving time, the shifts of 1s [**GEG+17**], 10s [**KB16**], 60s [**Ach07**] have been used. The choice of the shift depends on what types of instances researchers focus on. Generally speaking, small shifts favor easy instances and large shifts favor hard instances.

We include the shifted geometric means with shifts 1s, 10s, 60s in Table 2.1. For every column, we color the winning algorithm. Note that the comparison results are not consistent using different shifts, so using shifted geometric means can be biased.

We notice that different traversing strategies have almost the same performance, and constructions of convex relaxations are the most important factor. For example in the minimization task, if the function is subadditive, then the best solution (primal bound) can be found very fast. The traversing strategy does not matter since no better solution can be found anyway. Among all sBB algorithms, the ones with fast bounds have the best performance. Generally speaking, in terms of performance, fast $>$ constant $>$ mixed $>$ LP. It seems that using fast bounds has a balanced tradeoff between the strength of convex relaxations and the effort to construct them. From the example in Figure 2.4 we can see that using fast bounds significantly reduces the number of nodes from using constant bounds, while aggressively solving LPs does not yield a much smaller tree size.

Compared to the naive algorithm, the best sBB algorithm has better performance in computing the minimum of $\Delta\pi$ and verifying (near-)subadditivity. As expected, the MIP solver has the longest running time. Although we are using different subsets of the benchmark set in two objective cutoff computational tasks, we can still see a trend that verifying near-subadditivity is easier than verifying subadditivity. We can even see a big speedup for the MIP solver with cutoff -0.01 compared to cutoff 0. In the additive faces generation task, the naive algorithm is the winner.

TABLE 2.1. Shifted geometric means (seconds) with shift 1s, 10s, 60s

algorithm	$\Delta\pi$ minimum			cutoff 0			cutoff -0.01			additive faces		
	1s	10s	60s	1s	10s	60s	1s	10s	60s	1s	10s	60s
naive	9.96	34.88	92.69	9.92	34.81	92.73	10.01	35.02	93.13	6.56	21.32	54.38
MIP	1404.59	1406.23	1414.81	1559.99	1561.68	1570.54	722.36	726.10	736.69			
BFS_constant	4.36	13.12	33.52	5.56	17.62	46.41	1.52	2.83	3.69	7.00	23.22	65.70
BFS_fast	3.19	9.11	22.19	4.52	13.41	33.00	0.81	1.20	1.35	7.39	24.05	67.24
BFS_mixed	12.01	26.62	60.65	17.69	39.98	89.55	2.34	3.60	4.81	20.80	46.40	104.89
BFS_lp	13.47	29.78	67.05	19.03	43.66	99.31	4.06	7.01	10.22	20.82	47.19	106.72
DFS_constant	4.84	14.81	37.94	5.46	17.30	45.71	1.51	2.78	3.60	6.92	23.07	65.46
DFS_fast	3.84	10.85	25.32	4.42	13.15	32.48	0.81	1.19	1.32	7.24	23.66	66.47
DFS_mixed	16.08	35.27	77.63	17.50	39.73	89.31	2.29	3.55	4.77	20.84	46.63	105.32
DFS_lp	17.77	38.92	86.12	19.53	44.34	100.30	4.18	7.19	10.43	21.68	47.98	107.69
BB_constant	4.76	14.86	40.26	6.58	21.31	58.65	1.63	3.06	4.04	7.33	23.99	67.27
BB_fast	3.16	9.42	24.36	4.89	14.78	37.69	0.83	1.24	1.39	7.59	24.40	67.72
BB_mixed	10.46	23.79	55.70	17.71	40.09	90.15	2.29	3.55	4.75	20.53	45.87	103.76
BB_lp	12.41	27.73	63.48	19.28	44.05	100.12	4.17	7.21	10.58	21.65	48.16	108.13

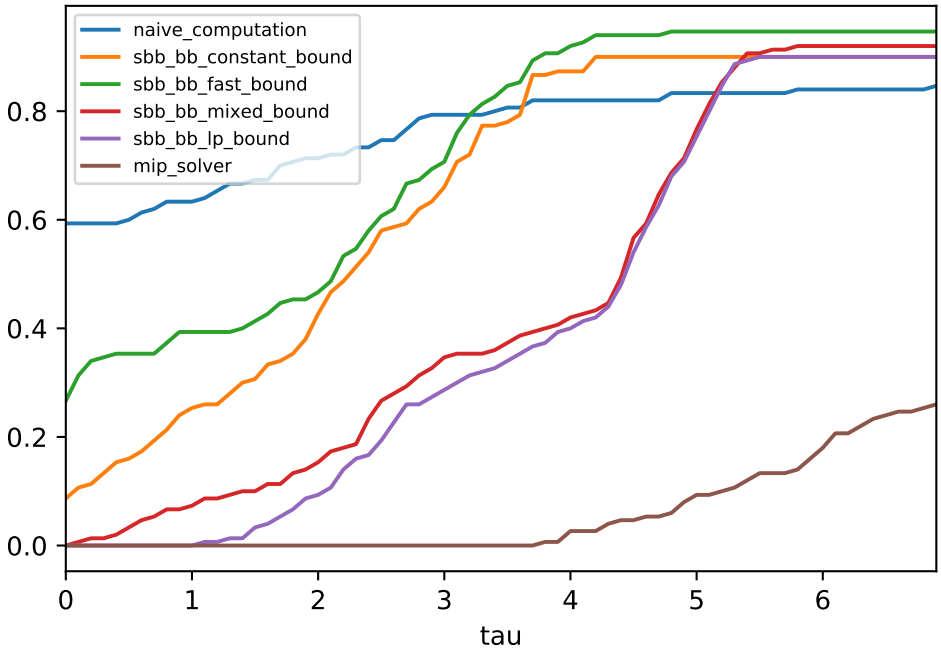

FIGURE 2.5. Performance profiles for computation of $\Delta\pi$ minimum (log scale).

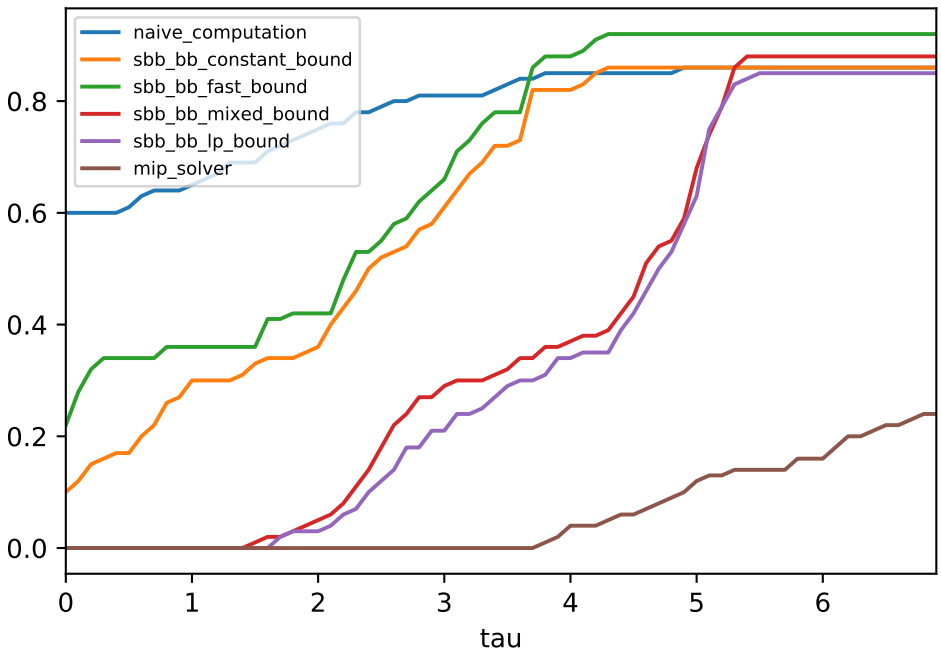

FIGURE 2.6. Performance profiles for verifying subadditivity (log scale).

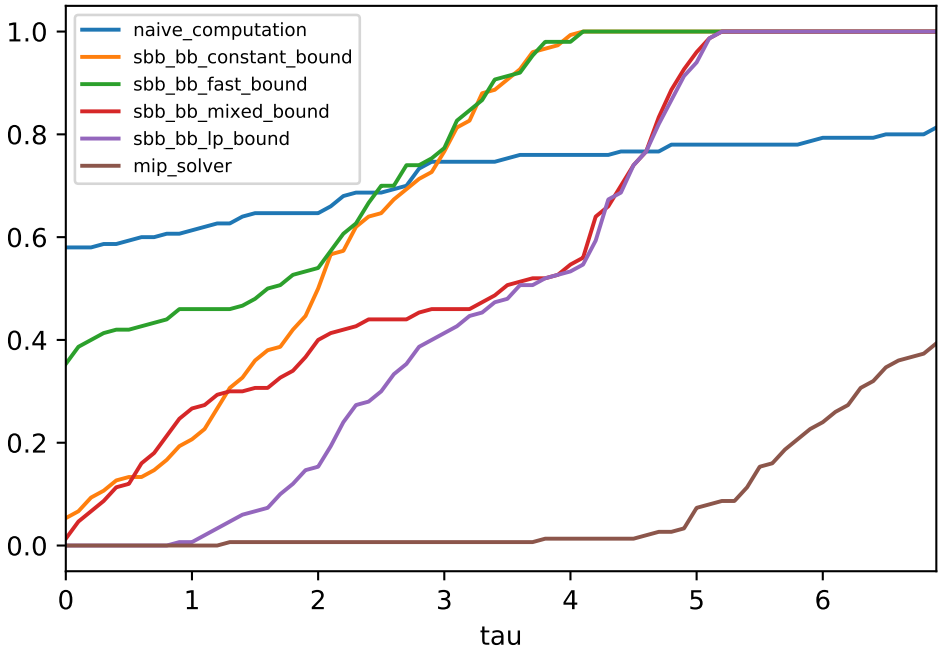

FIGURE 2.7. Performance profiles for verifying near-subadditivity (log scale).

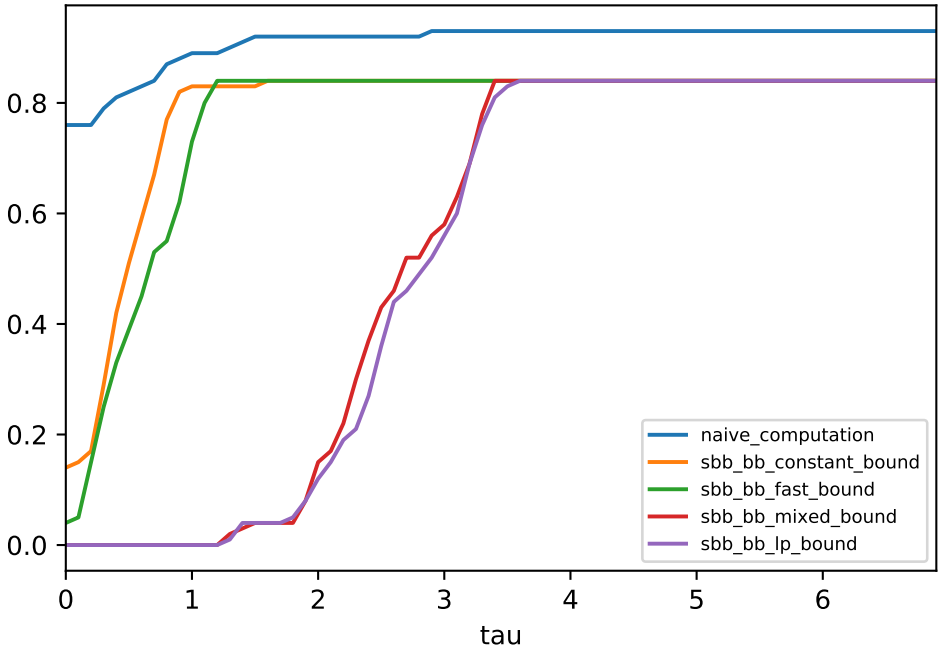

FIGURE 2.8. Performance profiles for computation of maximal additive faces (log scale).

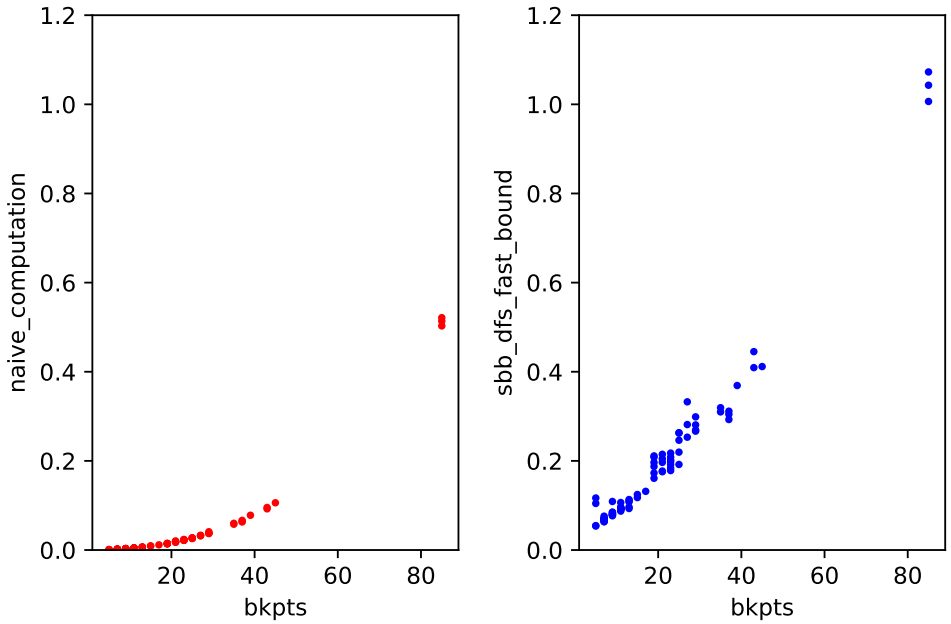

FIGURE 2.9. Running time (seconds) of test cases with fewer than 100 breakpoints in computing minimum of $\Delta\pi$.

We include performance profiles [**DM02**] for computational tasks in Figures 2.5 to 2.8. Due to the large number of sBB variants, we only plot sBB algorithms with Best Bound as the node selection strategy. Overall we see consistent results as in shifted geometric means. In computing the minimum of $\Delta\pi$ and verifying (near-)subadditivity tasks, we observe that the naive algorithm has most wins but overall the sBB algorithm with fast bounds is the best. In the additive faces generation task, the naive algorithm has the most wins and is also the overall best algorithm. However, the speedup of verifying near-subadditivity from verifying subadditivity cannot be observed by using performance profiles.

2.5.5. Complexity. We briefly discuss about the time and space complexity. We only pick one sBB algorithm: the one using fast bound and DFS as the node selection strategy and we compare it with the naive algorithm. By our intuition, the time complexity heavily depends on the number of breakpoints, so we show how time and space complexity vary based on the number of breakpoints.

Figure 2.9 and Figure 2.10 shows the running time of computing minimum of $\Delta\pi$ for test instances with number of breakpoints less than 100 and less than 400 respectively. As we expect, the running time of the naive algorithm has a clear quadratic increase as the number of breakpoints,

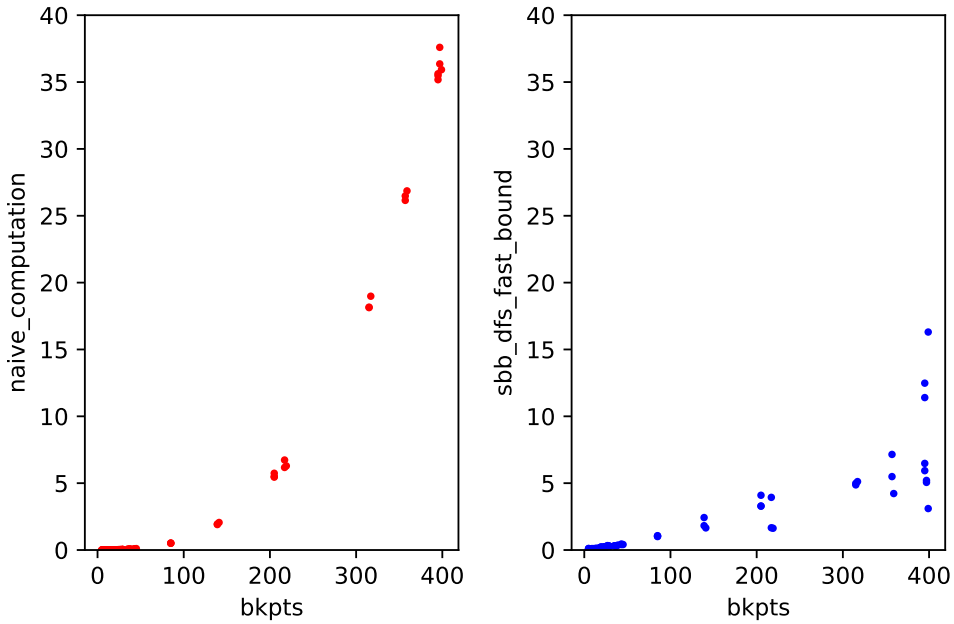

FIGURE 2.10. Running time (seconds) of test cases with fewer than 400 breakpoints in computing minimum of $\Delta\pi$.

and it also works better than the sBB algorithm if the number of breakpoints is small. As the test instance becomes more "complicated", the sBB algorithm is more time efficient although it is unclear to see the relation between running time and the number of breakpoints.

Figure 2.11 shows the memory usage of computing minimum of $\Delta\pi$ for test instances with number of breakpoints less than 400. Observe that the naive algorithm has nearly constant space complexity and the memory usage doesn't increase as the number of breakpoints increases. On the other hand, we see an increasing trend of memory usage in the sBB algorithm as the number of breakpoints increases. The reason is that we cache the entire branch and bound tree in our implementation, and more "complicated" test instance generally has a larger sized tree. The reason we cache the entire tree is for visualization and comparison with the two dimensional complex $\Delta\mathcal{P}$, and we also cache the estimators in every node for verification purposes. Depending on the specific purpose, it is possible to delete finalized nodes to release memory. For example, the DFS algorithm can be implemented in space $O(d)$, where d is the height of the tree [**Kor85**].

One difference between computing minimum of $\Delta\pi$ and generating additive faces is a lower bound on the space complexity. If explicit caching of additive faces is required, then we need at least the amount of memory to store all additive faces. However for the minimization problem, it

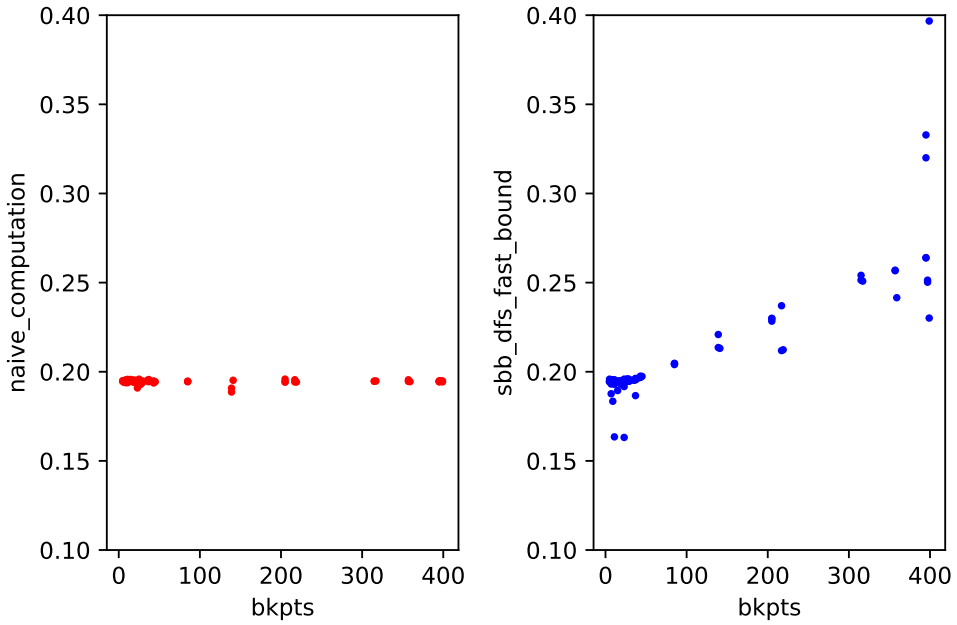

FIGURE 2.11. Memory usage (GB) of test cases with fewer than 400 breakpoints in computing minimum of $\Delta\pi$.

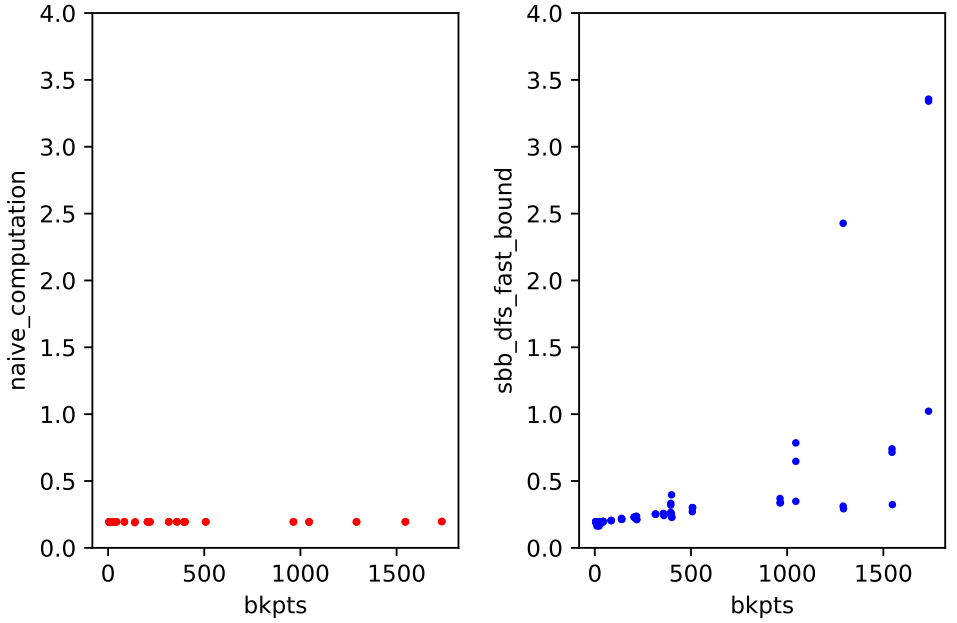

FIGURE 2.12. Memory usage (GB) of test cases with fewer than 2000 breakpoints in generating additive faces.

only requires constant space. In our implementation of generating additive faces, both the naive and the sBB algorithms store all additive faces in a list. Figure 2.12 shows the memory usage of generating additive faces for test instances with number of breakpoints less than 2000. Observe that naive computation no longer has constant memory usage. Similar to the minimization problem, the sBB algorithm requires much more memory than the naive algorithm. To beat the lower bound and save memory usage, it is also possible to use the "streaming" technique to (implicitly) generate additive faces on the fly (for example, using `yield` in Python).

2.6. Future Work

One limitation of the experiment is the benchmark set. The choice of the benchmark set is intuitive. Since our goal is to verify subadditivity for functions with complex structure, we "deliberately" complicated functions by using two slope approximations which will make the number of breakpoints large. One possible future work is to find more test cases which can "best represent today's cut-generating functions".

From the experiment results we see that sBB algorithms are promising at least for verifying near-subadditivity. We tested variants with different node selection strategies and convex relaxation constructions. From the algorithm perspective, there are two directions worth exploring. One is how to choose and divide the interval in the branching step. In our implementation, we just choose the largest interval and mid-indexed breakpoint. The other one is how to choose the slope values while constructing fast affine estimators with fixed slope values.

We also see applicability of sBB algorithms in generating additive faces, which can be used in further extremality test. Although the current sBB implementation cannot beat the naive algorithm, it is still promising for the extremality test in the future work. Unlike the naive algorithm which needs to generate all additive faces, an sBB algorithm can be implemented so that it only generate those "necessary" additive faces used in the extremality test. Adding certain pruning criteria can help early pruning, therefore speedup the computation.

In terms of the space complexity, depending on the specific usage, the sBB algorithm code can be further optimized. Deleting finalized nodes and unnecessary caching, choosing memory efficient node selection strategy, and using a streaming technique such as generators in Python can help save memory usage and reduce the number of out-of-memory instances.

Another (not near) future work is applying sBB algorithms in the multi-row cut-generating functions. One challenge is that the domain of a multi-dimensional affine linear function is not as trivial as an interval. Thus dividing and projecting regions in higher dimension are more complicated than the one-dimensional case.

CHAPTER 3

Benchmarking on High Performance Clusters

Researchers are interested in measuring the execution time of certain programs in a specific platform. Gathering such information can help to optimize the program as well as the platform. After the optimization, researchers need to perform the same benchmarks to detect speedups. A general question to ask is how to complete such task "rigorously"? A good scientific researcher needs to setup a well designed experiment, analyze experiment results in a statistically rigorous way, and try to convince readers by providing reproducible data.

There are several works on the evaluation of the infrastructure for the High Performance Clusters (HPC) [**SBZ$^+$08**,**FMMS14**]. In [**SBZ$^+$08**], the difference between measuring real applications and kernel benchmarks is discussed. Since a large-scale scientific computing task usually occupies more resources and requires longer time to finish, it is essential to analyze performance of real applications using HPC technology. In [**FMMS14**], the authors propose a methodology for evaluating how important elements can influence the execution times, including class of real applications, the problem size, the programming language, the breakdown of CPU, memory and I/O time.

In general, most researchers are interested in two types of time measurements of a segment of programs: wall clock time and CPU time. The wall clock time measures how long the user has to wait in order to get the desired results. However, this measurement inevitably includes "time-sharing" overhead including I/O, memory sharing with other programs. To measure the total time the processor actually spends on the specific program, the CPU time is used. On the other hand, the CPU time doesn't include "time-sharing" overhead which sometimes is also of our interest, especially the experiment is on a high time/memory-shared platform like HPC. Reporting both wall clock time and CPU time can easily show how significant the time-sharing interference is. Depending on how researchers allocate resources to perform benchmarks, the CPU time can be much longer than the wall clock time if certain multiprocessing techniques are used.

The benchmark set contains all test instances, on which different algorithms are tested. To make the comparison fair, the benchmark set should try to best represent today's problems. Here today's

problems refer to real problems which researchers are interested in. Problems are not necessarily challenging, but they rather serve as a "performance ruler" which as a whole can be used to measure algorithms. We refer interested readers to several standard benchmark sets in various research fields, including MIPLIB [**mip18**] in the optimization community, MNIST [**Den12**] in computer vision, and PMLB [**OLCO$^+$17**] in data mining. These benchmark sets are well maintained and are continuously being extended to a broader context. For example, the EMNIST [**CATVS17**] is a generalization from MNIST to represent handwritten letters instead of digits, and MIPLIB library has evolved to the sixth generation. It is important that every test instance in the benchmark set has a unique key/index to refer to. Having a unique key is convenient for the user to retrieve test instance information as well as focus on special instances such as outliers or misclassifications. From the perspective of reproducibility, the unique key of each instance serves as a "universal name" which can be used in the interactive data analysis, and it is easy for researchers to identify discrepancies while doing a reproducibility study.

Although there is no formal definition of reproducibility of an experiment, various principles have been recommended [**Fei06**, **PVB$^+$19**, **HB15**]. In terms of reporting computational results, Jupyter Notebook becomes more favorable in producing reproducible data analysis. In [**KRKP$^+$16**], the authors state that the Jupyter Notebook is a "publishing format" for computational results in today's research work . The notebook interface is supported by many mathematical software systems, including Mathematica, Maple and SageMath. It is also convenient to explain the workflow of a computational work by using comments, figures and equations. The Jupyter Notebook can be run both locally or in a remote server, which makes it flexible to run data analysis interactively. Sometimes the same experiment needs to be done again for reproducibility purposes, or for testing performance with changed parameters and an implementation optimized. It is crucial that the computational results of different configurations are cleanly maintained. A version control system such as Git has been proved to facilitate both reproducibility and transparency of an experiment [**Ram13**, **Gan13**].

In this chapter, we review literature about reproducibility, statistics and resource sharing in benchmark studies. We try to apply the important principles to conduct a reproducible experiment to benchmark a real application, using a Python-based software in a HPC. Our experiment is run in a HPC because the overall run time would be too long if it is run in a personal computer. However, after gathering computational results, the data analysis can be done locally using Jupyter Notebook.

We explain our experiment which uses git submodule technique to store computational results. For computational results of each configuration, there is a git commit/branch corresponding to them. While performing data analysis, one can easily checkout the desired branch/commit without being worried about "data overwrite issue". The benchmark work is the computational work in Chapter 2, which compares running time and memory usage of different algorithms. So in this chapter, we omit mathematical parts and focus on reproducibility.

3.1. Reproducibility

In an article published in Nature [**Bak16**], it is pointed out that a large portion of published experiment failed to be reproduced by other researchers even by the authors themselves. The paper [**Bak16**] contains a survey, in which the percentage of researchers who tried but failed to reproduce other researchers' work is reported. The survey covers several natural scientific fields, including chemistry, biology, medicine, physics and engineering. A high percentage of unsuccessful reproduction raises the crisis of confidence. Compared to natural science fields, computer science has less randomness and the open source code makes it easier to reproduce experiment in computer science research. The database community is the pioneer which addresses reproducibility, and repositories containing results and software are portable [**FBS12**]. In cases where inconsistent results are obtained from a reproducibility study, unlike natural science field, the "bugs" in the computer science community make it easier to track down the root cause [**Pen11**]. Nevertheless, lack of reproducibility is still an issue in computer science community, for example the paper [**CP16**] shows that failure of reproducing the own work is not uncommon. The authors in [**CP16**] encountered technical obstacles that mathematical formula is not clearly stated while trying to implement one paper's algorithm. In the communication with the original authors, they indicated the proposed algorithm was still under development and refused to give out source code and data.

The reproducibility is more about qualitative analysis rather than quantitative reproduction, since it is practically impossible to reproduce the exact measurement results [**Pen11**, **Fei06**]. Due to randomness, reproduction of the exactly same execution time measurement is unlikely to happen even when the experiment is done in the same platform by the same researchers. Therefore, repetitions of the same experiment and statistical analysis are needed. Even if the variance among measurements in repetitions is relatively small, experimenters are expected to explain the full procedure [**PVB+19**]. The main objective of reproducibility is to gain insights and make progress based

on other researchers' work [**Fei06**, **PVB+19**, **Pen11**]. One benefit of conducting reproducibility study is that, by running the same experiment in a slightly different way (for example, in a different platform or different configuration parameters), it will bring the original experiment into a broader and more general theory [**Fei06**]. Even if some discrepancy is discovered, it is also a valuable context which leads to further necessary study. The authors in [**Fei06**] state that standardized experimental design is preferred, specifically collective instructions regarding how to conduct an experiment are expected to be provided. In their paper, the standardized experiment design is described to be analogous to a lab manual, which is widely used in natural science field (for example [**GS12**]). Researchers can then use the experiment design and adjust it based on their own research goal.

There is some work trying to reproduce other researchers' experiment, but the criteria for determining reproducibility are quite intuitive. For example, in [**HCA15**] the authors test the reproducibility of their own experiment and some experiment in other papers by comparing the variance. For every repetition of each experiment, the mean value ($\bar{t_i}$) is recorded. Using all mean values in all repetitions of the same experiment, the reproducibility is evaluated by observing the plots of the variance of the normalized execution times ($= \bar{t_i}/\min(\bar{t_i})$).

The claim that an experiment is reproducible relies heavily on the control of experimental environments and the rigorous statistical analysis of the experiment results. Non-proprietary publishing work is obviously more convenient for other researchers to perform reproducibility studies. In the following part of this section, we highlight three important aspects in reproducibility: experiment design, statistical analysis and non-proprietary work.

3.1.1. Experiment design. Good experimenters are expected to make the experiment well documented. A well explained documentation of an experiment is very important for other researchers to repeat the experiment and perform reproducibility study. The documentation should include any prerequisites, configuration parameters and external dependencies.

The authors in paper [**CBLA19**] empirically studied other researchers' bad practices in a microbenchmarking framework, namely Java's Microbenchmark Harness (JMH). Although these bad practices are Java-specific, we can still learn how to avoid similar mistakes in other programming languages, so we summarize several recommendations in the paper [**CBLA19**] as follows. Most modern programming languages have shortcuts to speed up the computation, for example skipping

useless code segments, unfolding loops and caching computed values. Researchers need to be careful and make sure the codes they write are actually doing what they intend to do step by step. One systematic error/overhead could be introduced by calling "`get_current_time`" or similar function before and after the execution code segment. The overhead can be an issue if the measured program has very short running time. One mistake about computing this error/overhead is calling the function "`get_current_time`" twice consecutively. If such function is called twice consecutively, some software will recognize the "useless" code block and speedup the second call, which will make the computed overhead smaller than the actual overhead. Sometimes a benchmarking paper is trying to convince readers that a new algorithm/platform is better than the old one. However, if the old algorithm/platform is not optimized, then it is not fair to say the old algorithm is worse. Another important assumption to make is that both algorithms should have the same functionality. To make the comparison fair, researchers should make sure that for the same input, all measured algorithms produce the same or at least equivalent output.

In the experimental design, it is also suggested to introduce randomization to the experiment. In [**HCA15**], the authors shuffled the order of the experiment "to respect the principles of experimental design (randomization, replication, blocking) [**Mon17**]". In the MIP community, the constraint matrix is usually permuted to introduce randomness.

3.1.2. Statistically rigorous report. Due to the variation in measured execution times, repetitions are necessary in order to generate statistically valid results. It is also crucial to minimize the cost of total experiment time while maintaining the rigorous and reproducible results.

Two level repetition experiment, which contains an execution/run level and an iteration level, has been used in literature [**KJ13**, **HB15**, **BHT17**]. One run/execution consists of one or more iterations, and each iteration gives a number/measurement. There is no standard guideline on how to choose the number of repetitions in the two levels and how to analyze gathered measurements [**BHT17**,**GBE07**]. Although there is no standard guideline on choosing the number of repetitions, providing and possibly reasoning about the detailed information on the number of repetitions are useful for other researchers to evaluate reproducibility of its work. In [**KJ13**], the authors introduced a general guideline to determine the number of repetitions needed in each level for the fixed platform and benchmark in their experiment. The recommended numbers of repetitions play an important role in terms of reproducibility. The idea in [**KJ13**] is to run an initial experiment

and use the collected data to determine the number of repetitions needed based on the variation in each level. The numbers of repetitions for the given benchmark and platform serve as the cookbook for other researchers to reproduce results and save excessive experiment time. The paper also gives recommended iterations for the hardware to reach the "stable state" in their experiment. The stable state in their experiment generally means that the gathered measurements have relatively small variance. However, waiting for the hardware to reach the "stable state" can be unrealistic, if it is already expensive to gather a single measurement, or simply the experiment times out. The measured execution times in this paper [**KJ13**] are very small, so it is not time/resource consuming to run an initial experiment and wait for the stable state. If gathering a single measurement is very expensive, the authors in [**HB15**] suggest to use only one iteration in each execution/run. In one execution, sequential analysis can be used to dynamically decide the number of iterations needed. The idea of sequential analysis is minimizing the experiment cost while keeping results statistically reliable. We will elaborate on the sequential analysis in Section 3.5.

The gathered execution times of the same instance have variance due to random errors. In practice, it is impossible to know the exact distribution of the execution time. So researchers are usually interested in the estimated value or confidence interval of the mean execution time. Two important aspects in benchmarking papers are measuring execution times and detecting differences in performance. These two aspects correspond to confidence interval and hypothesis testing in statistics. Various methods of computing confidence interval and hypothesis testing have been used in the benchmarking literature.

In terms of the computation of confidence intervals, researchers usually rely on the central limit theorem and assume that the sample mean is asymptotically normal if the sample size is large enough. The standard statistics textbook usually indicates that 30 is large sample size so that the sample mean follows normal distribution approximately. The paper [**GBE07**] explained how to compute the confidence interval of the execution times with different samples size and how to use Analysis of Variance (ANOVA) to compare more than two alternatives in terms of performance.

Here we introduce an evaluation in [**BHT17**] showing that using different analysis strategies may lead to different or even strange conclusions. Suppose for a fixed instance, there are n repetitions in the execution level and m repetitions in the iteration level. Specifically, we begin gathering execution times by running the instance m times, and we repeat the process n times. Then we get n groups of measurements $\{x_1^1, x_2^1, ..., x_m^1\}$, $\{x_1^2, x_2^2, ..., x_m^2\}$, ..., $\{x_1^n, x_2^n, ..., x_m^n\}$, where the superscript

represents repetitions in execution level and subscript represents repetitions in iteration level. The authors in [**BHT17**] used three commonly used strategies for picking samples to perform statistical analysis. One strategy considers all measurements $\{x_j^i : \forall i, j\}$ as a whole (called "merged"), one strategy takes the mean of every execution $\{AVG_j(x_j^i) : \forall i\}$ (called "means"), and another strategy takes one measurement (max/min/first/random) in every execution, for example $\{\min_j(x_j^i) : \forall i\}$ (called "one sample"). The authors evaluated confidence interval computations in four benchmarks in terms of the three analysis strategies. The evaluation is based on the probability of the computed confidence interval containing the true mean value. The true mean value $\bar{x}$ is defined to be the average of all measurements, i.e. $\bar{x} = \sum_{i,j} x_{i,j}/(mn)$. The probability computation is one Monte Carlo simulation which follows a bootstrap idea. Each time one confidence interval is computed using by the `stats.t.interval` method of the SciPy package, based on one random sampling with replacement. The probability is computed by counting how many times the computed confidence interval contains $\bar{x}$. It is discovered that the confidence interval from the merged method has high probability to miss the true mean in three benchmarks out of four, and the one sample and means methods may still give bad confidence interval which is likely to miss the true mean in one benchmark.

One common mistake in statistical analysis is applying the central limit theorem without checking the normality assumption. One example is illustrated in paper [**HCA15**]. In their experiment, the authors first gathered a large number (10000) of run-times of a program, and observe that the distribution of the 10000 measurements has two distinct peaks. Then the authors wanted to check how large the sample size can make the sample mean to have a normally distributed behavior. After random sampling with different sample size and plotting the distribution of the sample, the authors observed that the distribution of sample means with sample size 30 still has two distinct peaks, and at least 500 sample size is needed for a normal distribution. Most statistics textbooks assume that size 30 is large enough for the sample mean to obtain the normal distribution, but the experiment in [**HCA15**] shows that researchers need to check normality assumption before applying the central limit theorem.

The authors in [**VK12**] claimed that in the computer science field the execution time measurements do not follow the normal distribution. Their reason is that the random error can add much more on the execution time than reduce, therefore the actual distribution observed is usually skewed to the right. Another common assumption in the statistical study, like in the central limit

theorem, is the independent sampling. This assumption is usually not true in measuring execution times. Experimenters usually run the same instance multiple times consecutively. Due to complex programming language structure like cache, memory and garbage collector, it is difficult to reason that the previous run is independent from the following run. Non-parametric testing is considered robust for unknown type of distributions. Bootstrap [**DE96**] is an important non-parametric method widely used without the normality assumption.

There are several works comparing parametric and non-parametric methods in statistical analysis. The paper [**BHT17**] evaluates the Welch's t-test, Mann-Whitney U-test, as well as confidence interval overlapping. The first one requires the normality assumption and is used to determine whether two normal distributions have the same mean. The second one (also called Wilcoxon test) is a non-parametric hypothesis testing that is used to determine whether both distributions are the same. The authors used a Monte Carlo simulation to show that applying parametric and non-parametric methods can lead to different results and both can produce bad results. The authors in [**RG04**] used a Monte Carlo simulation to test various parametric methods and non-parametric methods regarding hypothesis tests of equal mean or equal variance. They observed that parametric statistical analysis is in general robust and only in few cases, the non-parametric Wilcoxon test has significantly better performance than other parametric methods. They claimed that the sample mean still follows the asymptotic normal distribution, so the standard statistically analysis can still provide important justification though the normality assumption and independent sampling may not hold. From the previously mentioned two papers, we can see that statistical methods play an important role in the data analysis quality, but the quality is also dependent on the actual data. For some data multiple statistical methods have robust results, whereas certain method needs to be used for some other data.

To measure the speedup of a new system/algorithm, researchers usually perform statistical hypothesis testing to detect the difference between new and baseline systems/algorithms. The ratio between execution times of two systems has also been used to verify speedup. Different aggregation functions have been used to estimate the ratio. The paper [**LJ04**] proposes a method to compute a confidence interval for the ratio by using (arithmetic) sample mean and sample variance while assuming approximately normal distribution. On the contrary, the authors in another paper [**Mas04**] claim that it is not fair to use arithmetic mean for the ratio, and the geometric mean should be used instead.

Besides mean and variance, the other two moments skew and kurtosis can also be computed to check whether the distribution is normal [**Mas04**]. If the skew is negative (or positive), then it suggests that the distribution has a long tail to the left (or the right). A positive kurtosis usually means that the samples cluster around the mean more compared to the normal distribution. A strongly positive kurtosis could be resulting from correlated samples.

3.1.3. Non-proprietary work. Proprietary source code and data is inconvenient for other researchers to repeat the experiment and produce reproducibility study, and the source code, including the input test cases, the scripts and the data, is encouraged to be made open-sourced [**VK12**,**RHGM18**]. The paper [**VK12**] also suggests that the computer science community should encourage researchers to publish reproducibility study of others' work, and the journal editors should accept such work no matter whether the reproducibility study supports the original paper or not. The authors in [**VK12**] believe that such encouragement and culture change can foster the reproducibility study and on the other hand lead researchers towards publishing reproducible papers.

In the artificial intelligence research field, researchers are studying various AI systems and trying to understand them better. A closed-source AI system simply serves as a "black-box" system and its algorithm or implementation doesn't provide any insight, if the licensor of the system keeps all works proprietary [**DSB17**]. The authors in [**DSB17**] also explain the proprietary AI has disadvantages on safety, trust, ethicality, and fairness. One challenge of the study of AI system is the fact that it is difficult for other researchers to access the system due to the use of proprietary data and the difference in computational power [**EPS+18**]. The use of proprietary data on the training step makes it challenging to apply the system to users' own prediction or classification problems [**Voo17**]. Fortunately, in the AI community, many published research papers use well-known databases (e.g. MNIST database) as their training data, which provides transparency at least in the training step.

In a broader field of machine learning models, the paper [**Rud19**] suggests that black-box machine learning models should not be used in high stakes decision making. The authors in [**Rud19**] summarizes several issues on using proprietary machine learning platform to solve practical problems, including criminal recidivism prediction, weather forecasting and medical diagnosis. The issues include racial bias in recidivism prediction and bad prediction/diagnosis results in certain

cases. The authors also pointed out that the proprietary machine learning platforms are business models which make profits are not responsible for the quality of individual predictions, and individual bad prediction/diagnosis could lead to dangerous results. On the other hand, the authors worried that companies would not develop such machine learning technology if the platform is forced to be transparent. In the paper [**SBO$^+$07**], it is encouraged to share the full code of the experiment in machine learning research. The authors believed that by keeping the code open-sourced, it is beneficial for reproducibility, fair comparison with other methodologies, and discovery of hidden tricks like a number of tuned functions that were not clearly documented in the original work. They also thought that sharing the code can prevent the unsatisfactory debate about reproducibility between the authors of the original work and other researchers trying to reproduce the work, which is not uncommon especially in the natural science community.

The situation is trickier in the data mining field, especially when the data contains private information. If such information is used, it is in general inappropriate to ask the authors to publish data for reproducibility work. Many research papers in data mining focused on fields with a large amount of confidential data and discussed how to maintain proprietary rights; see examples like [**DGB07**, **PBV08**]. The paper [**PT09**] summarizes several ethical issues in data mining and provided suggestions in customer, legal and business perspectives to foster customers' trust and maintain information quality while following legal private policy.

Nowadays, there is a trend of using online platforms (e.g. Github) to share open sourced codes for benchmarking experiments. Here we explain two examples: `github.com/h2oai/db-benchmark` and `github.com/JuliaCI/BenchmarkTools.jl`. The first repository does benchmark work for database-operations like `groupby` and `join`. The repository has a clear documentation on how to reproduce the benchmark and the benchmark is routinely rerun by its authors. The second repository provides the framework of running reproducible benchmarks in Julia, and it can measure execution times and memory used for certain Julia code blocks. Although these works are not too complex and there is no publishing literature based on their repositories, they do provide good examples on open sourced benchmarking codes. We recommend experimenters to utilize an online platform and provide open source codes of their benchmark work, and it would be best to provide the detailed documentation on how to reproduce the benchmark results in a file like `README.md`.

3.2. Computation on time/resource sharing hardware

There are several works on the evaluation of the infrastructure for a HPC [**SBZ+08**,**FMMS14**]. The breakdown of execution time into computation, communication and I/O is studied [**SBZ+08**]. One pitfall in that experiment is the lack of statistical analysis. Only a single number of execution time is reported for each experiment. In [**FMMS14**], the authors propose a methodology for evaluating how important elements can influence the execution times. Those important elements (called Essential Elements of Analysis in their paper) include the class of real applications, the problem size, the programming language, the breakdown of CPU, memory and I/O time. The authors in [**FMMS14**] compute the 95% confidence interval based on 30 gathered measurements for each instance. They reported that the confidence interval is very small, therefore only a single number is reported for every instance.

Before starting an experiment, it is important for the researchers to identify important factors which could influence the execution times. The paper [**Lil05**] discusses the source of errors in a experiment, and we summarize the authors' conclusions as follows. The errors in a time-sharing system among multiple users could result from interrupts to service network interfaces, time-of-day clocks, user interactions, cache misses, system exceptions, memory page faults and so on. These errors can be classified into two main categories, systematic errors and random errors. The systematic errors are usually considered to be "experimental mistakes" and tend to be invariant across all measurements. So experimenters should try their best to eliminate the systematic errors. On the other hand, random errors need not to be controlled by the experimenters since these errors are unbiased, although random errors will still impact the precision of measurements in the experiment.

In the paper [**MF17**], the authors propose a method trying to completely eliminate the systematic errors, given that the systematic errors are invariant across all measurements. For example, certain execution time measurement uses functions like "`get_current_time`" or "`initialize_parameters`", and these functions could potentially result in systematic errors especially for measuring small execution times. The proposed method in [**MF17**] uses a linear regression model, trying to fit a linear function $T = N \times T_e + \epsilon$, where N is a relatively small integer representing the number of repetitions. The actual execution time T_e is of interest, and the (invariant) systematic error ϵ should be eliminated. After finding the solution of the least square problem (with outliers

removed), the estimate value of T_e and its confidence interval are obtained. The authors claim that, compared with the conventional method which takes the mean of a large number of measurements, the proposed method have better accuracy and lower variance given the same number of total measurements.

Ideally, measuring execution times of a program should be done in a way that assures that the program is the only one running on the platform. However, if the computation is done in a time/resource sharing hardware such as a HPC, it is impossible to control the resource competition caused by other programs of other users. We can still use the computational results to see how the competitions of resources, like CPU or memory, can influence the measurements. The authors in [**FN95**] examined how the resource consumptions can influence the runtimes in a resource sharing platform, and they observed that the run time grew as the program requested more nodes for the computation. In the high performance cluster we use, it is possible to assign priority to the jobs we submitted. Since the computations with higher priority will be more competitive for the resource, it is also possible to test the influence of different priorities to execution times.

In the paper [**PVB$^+$19**], the authors propose a guideline consisting of several core principles for performance evaluation in the cloud computation. A similar set of rules has also been proposed for parallel computing systems [**HB15**]. Those principles can also be applied to the large scientific computation in HPC, so we summarize them in the following paragraph. Note that some of the principles have close relation to the reproducibility.

(1) Clear documentation.

Researchers should have a clear goal before the experiment. The setup of the experiment need to be clearly documented. The setup should include hardware and software information, environmental parameters, configurations, prerequisite and external dependencies. It would be best to show how to reproduce the results step by step, even it is only for a naive instance.

(2) Meaningful metric.

The measured metrics should be meaningful and simple enough for readers to understand. Usually the community has its own standard metric for performance evaluation. In certain rare cases, the reason for choosing a different metric needs to be addressed.

(3) Repetition.

The experiment which gathers measurements (e.g. execution times) is expected to be run for multiple times by the original experimenters. This principle at least makes sure the benchmark work is reproducible by the experimenters themselves. The repetitions may occur in different levels, and the experimenters should decide the number of necessary repetitions and describe the number clearly in the documentation.

(4) Statistical analysis.

The aggregated value, the confidence interval, $p-$value, assumptions, and other applicable and important results should be reported. Even if the confidence interval is very small, it is still valuable to report it and show that the variance of the experiment is small. In terms of the assumptions, do not use normal assumptions without normality check. If the community doesn't have a standard method to compute one aggregated value, it is suggested that arithmetic mean should be used for summarizing execution times, harmonic mean for rates and geometric mean for ratios. There are parametric and non-parametric methods to compute, for example, confidence intervals, it is best to describe the function used for the confidence interval computation. In cases where there are more than one available analysis method, some reasoning on why certain methods are chosen is always recommended.

(5) Trade-off.

The cost for the experiment should also be included to see the trade-offs. It is in general not ideal to get 10% execution time speedup but consume $10X$ more resource. The cost may require different statistical analysis strategy, for example the 99% quantile of system latency may be more useful compared with the mean value. Again, the method for computing one aggregated value of the cost should also be stated clearly.

(6) Open Source.

It is understandable that proprietary data is more appropriate in certain cases due to privacy or profit reasons. Under certain regulations, researchers are encouraged to make the experiment available to the scientific community, including source code and experiment data. If an open sourced experiment link/website is provided, it should stay valid and well maintained for a long enough period of time for other researchers to do reproducibility studies. An open source license is commonly used to protect contributors and users.

3.3. Evaluation of reproducibility

Here we evaluate a well known benchmark in the mixed integer programming community: MIPLIB 2017 [**mip18**] based on the six principles in Section 3.2. MIPLIB is a library of mixed integer programming test cases which are tested on multiple optimization solvers, including both open-sourced and commercial ones. MIPLIB 2017 is the sixth generation of MIPLIB library which was firstly created in 1992. The benchmark set consists of three categories of mixed integer programming problems: easy, hard and open problems. Easy problems can typically be solved within one hour, hard problems takes longer time to solve or with a specialized algorithm, open problems are not solved to optimality or even can't prove or disprove feasibility. We evaluate MIPLIB 2017 as follows.

(1) Clear documentation.

Overall MIPLIB 2017 has a clear documentation. The key component in the documentation is the instance selection methodology. The instances in the MIPLIB library serves as a standard library to test robustness of various optimization solvers. The method to select all instances is explained in detail in the MIPLIB 2017 website [**mip18**]. First about 5000 test cases are collected from the contributors around the world and previous MIPLIB libraries. The selection methodology is based on constraint and variable features so that the instances can best represent today's MIP instances. Every test instance has a unique instance name. After trivial presolving, the instance statistics including the size and class of constraints are recorded.

(2) Meaningful metric.

In MIPLIB 2017, for every combination of an instance and an optimization solver, the performance metric is the solving time, which is standard in the optimization community. If one instance cannot be solved within the given time limit, the solving time is recorded as the time limit. The shifted geometric mean is the standard metric in the MIP community to measure the solver performance. The shifted geometric means are reported in MIPLIB 2017 for all measured solvers.

(3) Repetition.

Based on the documentation of the source code in MIPLIB 2017, the number of repetitions can be specified by the user when reproducing the benchmark experiment. However,

every solving time reported in the MIPLIB 2017 website is based on one single solve without repetitions.

One possible pitfall in the MIPLIB library or in general optimization community is the lack of repetition. Although it is possible to solve certain easy problems multiple times to get more accurate solving time, the interesting cases researchers care about are usually hard and open problems. The fact that, these problems either take a very long time to solve or are currently unsolvable, makes repetitions either extremely inefficient or completely useless. In the MIP community, the shifted geometric mean is computed for the entire benchmark set. Therefore, even if some instances' solving times are outliers, the performance metric will not shift too much.

(4) Statistical analysis.

Based on the documentation of the source code in MIPLIB 2017, certain aggregation function is used to compute a single value as the solving time, if there are multiple repetitions. However, due to the author's limited knowledge of AWK, it is unclear to the author what aggregation function is used.

(5) Trade-off.

In the optimization community, the solving time is of interest while the memory usage is usually not as important. In MIPLIB 2017, there is no such tradeoff discussed.

(6) Open Source.

In terms of the source of code and data, the website of MIPLIB 2017 provides download links for all test instances, scripts, solutions and data. The website is trying to be transparent from instances selection to scripts and data of the experiment, which is both beneficial and encouraging for researchers to perform reproducibility studies. The website is also consistently updated to indicate improvements including less solving time, first feasible solution, better feasible solution, provable optimal solution, or provable infeasibility. Thus, an open problem can be changed to an easy or hard problem due to researchers' effort. The success of the MIPLIB library is due to a large number of contributions from both researchers and optimization solver companies, and such extensive collaboration wouldn't be efficient if the work was kept proprietary. In term of the license, there is no open source license associated with the MIPLIB library.

3.4. Statistical foundation

In this section, we review technical details of several statistical methods which are commonly used in analyzing the gathered measurements. Some statistical methods are applied to our benchmark experiment data, and we will report computational results in Section 3.7. The statistical results in this section are not new results, they can usually be found in standard probability or statistics text book, if reference is not provided.

Suppose that the random variable X represents the execution time of some program. The population mean (or true mean) of X is the expectation of X denoted by $E[X]$. We usually use μ to denote the population mean of X. The sample mean, denoted by $\bar{x}$, is an estimate of the population mean $E[X]$. The sample mean is a point estimation of the population mean, whereas the confidence interval is an interval estimation of the population mean.

One important theorem in statistical analysis of benchmark works is the central limit theorem, which can be used to approximate the population mean with unknown or complicated distributions, given the sample size is large enough. The theorem can also be used to compute the confidence interval of the population mean. We include the formal theorem here which can be found in any standard probability textbook.

THEOREM 3.4.1 (Central Limit Theorem). *Given $X_1, X_2, ..., X_n$ are i.i.d. random variables with mean μ and finite variance σ^2, then the random variable $\sqrt{n}\,(\frac{\sum_{i=1}^{n} X_i}{n} - \mu)$ converges in distribution to the normal distribution $N(0, \sigma^2)$.*

3.4.1. Check normal distribution. A large number of statistical analysis methods require normality assumptions. Before applying any rigorous analysis, it is important to verify that the normality assumption holds, if required [**GZ12**]. Given a set of measurements, quantile-quantile plot (Q-Q plot) is a graphical method to quickly observe whether the measurements come from a normal distribution. The Q-Q plot is a scatter plot containing the points (x_i, y_i), where x_i is the $i\%$ percentile of the Gaussian distribution and y_i is the $i\%$ percentile of the measurements. Typically i ranges from 1 to 99. If the scattered points appear in a straight line, then it is likely that the measurements come from a normal distribution. Otherwise, the normality assumption may not hold and statistical analysis methods that do not require normality assumptions should be used. Suppose all measurements are divided into different groups, the mean in each group is called the batched mean value. For skewed distribution, batched mean values are approximately normal

distributed theoretically by the central limit theorem. Nevertheless it is necessary for researchers to check that the batch size is large enough such that batched means are indeed approximately normal using Q-Q plot [**HCA15**].

3.4.2. Confidence interval computation. The confidence interval for the population mean is a range of values which has a given probability of containing the actual value. We explain the most commonly used method as follows.

If the number of measurement is large, usually at least 30, the sample mean can be well approximated by a normal distribution. Specifically, suppose we have n measurements $\{x_1, x_2, ..., x_n\}$, which are independent and follows some distribution with mean μ, and variance σ^2. Based on the central limit theorem, the sample mean $\bar{x}$ approximately follows a normal distribution with mean μ and variance $\frac{\sigma^2}{n}$. We are aimed at finding an interval $[c_1, c_2]$, such that the probability of the actual mean value being included in the interval $[c_1, c_2]$ is $1-\alpha$. The typical values of α are $0.1, 0.05$ or 0.01. The interval $[c_1, c_2]$ is called the confidence interval for the mean value $\bar{x}$ with significance level α and confidence level $1-\alpha$.

To calculate the confidence interval $[c_1, c_2]$, we first need to normalize $\bar{x}$. Let $z = \frac{\bar{x}-\mu}{\sigma/\sqrt{n}}$, and assume z follows the standard Gaussian distribution, which is the special normal distribution with mean 0 and variance 1. Given the significance level α, we can find one z-score $z_{1-\alpha/2}$, such that $P(z \in [-z_{1-\alpha/2}, z_{1-\alpha/2}]) = 1-\alpha$. We can translate the probability to

$$P\Big(\mu \in \Big[\bar{x} - z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}}, \bar{x} + z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}}\Big]\Big) = 1-\alpha.$$

However, the standard deviation σ is usually unknown, so we use the sample standard deviation $s = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$ to approximate σ. Therefore, the confidence interval for the actual mean μ is $[\bar{x} - z_{1-\alpha/2}\frac{s}{\sqrt{n}}, \bar{x} + z_{1-\alpha/2}\frac{s}{\sqrt{n}}]$.

On the other hand, if the number of the measurements is small, for example less than 30, then the normalized value z follows the t distribution with $n-1$ freedoms. The confidence interval can be computed analogously. Specifically $[\bar{x} - t_{1-\alpha/2,n-1}\frac{s}{\sqrt{n}}, \bar{x} + t_{1-\alpha/2,n-1}\frac{s}{\sqrt{n}}]$ is the confidence interval of the actual mean with confidence level $1-\alpha$, where $t_{1-\alpha/2,n-1}$ is the critical value of the t distribution with $n-1$ freedoms.

3.4.3. Hypothesis test. In a hypothesis testing, there are two mutually exclusive hypotheses. One hypothesis is called the null hypothesis, and the other one, which is usually the negation

of the null hypothesis, is called the alternative hypothesis. Hypothesis testing uses the current observations to decide whether there is enough evidence to say the null hypothesis is false and therefore accept the alternative hypothesis, under a given significance level α which is usually 5%. The intuition of hypothesis testing is that the event with low probability is unlikely to happen. The lower the probability is, we are more confident that the event will not happen. The key element in hypothesis testing is the test statistic. The test statistic is a random variable which is calculated by sample data and it follows a known distribution under null hypothesis. The distribution of the test statistic is used to calculate the probability of the occurrence of current observations against the alternative hypothesis. The calculated probability is called the $p-$value. If the $p-$value is less than the significance level, then it is unlikely that our assumption is true so we reject the null hypothesis. Of course the event with low probability may happen and thus we may make a wrong decision to reject the null hypothesis, but the probability of mistakingly rejecting null hypothesis (Type I error) does not exceed the significance level.

Note that the null hypothesis and the alternative hypothesis are not in symmetric positions. Usually the null hypothesis contains some "equal" relation which makes it easier to compute the distribution and probability under the null hypothesis. The significance level only controls the probability of mistakingly rejecting the null hypothesis, and it has no control of mistakingly not rejecting the null hypothesis (Type II error).

Here we explain two types of hypothesis testing. First type of hypothesis testing is regarding whether one group of samples is from a given distribution or two groups of samples are from the same distribution. Second type is to determine whether two groups of samples have equal variance.

The Kolmogorov-Smirnov test (K-S test) is a hypothesis testing, aimed at deciding whether samples are drawn from a specific distribution (called one sample) or two groups of samples are drawn from the same distribution (called two samples). The idea of K-S test is based on the cumulative distribution function (CDF), and the test statistic used in the hypothesis testing is the maximum distance of two cumulative distribution functions. We only explain the one sample K-S test since it can be applied to test goodness of fit [**MJ51**], i.e. determine whether the running time distribution follows a given distribution. The two sample K-S test is similar, where one actual distribution is replaced by an empirical cumulative distribution function. Suppose $\{x_1, x_2, ..., x_n\}$ is one group of samples and function $F(x)$ is the CDF of the tested distribution. First define the

empirical cumulative distribution function (ECDF) of the sample to be

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{[-\infty,x_i]}x.$$

The test statistic is $D_n = \max|F_n(x) - F(x)|$, which has the property that $\sqrt{n}D_n$ converges to Kolmogorov distribution in probability. If D_n is greater than the critical value in the K-S table for a given the significance level, then the null hypothesis is rejected. Note that in order to apply the K-S test, the tested distribution must be continuous and fully specified. If the distribution parameters are unknown and must be estimated from the data, then the original K-S test is conservative to reject the null hypothesis. There are works for modified K-S tests for distributions with estimated parameters, and critical values are recomputed [**ESP88**, **WMDC83**, **Lil67**].

Mann–Whitney–Wilcoxon U-test (MWW test) is a non parametric test which is used to determine whether two groups of samples are drawn from the same distribution. Given that samples $\{x_1, x_2, ..., x_n\}$ are drawn from one distribution and samples $\{y_1, y_2, ..., y_m\}$ are drawn from the other distribution, and the null hypothesis is that these two distributions are the same. For every pair of (x_i, y_j), we compare whether x_i is greater or y_j is greater. Let U_x be the total number of pairs where x_i is greater and U_y be the total number of pairs where y_j is greater. Observe that $U_x + U_y = nm$. If the null hypothesis is true, then intuitively we can guess that U_x and U_y are roughly the same. If one of U_x or U_y is close to 0, then it is likely that the null hypothesis is false. The critical value of the fixed sample size n, m and the fixed significance level α can be found in the Mann–Whitney table. If the sample size is large enough, then the statistic $U = \min(U_x, U_y)$ can be approximated by a normal distribution with mean $\frac{nm}{2}$ and variance $\frac{nm(n+m+1)}{12}$ under the null hypothesis. Then the corresponding p-value can be computed based on the approximated normal distribution.

Levene's test and Bartlett's test can be used to test whether two or more groups of samples have equal variance. We explain the details of these two tests and apply them to test whether CPU time and wall clock time have equal variance in the Section 3.7.

In Levene's test, the null hypothesis is that k groups of samples have equal variance and the alternative hypothesis is that there exist two groups of samples with different variance. Suppose there are N_i data points in the $i-$th group, and Y_{ij} is the $j-$th data point in the $i-$th group. We

abuse the notation j to index data points in every group, although the sample size may differ. The test statistic is defined as

$$W = \frac{N-k}{k-1} \frac{\sum_{i=1}^{k} N_i(\bar{Z}_{i.} - \bar{Z}_{..})^2}{\sum_{i=1}^{k}\sum_{j=1}^{N_i}(Z_{ij} - \bar{Z}_{i.})^2}.$$

In the above formula, $N = \sum_{i=1}^{k} N_i$, $Z_{ij} = |Y_{ij} - \bar{Y_{i.}}|$ where $\bar{Y_{i.}}$ is the mean of Y_{ij} in the $i-$th group, $\bar{Z}_{i.}$ is the mean of Z_{ij} in the $i-$th group, and $\bar{Z}_{..}$ is the mean of all Z_{ij}. The statistic W approximately follows the F distribution with $k-1$ and $N-k$ as degrees of freedom. If W is larger than the critical value with corresponding significance level α, the null hypothesis is rejected. Modifications of Levene's test are studied, and median or trimmed mean is used in the definition of Z variables instead of mean in [**BF74**].

Bartlett's test serves the same goal as Levene's test, but Bartlett's test only works for distributions which do not depart from normality much. Using the same notation as in Levene's test, the test statistic in Bartlett's test is defined to be

$$\frac{(N-k)\log(\sum_{i=1}^{k}(N_i-1)s_i^2/(N-k)) - \sum_{i=1}^{k}(N_i-1)\log s_i^2}{1 + [(\sum_{i=1}^{k} 1/(N_i-1)) - 1/(N-k)]/(3k-3)}.$$

The value s_i^2 is the sample variance in the $i-$th group. The statistic T follows the Chi-square distribution with $k-1$ degrees of freedom. If T is larger than the critical value with corresponding significance level α, the null hypothesis is rejected. Note that Levene's test works for a broader distribution family and therefore is more conservative than Bartlett's test.

3.4.4. Linear regression. Inspired by [**MF17**], whose authors used a linear regression model to eliminate the system errors, we introduce the basic knowledge of linear regression model and how to compute the confidence intervals of the coefficients of parameters.

If the execution time of a single run is too short to measure directly, the cumulative execution time of multiple runs can be considered. A linear regression model is used in [**MF17**] to compute the confidence interval of the single run execution time. Based on their simulation, it is shown that the linear regression model has better performance than computing the confidence interval directly from a set of gathered single run execution times.

Given a set of observations (y_i, x_i) where $i = 1, 2, ..., n$, suppose we know the pair (y, x) satisfies some theoretical linear relation $y = ax + b$, and the goal is to compute the coefficients a, b based on

observations. The best estimates of a, b are trying to fit a line to these data points. The objective is to minimize the error in the L^2 norm.

$$\min_{a,b} \sum_{i=1}^{n} |y_i - ax_i - b|^2$$

A closed formula of the optimal slope and intercept value can be found.

$$a^* = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}.$$

$$b^* = \bar{y} - a^*\bar{x}.$$

The optimal values a^*, b^* only give scalar estimates. Due to the randomness in the measurements, we can also find the confidence interval of the slope and intercept. Suppose the error follows a normal distribution with mean 0 and variance σ^2. Define the mean square error $MSE = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n-2}$ which is an unbiased estimate of σ^2. It can be shown that $\frac{a-a*}{\sqrt{MSE/\sum_{i=1}^{n}(x_i-\bar{x})^2}}$ and $\frac{b-b*}{\sqrt{MSE/n}}$ both follow a student t-distribution with $n-2$ freedoms. Therefore, given a confidence level $1-\alpha$, the confidence interval of the slope value a is $[a^* - t_{1-\alpha/2;n-2} \times \sqrt{\frac{MSE}{\sum_{i=1}^{n}(x_i-\bar{x})^2}}, a^* + t_{1-\alpha/2;n-2} \times \sqrt{\frac{MSE}{\sum_{i=1}^{n}(x_i-\bar{x})^2}}]$, and the confidence interval of the intercept b is $[b^* - t_{\alpha/2;n-2} \times \sqrt{\frac{MSE}{n}}, b^* + t_{\alpha/2;n-2} \times \sqrt{\frac{MSE}{n}}]$. If n is large enough, then $t_{1-\alpha/2;n-2}$ can be approximated by $z_{1-\alpha/2}$.

3.4.5. Bootstrap. In terms of execution times, the actual distribution is unknown and distribution type needs to be specified in a parametric analysis. Bootstrap is a widely used non parametric method, which doesn't have a specified distribution with parameters (e.g. normal distribution). In this subsection, we briefly explain how to use bootstrapping to compute confidence interval of the mean and hypothesis testing for determining whether two groups of measurements have the same mean.

Given a set of measurements $\{x_1, x_2, ..., x_n\}$. Suppose we are aimed at computing the confidence interval with 95% confidence level. Each time we randomly draw n measurements with replacement and compute the mean, denoted by $\bar{x}_i^*$. Usually this process is done N times with $N >> n$. The gathered mean values set $M = \{\bar{x}_1^*, \bar{x}_2^*, ..., \bar{x}_N^*\}$ is considered as an empirical distribution of the mean. Then $[Q_{0.025}, Q_{0.975}]$ is called the percentile bootstrap confidence interval where $Q_{0.025}$ and

$Q_{0.975}$ are the 2.5% and 97.5% percentile of the set M. The other interval $[\bar{x} - Q_{0.975}, \bar{x} - Q_{0.025}]$ is called the basic bootstrap confidence interval, where $\bar{x}$ is the mean of the original n measurements.

Suppose we want to figure out whether one algorithm is faster than the other algorithm when solving a problem. For each algorithm, we gather the executions times with multiple runs. Let $\{x_1, x_2, ..., x_n\}$ be the solving times for one algorithm and $\{y_1, y_2, ..., y_m\}$ be the solving times for the other algorithm. We are aimed at using hypothesis testing to determine whether these two algorithms have the same mean. Let $\bar{x}, \bar{y}$ and σ_x^2, σ_y^2 be the sample means and sample variances of the measurements in the two algorithms. Without loss of generality, we assume $\bar{x} > \bar{y}$. Compute the statistic

$$t = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{\sigma_x}{n} + \frac{\sigma_y}{m}}}.$$

We want to compute the $p-$value under the null hypothesis. First we need to normalize the values so that they meet the null hypothesis, i.e. have the same mean. Define $x_i' = x_i - \bar{x} + \bar{z}$ and $y_j' = y_j - \bar{y} + \bar{z}$, where $\bar{z}$ is the mean of all x_i and y_j. Each time we randomly draw n measurements from $\{x_1', x_2', ..., x_n'\}$ and m measurements from $\{y_1', y_2', ..., y_m'\}$ both with replacement, and repeat for N times. Compute the statistic

$$t_k = \frac{\overline{x'_{(k)}} - \overline{y'_{(k)}}}{\sqrt{\frac{\sigma_{x(k)}}{n} + \frac{\sigma_{y(k)}}{m}}}, \quad k = 1, 2, ..., N.$$

In the above formula, $\overline{x'_{(k)}}, \overline{x'_{(k)}}$ and $\sigma_{x(k)}^2, \sigma_{y(k)}^2$ represent the means and variances of the kth sampling. The empirical distribution of the statistic t is just $\{t_1, t_2, ..., t_N\}$. The p-value is computed as

$$\frac{\sum_{k=1}^{N} I\{t_k > t\}}{N}.$$

For instance, 5% significance level is given. If the p-value is less than 5%, then we reject the null hypothesis and determine that there is enough evidence to believe that the actual mean of x_i is larger than the actual mean of y_j.

3.5. Stopping rules

The sequential stopping rules are designed to determine when collected data is sufficient for research goal and the collection of data can stop. Some data is expensive to collect both in time cost and financial cost. In the clinical research field, it is ideal to draw a conclusion with minimum number of samples prepared in order to save cost [**HP88**]. A more general optimization is using a so-called loss function, which is usually defined to be $L = w \times var + c \times n$. In the loss function definition, n is the sample size, var represents some type of sample variation (for example the sample variance), c is a constant defining the cost of draw one sample, and w represents the weight of variance. Given w and c, the optimization problem is to minimize the loss function, and such study is under the name of minimum risk point estimation problem. The loss function has been used in the sequential stopping rules for estimation of the mean [**CM$^+$82**]. Besides estimation of the mean, the estimation of coefficient of variation is also studied using a sequential method [**CK16**]. We do not elaborate on the minimum risk point estimation problem, instead we refer interested readers to studies with different distribution assumptions: normal distribution [**Sta66**], exponential distribution [**SW72**], Poisson distribution [**Var79**], and distribution free [**CY$^+$81**, **GM79**].

Confidence interval procedure is another important topic besides point estimation. The general stopping rules are based on the quality of computed confidence interval. Specifically, the collection of data ends when the confidence interval is small enough. There are two senses of the "smallness", absolute precision and relative precision. Suppose the confidence interval of the (positive) mean is $[\mu - \epsilon, \mu + \epsilon]$. If absolute precision δ is used, then stop collecting if $\epsilon < \delta$. If relative precision δ is used, then stop collecting if $\epsilon < \delta\mu$. Generally it requires larger sample size to achieve results with higher quality. However, certain stopping rules may have early stopping in some instances even with small precision, for example the first few data are very close to each other by chance. But early stopping has an issue that it can have overall bad coverage [**LK82**], meaning the confidence interval has high probability to miss the true mean. Therefore, it is best to use a so-called pilot phase to gather a minimum number of data points [**Sin14**, **LK82**].

The most common method to compute confidence interval is based on the normality assumption, and we have explained the detail in Section 3.4. There is also research on confidence interval procedures with other skewed distributions. We focus on the lognormal distribution, based on which we will explain one stopping rule for our benchmark experiment in Section 3.7. Different

methods of confidence interval computation for lognormal distribution can be found in [**Ols05**]. We summarize one method here known as Cox's method. Suppose samples $\{x_1, x_2, ..., x_n\}$ are drawn from a lognormal distribution and $\{y_1, y_2, ..., y_n\}$ are log-transformed data, i.e. $y_i = \log x_i$. The goal is to estimate the mean (namely θ) of the lognormal distribution. The interval $[\bar{y} + \frac{s^2}{2} - z_{1-\alpha/2}\sqrt{\frac{s^2}{n} + \frac{s^4}{2(n-1)}}, \bar{y} + \frac{s^2}{2} + z_{1-\alpha/2}\sqrt{\frac{s^2}{n} + \frac{s^4}{2(n-1)}}]$ is one confidence interval for $\log\theta$, where $\bar{y}$ and s^2 are the sample mean and sample variance of the transformed data y_i. The value $z_{1-\alpha/2}$ is the two sided critical value of the standard normal distribution, i.e. $P(-z_{1-\alpha/2} \leq X \leq z_{1-\alpha/2}) = 1-\alpha$ with X standard normal distribution. The confidence interval of the parameter θ can be found by anti-log transformation. Specifically, the interval

$$\Big[\exp(\bar{y} + \frac{s^2}{2} - z_{1-\alpha/2}\sqrt{\frac{s^2}{n} + \frac{s^4}{2(n-1)}}),\ \ \exp(\bar{y} + \frac{s^2}{2} + z_{1-\alpha/2}\sqrt{\frac{s^2}{n} + \frac{s^4}{2(n-1)}})\Big] \tag{3.1}$$

is the confidence interval with confidence level α. Note that the point estimation of the mean is $\exp(\bar{y} + \frac{s^2}{2})$ which is not the midpoint of the confidence interval. To compute the half width of the confidence interval, choose the right (and larger) half interval, i.e. $[\exp(\bar{y} + \frac{s^2}{2}), \exp(\bar{y} + \frac{s^2}{2} + z_{1-\alpha/2}\sqrt{\frac{s^2}{n} + \frac{s^4}{2(n-1)}})]$. By a straightforward calculation, the claim that the confidence interval (3.1) has relative precision δ is equivalent to the following inequality

$$z^2_{1-\alpha/2}(\frac{s^2}{n} + \frac{s^4}{2(n-1)}) \leq \log^2(1+\delta). \tag{3.2}$$

The sequential analysis on estimation of the mean of lognormal samples are studied in [**Nag80**, **Zac66**]. The authors proposed stopping rules such that the point estimation $\hat{\theta}$ has predefined relative precision δ, i.e. $P(|\hat{\theta}-\theta| \leq \theta\delta) \geq 1-\alpha$ where $1-\alpha$ is the confidence level. The two stopping rules in these two papers are slightly different. In paper [**Nag80**], the collection of data terminates if $n \geq \chi^2_{1-\alpha}(1)\delta^{-2}s^2(1 + \frac{1}{2}s^2)$ where s^2 is the sample variance of the log-transformed data and $\chi^2_{1-\alpha}(1)$ is the critical value in chi-square distribution with degree 1, i.e. $P(X^2 \leq \chi^2_{1-\alpha}(1)) = 1-\alpha$ with X is standard normal distribution. In paper [**Zac66**], the author used $\log(1+\delta)$ instead of δ. We prove that the stopping rules proposed in [**Nag80**,**Zac66**] are almost equivalent to the stopping rule based on Cox's method. To the best of the author's knowledge, this "near equivalence" is a new result.

THEOREM 3.5.1. *Given a sequence of i.i.d. random variables $\{Y_n\}$, denote $\bar{Y_n}$ to be the sample mean and S_n^2 to be the sample variance of the first n random variables, i.e. $\bar{Y_n} = \frac{\sum_{i=1}^n Y_i}{n}$ and*

$S_n^2 = \frac{\sum_{i=1}^n (Y_i - \bar{Y_n})^2}{n-1}$. *Assume that there exist* $\sigma^2 > 0$ *such that the sequence of random variables* $\{S_n^2\}$ *converges pointwise to* σ^2, *i.e.* $\lim_{n\to\infty} S_n^2 = \sigma^2$. *Given* $\alpha \in [0,1]$ *and* $\delta > 0$, *define two random variables* $G(\delta) = \min\{n\colon n \geq \chi^2_{1-\alpha}(1)\delta^{-2}S_n^2(1 + \frac{1}{2}S_n^2)\}$ *and* $H(\delta) = \min\{n\colon z^2_{1-\alpha/2}(\frac{S_n^2}{n} + \frac{S_n^4}{2(n-1)}) \leq \log^2(1+\delta)\}$ *with* α *fixed. Then we have* $\lim_{\delta\to 0} \frac{H(\delta)}{G(\delta)} = 1$.

In Theorem 3.5.1, the pointwise convergence assumption on the sample variance S_n^2 is stronger than convergence in probability, which can be derived from the central limit theorem under mild assumptions. If Y_i are i.i.d. random variables with mean μ and variance σ^2 and the first four moments exist, then the sample variance converges to σ^2 in probability.

Proof. Note that Theorem 3.5.1 is a straightforward implication of the following Lemma 3.5.1. We need to show that for any w in the probability space, $\lim_{\delta\to 0} \frac{H(\delta)(w)}{G(\delta)(w)} = 1$, which is exactly the claim of Lemma 3.5.1 when $y_n = Y_n(w)$, $\min A(\delta) = G(\delta)(w)$ and $\min B(\delta) = H(\delta)(w)$. □

Lemma 3.5.1. *Given a sequence* $\{y_n\}$, *denote* $\bar{y_n}$ *to be the sample mean and* s_n^2 *to be the sample variance of the first* n *numbers, i.e.* $\bar{y_n} = \frac{\sum_{i=1}^n y_i}{n}$ *and* $s_n^2 = \frac{\sum_{i=1}^n (y_i - \bar{y_n})^2}{n-1}$. *Assume there exist* $\sigma^2 > 0$ *such that* $\lim_{n\to\infty} s_n^2 = \sigma^2$. *Given* $\alpha \in [0,1]$ *and* $\delta > 0$, *define* $A(\delta) = \{n\colon n \geq \chi^2_{1-\alpha}(1)\delta^{-2}s_n^2(1 + \frac{1}{2}s_n^2)\}$ *and* $B(\delta) = \{n\colon z^2_{1-\alpha/2}(\frac{s_n^2}{n} + \frac{s_n^4}{2(n-1)}) \leq \log^2(1+\delta)\}$ *with* α *fixed. Then we have* $A(\delta), B(\delta)$ *are both non empty and* $\lim_{\delta\to 0} \frac{\min B(\delta)}{\min A(\delta)} = 1$.

To prove Lemma 3.5.1, we first introduce the following Lemma 3.5.2.

Lemma 3.5.2. *Given a sequence* $\{y_n\}$, *denote* $\bar{y_n}$ *to be the sample mean and* s_n^2 *to be the sample variance of the first* n *numbers, i.e.* $\bar{y_n} = \frac{\sum_{i=1}^n y_i}{n}$ *and* $s_n^2 = \frac{\sum_{i=1}^n (y_i - \bar{y_n})^2}{n-1}$. *Assume there exist* $a, b > 0$ *and* $n_0 \in \mathbb{N}$, *such that* $a < s_n^2 < b$ *for any* $n > n_0$. *Given* $\alpha \in [0,1]$ *and* $\delta > 0$, *let* $A = \{n\colon n \geq \chi^2_{1-\alpha}(1)\delta^{-2}s_n^2(1 + \frac{1}{2}s_n^2), n > n_0\}$ *and* $B = \{n\colon z^2_{1-\alpha/2}(\frac{s_n^2}{n} + \frac{s_n^4}{2(n-1)}) \leq \log^2(1+\delta), n > n_0\}$. *Then we have* A, B *are both non empty and* $\min A \leq \min B \leq \lceil \frac{b^2}{a^2} \min A(1 - \frac{\delta}{2})^{-2} \rceil + 1$.

PROOF OF LEMMA 3.5.2. We first prove $B \subset A$. For any $n \in B$, we have

$$\begin{aligned}
& z_{1-\alpha/2}^2\left(\frac{s_n^2}{n}+\frac{s_n^4}{2(n-1)}\right) \leq \log^2(1+\delta) \\
\Rightarrow\ & z_{1-\alpha/2}^2\left(\frac{s_n^2}{n}+\frac{s_n^4}{2(n-1)}\right) \leq \delta^2 \\
\Rightarrow\ & z_{1-\alpha/2}^2\left(\frac{s_n^2}{n}+\frac{s_n^4}{2n}\right) \leq \delta^2 \\
\Leftrightarrow\ & z_{1-\alpha/2}^2 s_n^2\left(1+\frac{1}{2}s_n^2\right)\delta^{-2} \leq n \\
\Leftrightarrow\ & n \geq \chi_{1-\alpha}^2(1)\delta^{-2}s_n^2\left(1+\frac{1}{2}s_n^2\right).
\end{aligned}$$

The last step is from the definition of the chi-square distribution. Suppose random variable X follows standard normal distribution then X^2 follows the chi-square distribution with degree 1. Then

$$P(X^2 \leq \chi_{1-\alpha}^2(1)) = 1-\alpha = P(-z_{1-\alpha/2} \leq X \leq z_{1-\alpha/2}) = P(X^2 \leq z_{1-\alpha/2}^2).$$

Therefore $\chi_{1-\alpha}^2(1) = z_{1-\alpha/2}^2$. Then $B \subset A$ implies $\min A \leq \min B$.

Next we prove the other inequality. Suppose $n = \min A$, use the identity $\log(1+\delta) > \delta - \frac{\delta^2}{2}$, then we have

$$\begin{aligned}
& n \geq \chi_{1-\alpha}^2(1)\delta^{-2}s_n^2\left(1+\frac{1}{2}s_n^2\right) \\
\Rightarrow\ & n \geq \chi_{1-\alpha}^2(1)\delta^{-2}a\left(1+\frac{1}{2}a\right) \\
\Rightarrow\ & \frac{b^2}{a^2}n \geq \chi_{1-\alpha}^2(1)\delta^{-2}b\left(1+\frac{1}{2}b\right) \\
\Rightarrow\ & \frac{b^2}{a^2}n \geq \chi_{1-\alpha}^2(1)\left(1-\frac{\delta}{2}\right)^2\log^{-2}(1+\delta)b\left(1+\frac{1}{2}b\right) \\
\Leftrightarrow\ & \log^2(1+\delta) \geq z_{1-\alpha/2}^2\left(\frac{b}{\frac{b^2}{a^2}n(1-\frac{\delta}{2})^{-2}}+\frac{b^2}{2\times\frac{b^2}{a^2}n(1-\frac{\delta}{2})^{-2}}\right). \\
\Rightarrow\ & \log^2(1+\delta) \geq z_{1-\alpha/2}^2\left(\frac{b}{\lceil\frac{b^2}{a^2}n(1-\frac{\delta}{2})^{-2}\rceil+1}+\frac{b^2}{2\times\lceil\frac{b^2}{a^2}n(1-\frac{\delta}{2})^{-2}\rceil}\right).
\end{aligned}$$

Let $m = \lceil\frac{b^2}{a^2}n(1-\frac{\delta}{2})^{-2}\rceil + 1$, then from the previous inequality and $s_m^2 < b$ we can conclude that

$$z_{1-\alpha/2}^2\left(\frac{s_m^2}{m}+\frac{s_m^4}{2(m-1)}\right) \leq \log^2(1+\delta),$$

which implies that $m \in B$ from the previous proof. Then we know $\min B \leq \lceil \frac{b^2}{a^2} n(1-\frac{\delta}{2})^{-2} \rceil + 1$, which concludes the proof. □

PROOF OF LEMMA 3.5.1. Since sample variance s_n^2 is bounded, it is not hard to see that both $A(\delta)$ and $B(\delta)$ are non empty and $\lim_{\delta\to 0} \min A(\delta) = \lim_{\delta\to 0} \min B(\delta) = +\infty$. From Lemma 3.5.2 we know that $\min A(\delta) \leq \min B(\delta)$. Therefore, we only need to show the following claim: for any $\epsilon > 0$, there exists δ_ϵ, such that $\frac{\min B(\delta)}{\min A(\delta)} < 1 + C\epsilon$ holds for any $0 < \delta < \delta_\epsilon$ where C is a constant independent of ϵ. Without loss of generality, we assume $\epsilon < \frac{\sigma^2}{2}$.

For any fixed $\epsilon > 0$, since the limit exists for s_n^2, there exists $n_\epsilon > \frac{2}{\epsilon}$ such that $\sigma^2 - \epsilon < s_n^2 < \sigma^2 + \epsilon$ for any $n > n_\epsilon$. Since $\lim_{\delta\to 0} \min A(\delta) = \lim_{\delta\to 0} \min B(\delta) = +\infty$, there exist $\delta_\epsilon < \min(\frac{1}{2}, \epsilon)$ such that $\min A(\delta) > n_\epsilon$ and $\min B(\delta) > n_\epsilon$ for any $0 < \delta < \delta_\epsilon$. Then we claim that $\frac{\min B(\delta)}{\min A(\delta)} < 1 + (3 + \frac{32}{\sigma^2})\epsilon$ holds for any $0 < \delta < \delta_\epsilon$.

From $0 < \delta < \delta_\epsilon < \frac{1}{2}$ one can easily prove $(1-\frac{\delta}{2})^{-2} < 1 + 2\delta$ by the Taylor expansion. From Lemma 3.5.2 and the conditions on n_ϵ and δ_ϵ, we know that

$$
\begin{aligned}
\frac{\min B(\delta)}{\min A(\delta)} &\leq \frac{\lceil \frac{(\sigma^2+\epsilon)^2}{(\sigma^2-\epsilon)^2} \min A(\delta)(1-\frac{\delta}{2})^{-2} \rceil + 1}{\min A(\delta)} \\
&\leq \frac{\frac{(\sigma^2+\epsilon)^2}{(\sigma^2-\epsilon)^2} \min A(\delta)(1-\frac{\delta}{2})^{-2} + 2}{\min A(\delta)} \\
&< \frac{(\sigma^2+\epsilon)^2}{(\sigma^2-\epsilon)^2}(1-\frac{\delta}{2})^{-2} + \frac{2}{n_\epsilon} \\
&< \frac{(\sigma^2+\epsilon)^2}{(\sigma^2-\epsilon)^2}(1+2\delta) + \epsilon \\
&= (1 + \frac{4\sigma^2\epsilon}{(\sigma^2-\epsilon)^2})(1+2\delta) + \epsilon \\
&< (1 + \frac{4\sigma^2\epsilon}{(\sigma^2-\frac{\sigma^2}{2})^2})(1+2\delta) + \epsilon \\
&= (1 + \frac{16\epsilon}{\sigma^2})(1+2\delta) + \epsilon \\
&< 1 + 2\delta + 2\frac{16\epsilon}{\sigma^2} + \epsilon \\
&< 1 + (3 + \frac{32}{\sigma^2})\epsilon.
\end{aligned}
$$

Since σ^2 is a constant independent of ϵ, we can conclude that $\lim_{\delta \to 0} \frac{\min B(\delta)}{\min A(\delta)} = 1$. $\square$

To test the robustness and efficiency of a stopping rule in the computer science field, a commonly used method is the Monte Carlo simulation; see examples [**BHT17**, **RG04**]. We explain how researchers perform such Monte Carlo simulation as follows. In every independent run of a simulation, the proposed stopping rule is applied to a sequence of data which is randomly drawn from a given distribution. Then robustness and efficiency are evaluated by computed confidence intervals and final sample sizes from all independent runs. The percentage of independent runs with confidence interval covering the true mean is called (simulation) coverage. The simulation coverage is compared with the nominal coverage, which is the term referring to confidence level set in the confidence interval computation, to determine the robustness. Ideally a robust stopping rule has a coverage close to the nominal coverage. The final sample size can be used to measure how efficiently the stopping rule works. It is best to achieve a good coverage with smallest final sample size. In a real world experiment, the distribution is usually unknown and researchers will pose certain distribution assumptions like normal distribution. If the coverage in the simulation is far from nominal coverage or the average final sample size is much larger than the theoretical final sample size, there is a large probability that the distribution assumption does not hold. In such case, batched means can be used to approximate normal distribution [**TSW10**].

It is also possible that the stopping rule does not depend on any specific distribution (distribution free), for example [**YH19**]. In their paper five different distributions are tested on the proposed stopping rule, and it worked well for four not too skewed distributions. The exception is the lognormal distribution. Non parametric methods, such as bootstrap, can also be used in the confidence interval computation. In the sequential iterated bootstrap method, there are two levels of Monte Carlo simulation. In the inner level, confidence interval is computed by a standard bootstrap method, while the outer level consists of independent runs. The major limitation of the method is that it requires a large resampling size. There are studies trying to optimize the inner level by combining the bootstrap technique and analytical estimations [**DMY92**, **LY95**, **LY96**].

A recent paper [**CW20**] investigated stopping rules for multiple algorithms applying to multiple instances. The goal of the stopping rules is to determine when the collection of data terminates if we are confident that different algorithms have different performance. This type of stopping rules is based on ANOVA and hypothesis testing while the previously explained stopping rules are based

on confidence interval. Although they are closely related, researchers should choose a proper type depending on the aim of the experiment. If the aim is to accurately measure execution time, we recommend to use a stopping rule based on confidence intervals. If the aim is to decide whether different algorithms have the same performance, using ANOVA based stopping rules is preferred.

3.6. Organization of large scale computation

In our benchmark experiment, one goal of the computation is to compare several different algorithms applied to hundreds of test cases. The details of those algorithms make it hard to analyze the running time theoretically. For example, in some algorithms, heuristics are used with branch and bound technique, and optimization solvers are also used to solve linear programs. The use of heuristics and solvers makes the theoretical worst case time complexity of certain algorithms extremely bad. However, by running several test cases on a personal laptop, we discovered that those algorithms could outperform other algorithms which have guaranteed (quadratic) time complexity. Therefore we need to prepare enough test cases in this benchmark work to test which algorithm is more robust.

By testing a few hard cases in a laptop, we found that certain combinations of algorithms and instances didn't finish in 12 hours. The scale of the computation makes it impossible to run a single notebook in a single computer because the overall computation time is too long. So we decide to use the HPC to finish the benchmark work where we can utilize multiple nodes to execute multiple programs simultaneously. The HPC we use is managed by the `slurm` manager. A program will be automatically killed by the `slurm` manager when reaching the preset time limit or running out of memory. After gathering all running times, Jupyter notebook can be used to perform reproducible and reusable data analysis, including confidence interval, performance profile, and visualization [**Per18**].

For benchmarking, it is ideal to wait until the machine reaches the "stable state" to begin the actual measurement [**KJ13**]. However, sometimes this is practically impossible especially when the program is executed in a platform with high resource sharing, like the HPC. If a single execution takes a long time, then reaching to the stable state in a reasonable time is not practical. In our benchmark work, we don't wait for the stable state and we begin gathering measurements as soon as the computational resource is acquired.

In this section, we explain how we organize the benchmark experiment. We make our source code and data[1] available to all. The benchmark experiment was conducted in SageMath on the Peloton cluster[2], where each node has 64GB RAM, 2 sockets, 16 cores, and 32 threads running Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz.

3.6.1. Test cases. We write scripts to generate all test cases, which should be fixed throughout the computation. For each test case, multiple different algorithms may be tested. We introduce one unique key for every test case. The unique key of each instance is convenient for users to perform reproducible data analysis in a Jupyter notebook.

3.6.2. Batching. It is ideal to have only one program measured at each node, since in this way we can get rid of the caching/memory issue from previous computations. However, if the actual measurement of a key takes little memory and time, the computation on the node can be considered as a waste of resource. The worst scenario is that the whole computation seems like just starting and exiting the software in all available nodes. In particular, starting and exiting the software in many nodes frequently may cause I/O issues and is harmful for the whole cluster and other user's running programs.

In our benchmark work, we use one node to measure execution times and memory usage for a single test instance and a single algorithm. We set the time limit to 1 hour and maximum memory usage to 8GB for each node and we gather at most 30 measurements. For some small cases, the total execution time would be very short but one node is occupied and released very soon. This "no-batching" strategy does no harm to the precision of the measurement, but it inevitably introduce a overhead to an overall computation time since in a HPC it needs some competitions to get access to a node for the computation.

Once we have an initial experiment and get the approximated execution time for every computation, one future step is to use a batching strategy to properly batch several instances to one node and try to make the most of the (1 hour) time limit. We should be careful about the memory used by previous computations, and it is necessary to use some kind of garbage collector in the software to release memory. On the other hand, by reporting the memory usage in every repetition the experiment itself can help to find memory leak problems in the software, if garbage collector

[1]Github repository: https://github.com/mkoeppe/jiawei-computations
[2]https://wiki.cse.ucdavis.edu/support/systems/peloton

is used correctly. For example, it is reported that solving a mixed integer problem using certain backends could cause memory leak problem[3].

3.6.3. Data management. We store computation results using git submodule technique. In the top level directory, we have all scripts to start the experiment and gather raw data, and it also include a Jupyter notebook which explains how we perform data analysis. All raw data is stored in a submodule included in the top level directory. Different branches of the submodule correspond to different computational tasks. The submodule, which stores raw data, also include another submodule which contains all necessary testing code. Every version of the computational results corresponds to a version of the testing code, and the correspondence can be conveniently seen in the git commit information. The testing code is continuously under development, so it is easy to rerun the experiment to test speedup and store the new data in a new commit without worrying about overwriting the old data.

We provide a detailed documentation about how to rerun the entire experiment in `README.md`. Note that several prerequisite software need to be installed, including SageMath and some optimization solver. The optimization solver needs to be accessible in SageMath. After all computations finish in HPC, we create a git commit that adds the result files and push the submodule to the remote. Then we can analyze data locally using the Jupyter notebook. The notebook serves as a template which has instructions to query necessary data as well as visualization, like shifted geometric mean table and performance profile plots. From the template, other users can perform their customized data analysis.

The following diagram shows the structure the top level repository. Suppose `testing code submodule` is the correct testing code version, a pseudo-experiment works as follows. The `top level repo` exists in both HPC and the local computer. In HPC, use `scripts` to start the experiment. Computations will update the `raw data` folder and cache the testing code version in `raw data version`. After all computations finish, commit and publish the changes in `computational results submodule`. In the local computer, pull the changes in `computational results submodule` and use `jupyter notebook` to perform data analysis. Note that remote `computational results submodule` only gets pushed from HPC and pulled to the local computer, whereas `scripts`[4] only gets pushed from the local computer and pulled to HPC. In this way

[3]https://trac.sagemath.org/ticket/21825

[4]The notebook will be pulled to HPC as well but it will never be used there.

we have a clean data management which separates obtaining raw data and performing data analysis. If more than one version of `raw data` is needed, one can add more copies of `computational results submodule` in the `top level repo` and checkout the desired branch/commit.

```
top level repo
├── scripts
├── jupyter notebook
└── computational results submodule
    ├── testing code submodule
    ├── raw data
    └── raw data version
```

3.7. Computational results

3.7.1. Normality Assumption. We use Q-Q plot (Quantile-Quantile plot) to show whether the measurements (CPU times) follow the normal distribution. In our experiment, we gather (at most) 30 measurements for every instance and every algorithm. Some hard instances may have fewer than 30 measurements or even no measurement, so we only pick a few small instances and we plot the quantiles of the gathered 30 measurements with respect to theoretical standard normal distribution. From Figure 3.1 we can see that some gathered measurements match with the normal distribution while others show a tail to the right.

We computed skew and kurtosis for all 5616 computations with 30 measurements in our experiment, about 74% cases have a positive skew which indicates a tail to the right, and about 65% cases have a positive kurtosis which indicates samples are more clustered to the mean compared to normal distribution. We also plot histograms of skew and kurtosis of the 5616 computations in Figure 3.2.

3.7.2. Distribution fitting. Normal distribution is hardly the best match for the distribution of CPU time. So the question is what family of distribution is a best/better fit. Since there are a large number of distributions available and some may have many parameters so that overfitting can occur, we only focus on some common distributions. We do not make a rigorous claim about what CPU time distribution actually is, instead, we show evidences that there are other distributions which could be a better fit.

We use the `scipy.stats` package to fit the data to various distributions, and use K-S test to determine whether to reject the distribution with estimated parameters. For every given distribution family, there is a standard distribution. From the standard distribution, the function

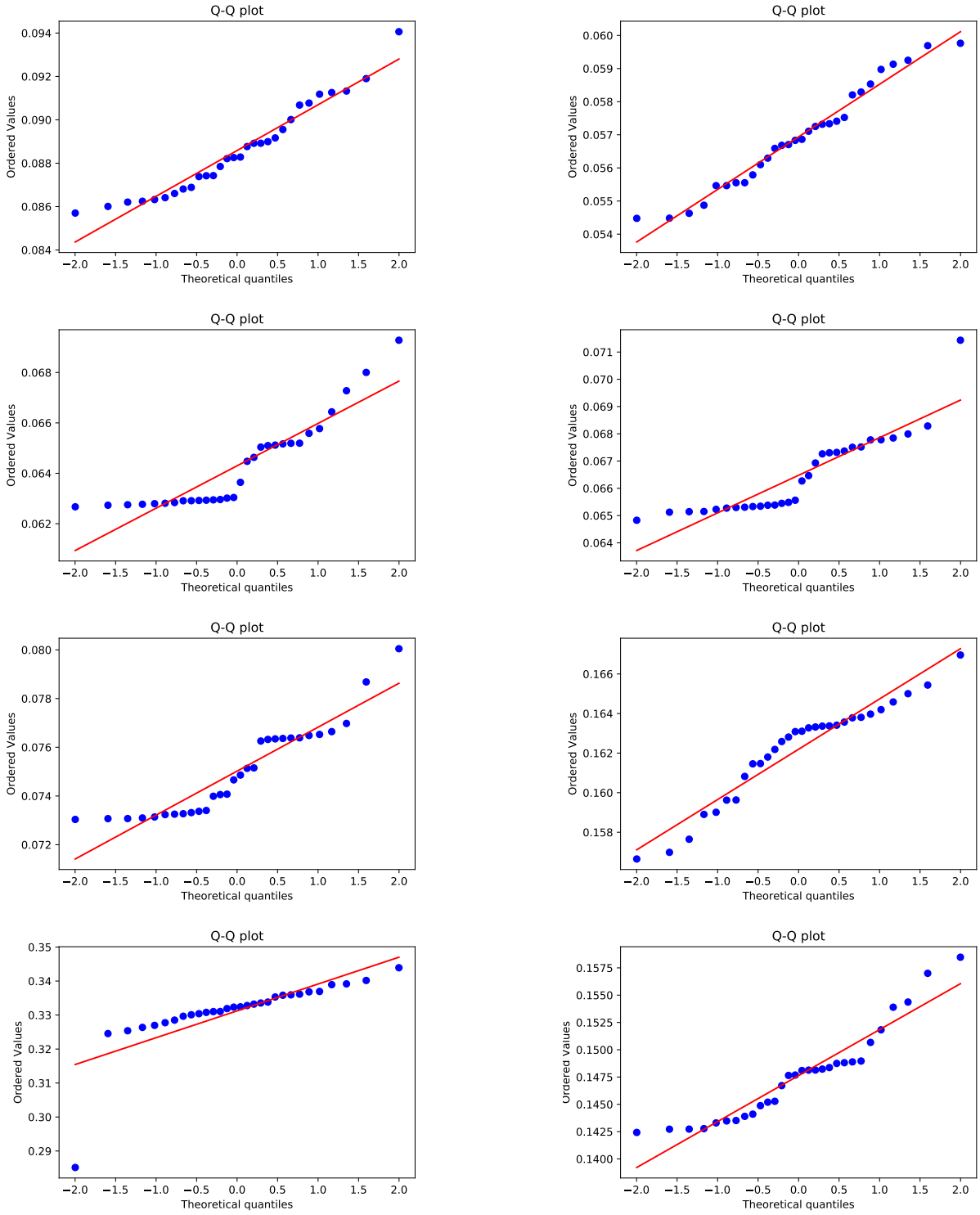

FIGURE 3.1. The Q-Q plots for cpu time measurements in eight randomly chosen instances in computation of $\Delta\pi$ minimum. These eight instances are keyed `extreme_i` for $i \in \{1, 3, 5, 7, 9, 11, 13, 15\}$.

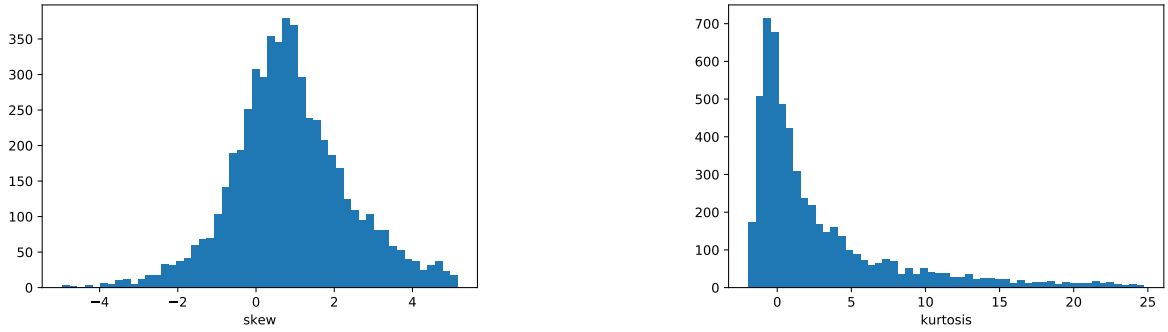

FIGURE 3.2. The left figure shows the skew distribution and the right figure shows the kurtosis distribution.

`scipy.stats.fit` uses maximum likelihood estimation to compute the estimated location, scale and shape parameters. The location parameter specifies the left and right shift of the PDF, and the scale parameter specifies how to stretch the PDF. The shape parameters are other necessary distribution parameters to determine the actual PDF. For example, in the normal distribution, the standard Gaussian distribution is the standard distribution and the mean and variance can be incorporated in the location and scale parameters respectively, so there is no shape parameter in the normal distribution. After the fitting process, we use the function `scipy.stats.kstest` to perform one sample K-S test for goodness of fit.

We still consider all 5616 computations with 30 measurements. One limitation of our distribution fitting process is the sample size. Due to the current experiment design, where at most 30 measurements are collected, it is hard to claim that we have enough samples to distinguish different distributions. We rather demonstrate one framework which uses the current data we have. The same framework can be reused when the benchmark experiment is reproduced and more data are collected. We use all default settings in `scipy.stats.fit`, meaning all location, scale and shape parameters are estimated. Note that the fitting process can take a long time since some maximum likelihood estimation does not have an analytical solution and slow convergence may occur. In the K-S test we use 0.95 as the significance level. We include the distribution fitting results for 7 common distributions in Table 3.1. Some distribution families have no shape parameters while other distribution families have one shape parameter. We observe that the lognormal, t and Cauchy distributions have better fit than the normal distribution. Note that the t distribution is approximately equal to the normal distribution if the degree of freedom is large, so it is likely that the extra shape parameter in the t distribution lead to the better fitting power. It is interesting that the

TABLE 3.1. Goodness of fit for various distributions

	normal	lognormal	t	Cauchy	exponential	Chi	Chi squared
parameters	2	3	3	2	2	3	3
rejection rate	27.58%	21.03%	12.02%	11.89%	53.01%	55.02%	94.62%

Cauchy distribution also has a good fitting result while it only has location and scale parameters. However, it is known that no moment exists for the Cauchy distribution. Modifications like the truncated Cauchy distribution are needed if the mean is of interest, which is out of the scope of the dissertation.

3.7.3. Stopping rules. In this subsection, we investigate sequential stopping rules which can be applied to our benchmark experiment. Note that in our current experiment, which can be reproduced by using the Github repository, we do not use any stopping rules. Instead, we simply gather at most 30 measurements in every computation within the one hour time limit. Nevertheless we can use the current experiment results to investigate possible stopping rules. Due to the one hour limit, we only consider stoping rules for simple cases. In regards to hard instances for which it is relatively expensive to gather one single measurement, we do not consider any stopping rules since there are too few measurements gathered within one hour. The goal of our discussion about stopping rules is to provide one possible improvement of our benchmark experiment. For example if other researchers want to reproduce or redesign the experiment with more available computational resource, they can incorporate a stopping rule into the experiment design and utilize computational resource more efficiently.

We use confidence interval procedure as the stopping rule. Since the running time of different computations varies a lot, we do not use a uniform absolute precision and use a uniform relative precision instead. Specifically, we investigate two purely sequential stopping rules which use relative precision in the confidence interval computation. In contrast to batched sequential stopping rules, purely sequential stopping rules are in the sense that the stopping criterion is evaluated after every newly drawn sample. Stopping rules are distinguished further by the confidence interval computation method. One stopping rule is based on assuming data drawn from normal distribution and the other assumes data drawn from lognormal distribution. The reason to use these two parametric methods is that confidence interval computations for normal and lognormal distributions only require the sample mean and sample variance which can be updated efficiently when one data

point is added. The fast mean and variance update can be found in [**Wel62**]:

$$\bar{X}_{k+1} = \frac{k\bar{X}_k + X_{k+1}}{k+1}$$

$$S^2_{k+1} = \frac{k-1}{k} S^2_k + \frac{(X_{k+1} - \bar{X}_k)^2}{k+1}.$$

We do not use non-parametric methods like bootstrap for confidence interval computation because they are relatively expensive and we want to allocate almost all computational resource to actual computations.

Note that there is one distinction between the confidence interval computation assuming lognormal distribution and distribution fitting assuming lognormal distribution. In the distribution fitting, which we described in Section 3.7.2, all three parameters are estimated, meaning the fitted distribution is actually $s + exp(\mathcal{N}(\mu, \sigma))$. In the confidence interval computation, samples are assumed to be drawn from $exp(\mathcal{N}(\mu, \sigma))$ with both μ and σ unknown. The author hasn't found any literature on confidence interval study of lognormal distribution with shifted location, so the assumed lognormal distribution in this subsection is $exp(\mathcal{N}(\mu, \sigma))$. It is possible to estimate the location parameter first, however, the approach is in general not computationally cheap. For example, the maximum likelihood estimation does not have an analytical solution. To use a purely sequential analysis, the stopping criterion needs to be relatively easy to check, so we only study the two distributions whose confidence interval procedure only requires sample mean and sample variance which can be efficiently updated.

Our stopping rule works as follows: in the pilot phase, collect 10 samples, after that each time collect one sample until the relative precision of the computed confidence interval with confidence level 0.95 is less than 0.01. Note that to evaluate this proposed stopping rule, we do not rerun the experiment to gather samples, instead we use the bootstrap technique to randomly resample from the data collected in the experiment.

We use a simulation to test the robustness and efficiency of the two stopping rules. In every independent run, we calculate a confidence interval when the collection of data is terminated. Among all independent runs (we use 1000 independent runs), we calculate the percentage of the computed confidence interval containing the true mean, and we also compute the average final sample size. Since it is impossible to know the true mean, we estimate the true mean as the average of all randomly resampled data in all independent runs.

In our simulation, we also do not consider very easy cases where the approximate run time is less than 1 second. The reason not to include those cases is that in our benchmark experiment we do not need high precision for short running times. Using relative precision can also lead to large final sample size if the true mean is very small. There are 994 cases in the simulation based on our policy where there are 30 measurements and every measurement is larger than 1 second. Using the stopping rule based on normal distribution has average coverage 91.07% and average final sample size 33.69, while using the stopping rule based on lognormal distribution has average coverage 91.02% and average final sample size 29.52.

From the simulation results, we observe that two stopping rules have almost the same and robust coverage, which is close to the nominal coverage 95%. In terms of efficiency, the stopping rule assuming lognormal distribution has smaller average sample size, meaning that the data collection will terminate early therefore save computational resource. If other researchers want to use some stopping rule to improve our benchmark, we recommend to use the stopping rule based on lognormal distribution.

3.7.4. CPU time vs wall clock time. One interesting topic about execution time is the difference between CPU time and wall clock time. Most research about computational performance is based on CPU time since it has less overhead introduced. In our experiment, we measure both CPU time and wall clock time. The difference of the absolute values between these two time measurements are hard to measure. In the computation on the HPC, it is possible that the actual wall clock time is shorter than the CPU time due to multiprocessing and resource allocation. Instead, we compare the variation between these two time measurements.

We consider all 5616 computations with 30 measurements in our experiment. We find that about 66% cases the variance of wall clock time is greater than that of CPU time. The maximum range of wall clock time is 105s while the maximum of CPU time is 51s. To make more statistically rigorous results, we use Levene's test and Bartlett's test to determine whether CPU time and wall clock time have equal variance. We use the implementations of these tests in the Scipy package, `scipy.stats.levene` and `scipy.stats.bartlett`. Out of 5616 cases, null hypothesis is rejected in 2% cases in Levene's test and is rejected in 4% cases in Bartlett's test. The result indicates that there is not enough evidence to claim that wall clock time has larger variance than CPU time. We include histograms of p-values in the Levene's test and Bartlett's test in Figure 3.3.

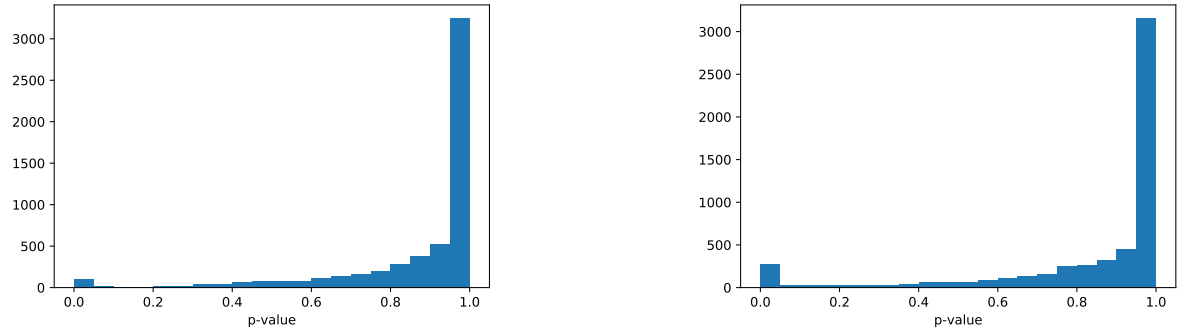

FIGURE 3.3. The left figure shows the p-values in Levene's test and the right figure shows the p-values in Bartlett's test.

3.8. Future work

Our benchmark experiment as well as the Github repository is only the first generation of work which is motivated by reproducibility. There are several different approaches to improve the framework. First, given the results in the current experiment, it is possible to use a batching technique to batch several computations in the same node. Doing so can help to better use the acquired computation power and therefore improve the experiment efficiency. Instead of using a fixed-size sample in the current experiment, sequential stopping rules can improve the efficiency further. However, one needs to be careful about the time limit on the node, and it may break the experiment with batching strategy. It is possible that in one node, data collection of one computation takes too long to finish and there is no time for the remaining computations which are batched to the same node. Another possible improvement is also related to the computation power. In a HPC, it is mandatory to specify a time limit. With longer time limit more data will certainly be collected, therefore more statistical properties can be studied. For example, the distribution fitting will be more robust, hypothesis testing in general will have large statistical power, and distribution of long execution times will be available to study. Tradeoffs between computational cost and statistically rigorous conclusion always need to be addressed.

Our distribution fitting procedure is intuitive in the sense that an existing robust python package is used and all parameters are estimated. We observed that some parameter estimation is not what we expect. In the distribution fitting with the lognormal distribution in Section 3.7.2, we expect that the estimated distribution is supported on a subset of $(0, +\infty)$, since the execution time can never be negative. But we found that in some cases the estimated location parameter is negative,

contradicting to our expectation. More careful studies can be done regarding how to best estimate parameters of interest. In the confidence interval computation with lognormal distribution, the formula in the literature is based on fixed location parameter 0. An open question can be formulated as follows. Given samples drawn from the distribution $s+exp(\mathcal{N}(\mu,\sigma^2))$ with unknown s,μ,σ, how to calculate the confidence interval of population mean $s+exp(\mu+\sigma^2/2)$.

CHAPTER 4

Dual-Feasible Functions

Dual-feasible functions have been used in several combinatorial optimization problems including knapsack type inequalities and proved to generate lower bounds efficiently. DFFs are in the scope of superadditive duality theory, and superadditive and nondecreasing DFFs can provide valid inequalities for general integer linear programs. Lueker [**Lue83**] studied the bin-packing problems and used certain DFFs to obtain lower bounds for the first time. Vanderbeck [**Van00**] proposed an exact algorithm for the cutting stock problems which includes adding valid inequalities generated by DFFs. Rietz et al. [**RACC12**] recently introduced a variant of this theory, in which the domain of DFFs is extended to all real numbers. Rietz et al. [**RAdCC14**] studied the maximality under the name of "general dual-feasible functions". They also summarized recent literature on DFFs in the monograph [**ACdCR16**].

4.1. Key results from the DFF literature

We summarize the basic definitions and key results from the monograph [**ACdCR16**].

Definition 4.1.1. *A function $\phi\colon [0,1] \to [0,1]$ is called a (valid)* classical dual-feasible function (cDFF)*, if for any finite list of real numbers $x_i \in [0,1]$, $i \in I$, it holds that $\sum_{i\in I} x_i \leq 1 \Rightarrow \sum_{i\in I} \phi(x_i) \leq 1$. A function $\phi\colon \mathbb{R} \to \mathbb{R}$ is called a (valid)* general dual-feasible function (gDFF)*, if for any finite list of real numbers $x_i \in \mathbb{R}$, $i \in I$, it holds that $\sum_{i\in I} x_i \leq 1 \Rightarrow \sum_{i\in I} \phi(x_i) \leq 1$.*

Definition 4.1.2. *A cDFF/gDFF is* maximal *if it is not (pointwise) dominated by a distinct cDFF/gDFF. A cDFF/gDFF is* extreme *if it cannot be written as a convex combination of other two different cDFFs/gDFFs.*

In the monograph [**ACdCR16**], the authors explored maximality of both cDFFs and gDFFs.

Theorem 4.1.1 (Characterization of maximal cDFFs, [**ACdCR16**, Theorem 2.1]). *A function $\phi\colon [0,1] \to [0,1]$ is a maximal cDFF if and only if $\phi(0) = 0$, ϕ is superadditive and ϕ is symmetric in the sense $\phi(x) + \phi(1-x) = 1$ for all $x \in [0,1]$.*

THEOREM 4.1.2 (Conditions for maximality of gDFFs, [**ACdCR16**, Theorem 3.1]). *Let $\phi\colon \mathbb{R} \to \mathbb{R}$ be a given function. If ϕ satisfies the following conditions, then ϕ is a maximal gDFF: (i) $\phi(0) = 0$. (ii) ϕ is symmetric in the sense $\phi(x)+\phi(1-x) = 1$ for all $x \in \mathbb{R}$. (iii) ϕ is superadditive. (iv) There exists an $\epsilon > 0$ such that $\phi(x) \geq 0$ for all $x \in (0, \epsilon)$.*

If ϕ is a maximal gDFF, then ϕ satisfies conditions (i), (iii) and (iv).

REMARK 4.1.1. *The function $\phi(x) = cx$ for $0 \leq c < 1$ is a maximal gDFF but it does not satisfy condition (ii).*

REMARK 4.1.2. *Note that conditions (i), (iii) and (iv) guarantee that any maximal gDFF is nondecreasing and consequently nonnegative on $\mathbb{R}_+$.*

Different approaches to construct non-trivial cDFFs from "simple" functions are explained in [**ACdCR16**], including convex combination and function composition.

PROPOSITION 4.1.1 ([**ACdCR16**, Section 2.3.1]). *If ϕ_1 and ϕ_2 are two maximal cDFFs, then $\alpha\phi_1 + (1-\alpha)\phi_2$ is also a maximal cDFF, for $0 < \alpha < 1$.*

PROPOSITION 4.1.2 ([**ACdCR16**, Proposition 2.3]). *If ϕ_1 and ϕ_2 are two maximal cDFFs, then the composed function $\phi_1(\phi_2(x))$ is also a maximal cDFF.*

Maximal gDFFs can also be obtained by extending maximal cDFFs to the domain $\mathbb{R}$. Theorem 4.1.3 uses quasiperiodic extensions and Theorem 4.1.4 uses affine functions when x is not in $[0, 1]$. Throughout the chapter, we use $\{a\}$ to represent the fractional part of a.

THEOREM 4.1.3 ([**ACdCR16**, Proposition 3.10]). *Let ϕ be a maximal cDFF, then there exists $b_0 \geq 1$ such that for all $b > b_0$ the following function $\hat{\phi}(x)$ is a maximal gDFF.*

$$\hat{\phi}(x) = \begin{cases} b \times \lfloor x \rfloor + \phi(\{x\}) & \text{if } x \leq 1 \\ 1 - \hat{\phi}(1-x) & \text{if } x > 1 \end{cases}.$$

THEOREM 4.1.4 ([**ACdCR16**, Proposition 3.12]). *Let ϕ be a maximal cDFF, then there exists $b \geq 1$ such that the following function $\hat{\phi}(x)$ is a maximal gDFF.*

$$\hat{\phi}(x) = \begin{cases} bx + 1 - b & \text{if } x < 0 \\ bx & \text{if } x > 1 \\ \phi(x) & \text{if } 0 \leq x \leq 1 \end{cases}.$$

Proposition 4.1.3 shows that every maximal gDFF is the sum of a linear function and a bounded function. Proposition 4.1.4 explains the behavior of nonlinear maximal gDFFs at given points.

PROPOSITION 4.1.3 ([**ACdCR16**, Proposition 3.4]). *If $\phi\colon \mathbb{R} \to \mathbb{R}$ is a maximal gDFF and $t = \sup\{\frac{\phi(x)}{x} : x > 0\}$, then we have $\lim_{x\to\infty} \frac{\phi(x)}{x} = t \leq -\phi(-1)$, and for any $x \in \mathbb{R}$, it holds that: $tx - \max\{0, t-1\} \leq \phi(x) \leq tx$.*

PROPOSITION 4.1.4 ([**ACdCR16**, Proposition 3.5]). *If $\phi\colon \mathbb{R} \to \mathbb{R}$ is a maximal gDFF and not of the kind $\phi(x) = cx$ for $0 \leq c < 1$, then $\phi(1) = 1$ and $\phi(\frac{1}{2}) = \frac{1}{2}$.*

The following proposition utilizes the fact that maximal gDFFs are superadditive and nondecreasing, which can be used to generate valid inequalities for general linear integer optimization problems.

PROPOSITION 4.1.5 ([**ACdCR16**, Proposition 5.1]). *If ϕ is a maximal gDFF and $L = \{x \in \mathbb{Z}^n_+ : \sum_{j=1}^n a_{ij}x_j \leq b_i, i = 1, 2, \ldots, m\}$, then for any i, $\sum_{j=1}^n \phi(a_{ij})x_j \leq \phi(b_i)$ is a valid inequality for L.*

4.2. Automatic tests and search for classical DFFs

In this section, we restrict ourselves to piecewise linear cDFFs. We introduce the automatic maximality and extremality tests of given piecewise linear functions, and a computer-based search method which is used to find new extreme functions. Our methods are released as part of the software [**KZHW20**] with `cutgeneratingfunctionology.dff` module.

4.2.1. Piecewise linear functions and polyhedral complexes underlying the algorithmic maximality test of classical DFFs. The definition of piecewise linear functions in the

cDFF setting is very similar to that of Gomory–Johnson cut-generating functions, and the major difference is that cDFFs has bounded domain while cut-generating functions are defined in $\mathbb{R}$. Therefore, we omit the formal definition and just emphasize on the difference.

Let $0 = a_0 < a_1 < \cdots < a_{n-1} < a_n = 1$. Denote by $B = \{a_0, a_1, \ldots, a_{n-1}, a_n\}$ the set of all possible *breakpoints*. Based on the set of breakpoints, The one-dimensional polyhedral complex $\mathcal{P} = \mathcal{P}_B$ can be defined. Similar to [**BHK14**,**BHK16a**,**BHK16b**,**BHKM13**,**HKZ18a**], we introduce the function $\nabla\phi\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, $\nabla\phi(x, y) = \phi(x + y) - \phi(x) - \phi(y)$. The function $\nabla\phi$ measures the slack in the superadditivity condition. Similarly, a two-dimensional polyhedral complex $\Delta\mathcal{P} = \Delta\mathcal{P}_B$ can be defined. Every face F of the complex are defined as follows. Let $I, J, K \in \mathcal{P}_B$, so each of I, J, K is either a breakpoint of ϕ or a closed interval delimited by two consecutive breakpoints. Then $F = F(I, J, K) = \{\, (x, y) \in \mathbb{R} \times \mathbb{R} : x \in I,\, y \in J,\, x + y \in K \,\}$. Note that piecewise linearity of function ϕ also implies piecewise linearity of $\nabla\phi$.

Unlike Gomory–Johnson cut-generating functions, which may be discontinuous at 0 on both sides, a maximal cDFF is always continuous at 0 from the right and at 1 from the left.

LEMMA 4.2.1. *Any piecewise linear maximal cDFF is continuous at* 0 *from the right and continuous at* 1 *from the left.*

PROOF. Consider ϕ to be a piecewise linear maximal cDFF, and $\phi(x) = sx + b$ on the first open interval (a_0, a_1). Note that the maximality of ϕ implies that $\phi(0) = 0$. Choose $x = y = \frac{a_1}{3}$, and based on superadditivity, we have

$$\phi(x) + \phi(y) \leq \phi(x + y) \Rightarrow sx + b + sy + b \leq s(x + y) + b \Rightarrow b \leq 0.$$

Since b is also the right limit at 0, so b is nonnegative. Therefore, $b = 0$, which implies ϕ is continuous at 0 from the right. By the symmetry condition, ϕ is continuous at 1 from the left. □

4.2.2. Maximality test. We introduce an efficient method to check the maximality of a given piecewise linear function. The code `maximality_test(`ϕ`)` implements a fully automatic test whether a given function ϕ is maximal, by using the information that is described in $\Delta\mathcal{P}$.

Based on Theorem 4.1.1, we need to first check that the range of the function stays in $[0, 1]$ and $\phi(0) = 0$. Since we assume the function is piecewise linear with finitely many breakpoints, only function values and left and right limits at the breakpoints need to be checked. Similarly, the symmetry condition only needs to be checked on all breakpoints including the left and right limits

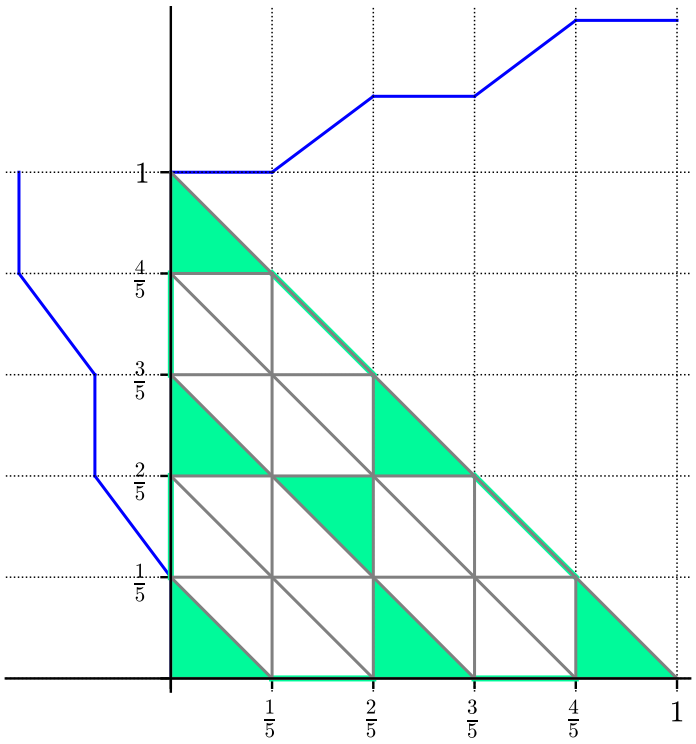

FIGURE 4.1. Maximal cDFF $\phi_{BJ,1}(x;C) = \frac{\lfloor Cx \rfloor + \max(0, \frac{\{Cx\}-\{C\}}{1-\{C\}})}{\lfloor C \rfloor}$ for $C = \frac{5}{2}$.

at each breakpoint. In regards to the superadditivity, it suffices to check $\nabla\phi(u,v) \geq 0$ for any $(u,v) \in \mathrm{vert}(F)$, including the limit values $\nabla\phi_F(u,v)$ when ϕ is discontinuous.

As for the diagrams of $\Delta\mathcal{P}$, we start with a triangle complex $I = J = K = [0,1]$, and then refine I, J, K based on the set of breakpoints, namely B. In practice, the code `plot_2d_diagram_dff(`ϕ`)` will show vertices where superadditivity or symmetry condition is violated (marked red). It also paints additive faces green, including 1-dimensional and 0-dimensional additive faces, which are additive edges and vertices not contained in any higher dimensional additive faces.

Figure 4.1 is an example of the $\Delta\mathcal{P}$ of a maximal cDFF. We also plot the function on the upper and left borders. There is no vertex where superadditivity or symmetry condition is violated, so the function is maximal.

4.2.3. Extremality test. Our automatic extremality test, `extremality_test_dff`, builds upon the techniques of the grid-free extremality test for the Gomory–Johnson setting, which is described in [**Zho17**, Chapter 4] and [**HKZ18c**, **HKZ18a**] and the forthcoming paper [**HKZ18b**], and implemented in [**KZHW20**].

In this subsection, we provide the technical results that allow us to adapt these techniques to cDFFs. First, there is a simple necessary condition for piecewise linear extreme cDFFs.

LEMMA 4.2.2. *Let ϕ be a piecewise linear extreme cDFF. If ϕ is strictly increasing, then $\phi(x) = x$. In other words, there is no strictly increasing piecewise linear extreme cDFF except for $\phi(x) = x$.*

PROOF. We know ϕ is continuous at 0 from the right. Suppose $\phi(x) = sx$, $x \in [0, a_1)$ and $s > 0$, since ϕ is not strictly increasing if $s = 0$. We claim that s is the smallest slope value of ϕ. Suppose otherwise $\phi(x) = s'x + t$, $x \in [r, r + \epsilon]$ with $s' < s$ and $\epsilon < a_1$. In order to satisfy the superadditivity, we have $\phi(r + \epsilon) \geq \phi(\epsilon) + \phi(r)$, which can be reduced to $s' \geq s$. The contradiction indicates that s is the smallest slope value. We have $s \leq 1$ since $\phi(1) = 1$. Similarly if $s = 1$, then $\phi(x) = x$.

Next, we can assume $0 < s < 1$. Define a function:

$$\phi_1(x) = \frac{\phi(x) - sx}{1 - s}.$$

It is not hard to show $\phi_1(x) = 0$ for $x \in [0, a_1)$, and $\phi_1(1) = 1$. The function ϕ_1 is superadditive because it is obtained by subtracting a linear function from a superadditive function. These two together guarantee that ϕ_1 stays in the range $[0, 1]$. The function ϕ_1 satisfies the symmetry condition due to the following equation:

$$\phi_1(x) + \phi_1(1 - x) = \frac{\phi(x) + \phi(1 - x) - sx - s(1 - x)}{1 - s} = 1.$$

Therefore, ϕ_1 is also a maximal cDFF. Moreover, $\phi(x) = sx + (1 - s)\phi_1(x)$ implies ϕ is not extreme, since it can be expressed as a convex combination of two different maximal cDFFs: x and ϕ_1. □

Next we give the definition of effective perturbation functions.

DEFINITION 4.2.1. *Let ϕ be a maximal cDFF. Then a function $\tilde{\phi}\colon [0, 1] \to \mathbb{R}$ is called an* effective perturbation function *of ϕ, if there exists $\epsilon > 0$, such that $\phi + \epsilon\tilde{\phi}$ and $\phi - \epsilon\tilde{\phi}$ are both maximal cDFFs.*

From the definition above, the zero function is always an effective perturbation function, and we call it the trivial effective perturbation function. There exists a nontrivial effective perturbation function of ϕ if and only if ϕ is not extreme.

Effective perturbations of a DFF ϕ have a close relation to the functions ϕ in regards to continuity and additivity.

LEMMA 4.2.3. *Let ϕ be a piecewise linear maximal cDFF. If ϕ is continuous on a proper interval $I \subseteq [0,1]$, then for any perturbation function $\tilde{\phi}$, we have that $\tilde{\phi}$ is Lipschitz continuous on the interval I. Furthermore, $\tilde{\phi}$ is continuous at all points at which ϕ is continuous.*

PROOF. We know ϕ is continuous at 0 from the right. Let $\tilde{\phi}$ be an effective perturbation function. Since ϕ is piecewise linear, there exists a nonnegative s, such that $\phi(x) = sx$ on the first interval $[0, a_1)$. Let $I = J = K = [0, a_1]$, and let $F = F(I, J, K)$. Then for any $x \in I$, $y \in J$, $x + y \in K$, $\nabla\phi_F(x, y) = s(x + y) - sx - sy = 0$. Thus, F is a two-dimensional additive face of ΔP. From the Interval Lemma, we know that there exists $\tilde{s}$, such that $\tilde{\phi}(x) = \tilde{s}x$, when $x \in [0, a_1)$. Since $\tilde{\phi}$ is an effective perturbation function, there exists $\epsilon > 0$, such that $\phi^+ = \phi + \epsilon\tilde{\phi}$ and $\phi^- = \phi - \epsilon\tilde{\phi}$ are both maximal cDFFs. We know that ϕ^+ and ϕ^- have slope $s^+ = s+\epsilon\tilde{s} \geq 0$ and $s^- = s-\epsilon\tilde{s} \geq 0$ respectively.

Let $I \subseteq [0, 1]$ be a proper interval where ϕ is continuous. Since ϕ is piecewise linear, there exists a positive constant C such that $|\phi(x) - \phi(y)| \leq C|x - y|$, for any $x, y \in I$. We can simply choose C to be the largest absolute value of the slopes of ϕ. Assume $x \geq y$ and $x - y < a_1$, from the superadditivity of ϕ^+ and ϕ^-, $\phi^+(x) \geq \phi^+(y) + \phi^+(x - y) = \phi^+(y) + s^+(x - y)$ and $\phi^-(x) \geq \phi^-(y)+\phi^-(x-y) = \phi^-(y)+s^-(x-y)$. It follows that $-(C+s^-)(x-y) \leq \epsilon(\tilde{\phi}(x)-\tilde{\phi}(y)) \leq (C + s^+)(x - y)$. Therefore, $|\tilde{\phi}(x) - \tilde{\phi}(y)| \leq \tilde{C}|x - y|$, where $\tilde{C} = \frac{1}{\epsilon}\max(C + s^-, C + s^+)$. Hence, $\tilde{\phi}$ is Lipschitz continuous on the interval I. □

We remark that, in contrast to the Gomory–Johnson setting, Lemma 4.2.3 holds without further hypotheses, and so the subtle issues regarding two-sided discontinuous functions explored in [**KZ18**] do not arise for our cDFFs.

For the following lemma, recall from Section 4.2.1 the notation $\nabla\tilde{\phi}_F(x, y)$ to denote the limit within the face F of the two-dimensional complex.

LEMMA 4.2.4. *Let ϕ be a piecewise linear maximal cDFF. For any effective perturbation function $\tilde{\phi}$, we have that $\tilde{\phi}$ satisfies additivity where ϕ satisfies additivity. This also holds true in the limit: If $F \in \Delta\mathcal{P}$, $(x, y) \in F$, and $\nabla\phi_F(x, y) = 0$, then $\nabla\tilde{\phi}_F(x, y) = 0$.*

PROOF. Since $\tilde{\phi}$ is an effective perturbation function, there exists $\epsilon > 0$, such that $\phi^+ = \phi + \epsilon\tilde{\phi}$ and $\phi^- = \phi - \epsilon\tilde{\phi}$ are both maximal cDFFs. If ϕ satisfies additivity at (x, y), we have $\phi(x) + \phi(y) = \phi(x+y)$. Applying superadditivity of ϕ^+ and ϕ^- at (x, y), we get $\tilde{\phi}(x)+\tilde{\phi}(y) = \tilde{\phi}(x+y)$. Likewise, if the limit $\nabla\phi_F(x, y)$ is zero, then the superadditivity of ϕ^+ and ϕ^- implies that the limit $\nabla\phi_F(x, y)$ exists and is zero. □

From the continuity (Lemma 4.2.3) and additivity (Lemma 4.2.4), our algorithm deduces further properties of every effective perturbation function $\tilde{\phi}$. One tool is the famous Gomory–Johnson Interval Lemma; we include a version of it below.

LEMMA 4.2.5 (Interval Lemma). *[**BHK16a**, Lemma 4.1] Let $a_1 < a_2$ and $b_1 < b_2$. Consider the intervals $A = [a_1, a_2]$, $B = [b_1, b_2]$, and $A + B = [a_1 + b_1, a_2 + b_2]$. Let $f\colon A \to \mathbb{R}$, $g\colon B \to \mathbb{R}$, and $h\colon A+B \to \mathbb{R}$ be bounded functions on A, B and $A+B$, respectively. If $f(a)+g(b) = h(a+b)$ for all $a \in A$ and $b \in B$, then f, g, and h are affine functions with identical slopes in the intervals A, B, and $A + B$, respectively.*

Using this lemma and additional techniques from [**Zho17**, Chapter 4], [**HKZ18a**], our algorithm constructs a list of pairwise disjoint *connected covered components* $C_1, \ldots, C_k$, each of which is a finite union of open intervals with the following property. If $\tilde{\pi}$ is any effective perturbation function and C_i is one of the connected covered components, then the restrictions of $\tilde{\pi}$ to the intervals of C_i are affine functions with identical slopes. Like in the Gomory–Johnson case, we can prove the finiteness of this construction when the breakpoints of ϕ are rational.

If there is some *uncovered interval*, i.e., an open interval of $[0, 1] \setminus \bigcup_i C_i$, then a nontrivial "equivariant" perturbation is guaranteed to exist, and hence `extremality_test_dff` returns `False`. (Our algorithm can actually construct such a perturbation using the method of inverse semigroups of restricted translations and reflections from [**Zho17**, Chapter 4], [**HKZ18a**]; we suppress all details.)

On the other hand, if the domain $[0, 1]$ is covered by the closures of $C_1, \ldots, C_k$, then any effective perturbation function is guaranteed to be piecewise linear. The existence of a nontrivial effective perturbation function depends on whether a finite dimensional linear system has a nontrivial solution.

We start with the continuous case. Suppose ϕ is a continuous piecewise linear maximal cDFF, thus any effective perturbation function $\tilde{\phi}$ must be continuous. If the domain $[0, 1]$ is covered by the

closures of $C_1, \ldots, C_k$, then $\tilde{\phi}$ has the same slope value on each C_i and we denote the slope value by s_i. Note that the effective perturbation function $\tilde{\phi}$ is uniquely determined by the set of slope values $\{s_1, \ldots, s_k\}$. Specifically, there exists a vector-valued linear function $g\colon [0,1] \to \mathbb{R}^k$ so that $\tilde{\phi}(x) = g(x) \cdot (s_1, \ldots, s_k)$. The ith coordinate of $g(x)$ represents the total length of the connected component C_i contained in the interval $[0, x]$, i.e. $g(x) = (|C_1 \cap [0,x]|, \ldots, |C_k \cap [0,x]|)$.

In the general case where ϕ may be discontinuous, the effective perturbation function $\tilde{\phi}$ may also be discontinuous. If $\tilde{\phi}$ may be discontinuous at x, then there exist jumps $h^- = \tilde{\phi}(x) - \tilde{\phi}(x^-)$ and $h^+ = \tilde{\phi}(x^+) - \tilde{\phi}(x)$, where h^-, h^+ represent the left and right discontinuity at x respectively. Note that h^-, h^+ could also be 0 representing (left/right) continuity at point x. It is not hard to see there are only finitely many points where discontinuity may occur, since discontinuity can only occur at the endpoints of connected components $C_1, \ldots, C_k$. The effective perturbation function $\tilde{\phi}$ is uniquely determined by the set of slope values $\{s_1, \ldots, s_k\}$ and potential jumps $\{h_1, \ldots, h_m\}$. The general form of an effective perturbation function $\tilde{\phi}$ can then be expressed using a vector-valued linear function $g\colon [0,1] \to \mathbb{R}^{k+m}$, slope variables and jump variables so that

$$\tilde{\phi}(x) = g(x) \cdot (s_1, \ldots, s_k, h_1, \ldots, h_m). \tag{4.1}$$

The last m coordinates of $g(x)$ are binaries indicating whether those potential jumps are contained in the interval $[0, x]$. Observe that the function g is determined only by the original function ϕ.

The next step is to find all constraints that $\tilde{\phi}(x)$ needs to satisfy and solve a linear system of $(s_1, \ldots, s_k, h_1, \ldots, h_m)$. If there is only the trivial solution, then `extremality_test_dff` returns `True`. If one nontrivial function $\tilde{\phi}(x)$ is found, then `extremality_test_dff` returns `False`. We use the following proposition. Recall from Section 4.2.1 the notation $\tilde{\phi}(a^-)$ to represent the left limit to a.

PROPOSITION 4.2.1. *Let ϕ be a piecewise linear maximal cDFF. Assume $\phi(a_1^-) = 0$ where a_1 is the breakpoint of ϕ next to 0, and assume ϕ has no uncovered interval. Let $\hat{B} = B \cup \bigcup_i \partial C_i$ be the union of breakpoints of ϕ and endpoints of the intervals of covered components, and $\hat{\mathcal{P}}$ be the new complex based on $\hat{B}$. The functions $\tilde{\phi}\colon [0,1] \to \mathbb{R}$ defined by (4.1), using the slope variables and jump variables given by all covered components of ϕ, are piecewise linear over $\hat{\mathcal{P}}$. Construct a*

linear system of equations for $\tilde{\phi}$*:*

$$\nabla\tilde{\phi}_F(x,y) = 0 \qquad \text{for all } F \in \Delta\hat{P},\ (x,y) \in \operatorname{vert}(F) \text{ such that } \nabla\phi_F(x,y) = 0, \tag{4.2a}$$

$$\tilde{\phi}(1) = \tilde{\phi}(a_1^-) = 0. \tag{4.2b}$$

If there is only the trivial solution $\tilde{\phi}(x) = 0$*, then* ϕ *is extreme. If there is some nontrivial solution* $\tilde{\phi}$*, then there exists* $\epsilon > 0$ *such that* $\phi + \epsilon\tilde{\phi}$ *and* $\phi - \epsilon\tilde{\phi}$ *are both maximal; thus* $\tilde{\phi}$ *is a nontrivial effective perturbation and* ϕ *is not extreme.*

PROOF. Note that if $\tilde{\phi}$ is an effective perturbation function, then it must satisfy the linear system (4.2) because of Lemma 4.2.4. By assumption, $[0,1]$ is covered by the closures of $C_1, \ldots, C_k$, where each C_i is a connected covered component. We know that $\tilde{\phi}$ is affine linear on each C_i with the same slope. Observe that if the linear system has only the trivial solution, then the only effective perturbation function is the zero function, thus ϕ is extreme.

Suppose there is a nonzero solution $\tilde{\phi}$ to the linear system (4.2); it is by definition a piecewise linear function on $[0,1]$ with possible discontinuities at the breakpoints. Let

$$\delta = \min\{\, \nabla\phi_F(x,y) \quad : F \in \Delta\hat{\mathcal{P}},\ (x,y) \in \operatorname{vert}(F),\ \nabla\phi_F(x,y) > 0 \,\},$$
$$\sigma = \max\{\, |\nabla\tilde{\phi}_F(x,y)| : F \in \Delta\hat{\mathcal{P}},\ (x,y) \in \operatorname{vert}(F),\ \nabla\phi_F(x,y) > 0 \,\}.$$

Note the minimum and maximum are over a finite set. Choose $\epsilon = \frac{\delta}{\max(\sigma,1)} > 0$; then we claim that $\phi + \epsilon\tilde{\phi}$ and $\phi - \epsilon\tilde{\phi}$ are both superadditive. Let $F \in \Delta\hat{P}$ and $(x,y) \in \operatorname{vert}(F)$. We compute $\nabla(\phi + \epsilon\tilde{\phi})_F(x,y) = \nabla\phi_F(x,y) + \epsilon\nabla\tilde{\phi}_F(x,y)$. If $\nabla\phi_F(x,y) = 0$, then by (4.2), also $\nabla\tilde{\phi}_F(x,y) = 0$. Otherwise $\nabla\phi_F(x,y) > 0$, and then $\nabla\phi_F(x,y) + \epsilon\nabla\tilde{\phi}_F(x,y) \geq \delta - \epsilon\sigma \geq 0$. Thus, $\phi \pm \epsilon\tilde{\phi}$ are both superadditive.

Consider ϕ and $\tilde{\phi}$ on the interval $[0, a_1)$. The function ϕ is the zero function on $[0, a_1)$ since $\phi(0) = \phi(0^+) = \phi(a_1^-) = 0$. Note that $(0, a_1)$ belongs to some covered component, i.e., $(0, a_1) \subset C_i$ for some i. Then $\tilde{\phi}$ is also a linear function on $(0, a_1)$. Due to $\tilde{\phi}(0) = \tilde{\phi}(0^+) = \tilde{\phi}(a_1^-) = 0$, we know that $\tilde{\phi}$ is also the zero function on $[0, a_1)$. Then $\phi + \epsilon\tilde{\phi}$ and $\phi - \epsilon\tilde{\phi}$ are both nonnegative on $[0, a_1)$, so they are monotone increasing on $[0,1]$ by superadditivity. Since $(\phi \pm \epsilon\tilde{\phi})(1) = \phi(1) \pm \epsilon\tilde{\phi}(1) = 1$ and the functions are monotone increasing, $\phi \pm \epsilon\tilde{\phi}$ both stay in the range of $[0,1]$.

The symmetry condition of $\phi+\epsilon\tilde{\phi}$ and $\phi-\epsilon\tilde{\phi}$ is implied by the symmetry condition of ϕ. Indeed, for every face $F = F(I, J, K) \in \Delta\hat{P}$ such that $K = \{1\}$ and every vertex $(x, 1-x) \in \mathrm{vert}(F)$, we have $\nabla\phi_F(x, 1-x) = 0$. Then from the linear system (4.2), we also have $\nabla\tilde{\phi}_F(x, 1-x) = 0$, and thus $\nabla(\phi \pm \epsilon\tilde{\phi})_F(x, 1-x) = \nabla\phi_F(x, 1-x) \pm \epsilon\nabla\tilde{\phi}_F(x, 1-x) = 0$.

Therefore, both $\phi + \epsilon\tilde{\phi}$ and $\phi - \epsilon\tilde{\phi}$ are maximal cDFFs, thus ϕ is not extreme. □

4.2.4. Computer-based search. In this subsection, we discuss how computer-based search can help in finding extreme cDFFs. Most known cDFFs in the monograph [**ACdCR16**] have a similar structure: continuous cDFFs are 2-slope functions, and discontinuous cDFFs have slope 0 in every affine linear piece.

We transfer a computer-based search technique from [**KZ17**] for Gomory–Johnson functions to cDFFs. Our goal is to find piecewise linear extreme cDFFs with rational breakpoints, which have fixed common denominator $q \in \mathbb{N}$. The strategy is to discretize the interval $[0, 1]$ and consider discrete functions on $B_q := \frac{1}{q}\mathbb{Z} \cap [0, 1]$, or, equivalently, vectors in $\mathbb{R}^{q+1}$ whose components are the function values on the grid B_q. In this space, we define a polytope by inequalities from the characterization of maximality. Extreme points of the polytope can be found by vertex enumeration tools. Recent advances in polyhedral computation (Normaliz, version 3.2.0; see [**BIS16**]) allow us to reach $q = 31$ in under a minute of CPU time. Candidates for extreme cDFFs ϕ are obtained by interpolating values on $\frac{1}{q}\mathbb{Z} \cap [0, 1]$ from each extreme point (discrete function). Then we use our extremality test to filter out the non-extreme functions.

Based on a detailed computational study regarding the performance of vertex enumeration codes in [**KZ17**], we consider two libraries, the Parma Polyhedra Library (PPL) and Normaliz. Both are convenient to use within the software SageMath [**S$^+$16**].

We now introduce some notation, which will allow us to make precise statements that also include the discontinuous case.

Definition 4.2.2. *We use B_q to denote the set $\{0, \frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}, 1\}$. Denote $\Phi_C(q)$ to be the set of all maximal continuous piecewise linear cDFFs with breakpoints in B_q, and $\Phi_D(q)$ to be the set of all maximal possibly discontinuous piecewise linear cDFFs with breakpoints in B_q.*

Theorem 4.2.1. *Both $\Phi_C(q)$ and $\Phi_D(q)$ are linearly isomorphic to finite dimensional convex polytopes $\Phi'_C(q) \subset \mathbb{R}^{q+1}$ and $\Phi'_D(q) \subset \mathbb{R}^{3q-1}$, respectively, if q is fixed.*

PROOF. *Continuous case.* Note that any maximal cDFF $\phi \in \Phi_C(q)$ is uniquely determined by the values at the breakpoints. So we just need to consider discrete functions on B_q that are the restrictions $\phi|_{B_q}$ of ϕ to B_q. Since ϕ is maximal, $\phi|_{B_q}$ should also satisfy superadditivity and the symmetry condition.

For each possible breakpoint $\frac{i}{q}$, we introduce a variable v_i to be the value $\phi(\frac{i}{q})$. Considering inequalities from superadditivity, the symmetry condition and $0 \leq v_i \leq 1$, $v_0 = 0$, we get a polytope in $q+1$ dimensional space, because there are only finitely many inequalities and each variable is bounded. We denote this polytope by $\Phi'_C(q)$.

It is not hard to prove the convex combination of two maximal continuous piecewise linear cDFFs with breakpoints in B_q is also in $\Phi_C(q)$.

We can get ϕ back by interpolating $\phi|_{B_q}$. Therefore $\Phi_C(q)$ is linearly isomorphic to $\Phi'_C(q)$, a finite dimensional convex polytope.

Discontinuous case. Consider the linear map from $\Phi_D(q)$ to $\mathbb{R}^{3q-1}$ given by

$$\phi \mapsto \Big(\phi(0),\, \phi(\tfrac{1}{q}^-),\, \phi(\tfrac{1}{q}),\, \phi(\tfrac{1}{q}^+),\, \ldots,\, \phi(\tfrac{q-1}{q}^-),\, \phi(\tfrac{q-1}{q}),\, \phi(\tfrac{q-1}{q}^+),\, \phi(1)\Big),$$

where $\phi(a^-)$ and $\phi(a^+)$ again represent the left and right limits to a respectively. Denote the image by $\Phi'_D(q)$. This map is invertible by interpolating linearly between the given limit values of ϕ near the breakpoints. Moreover, we know from the maximality test (Section 4.2.2) that it suffices to test the limits $\nabla\phi_F(x,y)$ to the vertices of the complex $\Delta\mathcal{P}$ within faces $F \ni (x,y)$ of $\Delta\mathcal{P}$. Each vertex (x,y) is contained in at most 12 faces F. Each of the limits can be expressed as a linear combination of values and limits of ϕ at the breakpoints, such as $\phi(x^+) + \phi(y^-) - \phi((x+y)^-)$, $\phi(x^-) + \phi(y) - \phi((x+y)^-)$, etc. Therefore, $\Phi'_D(q)$ is a polytope. □

A function ϕ that we obtain from the interpolation of the discrete function values of a vertex of $\Phi'_C(q)$, or the interpolation of the function values and limits of a vertex of $\Phi'_D(q)$, is only a candidate of extreme functions. We need to use the extremality test described in Section 4.2.3 to pick those actual extreme cDFFs. The following theorem provides an easier verification for extremality: if ϕ has no uncovered interval, then we can claim we find an extreme cDFF.

THEOREM 4.2.2. *Let ϕ be a function from interpolating values of some extreme point of the polytope $\Phi'_C(q)$ or $\Phi'_D(q)$. Then ϕ is extreme if and only if there is no uncovered interval.*

PROOF. Here we only give the proof for continuous case, and the proof for discontinuous case is similar.

Suppose ϕ is obtained by in interpolating the discrete function $\phi|_{B_q}$, which is an extreme point of the polytope $\Phi'_C(q)$, and $\tilde{\phi}$ is an effective perturbation function.

If there is an uncovered interval, by Section 4.2.3, there exists an effective "equivariant" perturbation function, and the function is not extreme.

If there is no uncovered interval for ϕ, then the interval $[0,1]$ is covered by the closures of $C_1, \ldots, C_k$, where each C_i is a connected covered component. Since every breakpoint of ϕ is in the form of $\frac{i}{q}$, the endpoints of C_i are also in the form of $\frac{i}{q}$. We know ϕ and $\tilde{\phi}$ are affine linear on each C_i with the same slope by the Interval Lemma, and continuity of ϕ implies continuity of $\tilde{\phi}$. Therefore, we know $\tilde{\phi}$ is also a continuous function with breakpoints in B_q, which means $\phi+\epsilon\tilde{\phi}$ and $\phi+\epsilon\tilde{\phi}$ both have the same property. The maximality of $\phi+\epsilon\tilde{\phi}$ and $\phi+\epsilon\tilde{\phi}$ implies their restrictions to B_q are also in the polytope $\Phi'_C(q)$, and

$$\phi|_{B_q} = \frac{(\phi+\epsilon\tilde{\phi})|_{B_q} + (\phi-\epsilon\tilde{\phi})|_{B_q}}{2}.$$

Since $\phi|_{B_q}$ is an extreme point of the polytope $\Phi'_C(q)$, then $\phi|_{B_q} = (\phi+\epsilon\tilde{\phi})|_{B_q} = (\phi-\epsilon\tilde{\phi})|_{B_q}$, which implies $\phi = \phi+\epsilon\tilde{\phi} = \phi-\epsilon\tilde{\phi}$. Therefore, ϕ is extreme. □

Table 4.1 shows the results and the computation time for computing all vertices of $\Phi'_C(q)$ for different values of q. We then use Theorem 4.2.2 to filter out those non-extreme functions which have uncovered intervals. As we can see in the table, the actual extreme cDFFs are much fewer than the vertices of the polytope $\Phi'_C(q)$. PPL is faster when q is small and Normaliz performs well when q is relatively large. We can observe that the time cost increases dramatically as q gets large. Similar to [**KZ17**], we can apply the preprocessing program "redund" provided by lrslib (version 5.08), which removes redundant inequalities using Linear Programming. However, in contrast to the computation in [**KZ17**], removing redundancy from the system does not improve the efficiency. Instead, for relatively large q, the time cost after preprocessing is a little more than that of before preprocessing for both PPL and Normaliz.

For example, for $q = 31$, among 91761 functions interpolated from extreme points, there are 1208 extreme cDFFs, most of which do not belong to known families.

TABLE 4.1. Search for extreme cDFFs and efficiency of vertex enumeration codes (continuous case)

Polytope $\Phi_C(q)$						Running times (s)	
q	dim	inequalities		vertices	extreme DFF	PPL	Normaliz
		original	minimized				
2	0	4	3	1	1	0.00006	0.002
3	1	5	5	2	1	0.00009	0.006
5	2	9	7	3	2	0.00014	0.007
7	3	15	10	5	3	0.0002	0.007
9	4	23	14	9	3	0.0004	0.008
11	5	33	18	14	7	0.0006	0.010
13	6	45	23	25	8	0.001	0.012
15	7	59	29	66	14	0.003	0.018
17	8	75	35	94	22	0.005	0.025
19	9	93	42	221	32	0.010	0.042
21	10	113	50	677	55	0.036	0.105
23	11	135	58	1360	105	0.110	0.226
25	12	159	67	3898	189	0.526	0.725
27	13	185	77	12279	291	5.1	2.991
29	14	213	87	28877	626	41	9.285
31	15	243	98	91761	1208	595	35.461

4.3. Characterization of maximal general DFFs

Alves et al. [**ACdCR16**] provided several sufficient conditions and necessary conditions of maximal gDFFs in Theorem 4.1.2, but they do not match precisely. Inspired by the characterization of minimal cut-generating functions in the Yıldız–Cornuéjols model [**YC16**], we complete the characterization of maximal gDFFs.

PROPOSITION 4.3.1. *A function $\phi\colon \mathbb{R} \to \mathbb{R}$ is a maximal gDFF if and only if the following conditions hold:*

(i) $\phi(0) = 0$.

(ii) ϕ *is superadditive.*

(iii) $\phi(x) \geq 0$ *for all* $x \in \mathbb{R}_+$.

(iv) $\phi(x) = \inf_k\{\frac{1}{k}(1 - \phi(1 - kx)) : k \in \mathbb{Z}_{++}\}$ *for all* $x \in \mathbb{R}$.

PROOF. Suppose ϕ is a maximal gDFF, then conditions $(i), (ii), (iii)$ hold by Theorem 4.1.2, which also implies that ϕ is monotone increasing. For any $x \in \mathbb{R}$ and $k \in \mathbb{Z}_+$, $kx + (1 - kx) =$

$1 \Rightarrow k\phi(x) + \phi(1-kx) \leq 1$. So $\phi(x) \leq \frac{1}{k}(1-\phi(1-kx))$ for any positive integer k, then $\phi(x) \leq \inf_k\{\frac{1}{k}(1-\phi(1-kx)) : k \in \mathbb{Z}_+\}$.

If there exists x_0 such that $\phi(x_0) < \inf_k\{\frac{1}{k}(1-\phi(1-kx_0))\colon k \in \mathbb{Z}_+\}$, then define a function ϕ_1 which takes value $\inf_k\{\frac{1}{k}(1-\phi(1-kx_0))\colon k \in \mathbb{Z}_+\}$ at x_0 and $\phi(x)$ if $x \neq x_0$. We claim that ϕ_1 is a gDFF which dominates ϕ. Given a function $y\colon \mathbb{R} \to \mathbb{Z}_+$, with finite support satisfying $\sum_{x\in\mathbb{R}} x\, y(x) \leq 1$. We have $\sum_{x\in\mathbb{R}} \phi_1(x)\, y(x) = \phi_1(x_0)\, y(x_0) + \sum_{x\neq x_0} \phi(x)\, y(x)$. If $y(x_0) = 0$, then it is clear that $\sum_{x\in\mathbb{R}} \phi_1(x)\, y(x) \leq 1$. Let $y(x_0) \in \mathbb{Z}_+$, then $\phi_1(x_0) \leq \frac{1}{y(x_0)}(1-\phi(1-y(x_0)\, x_0))$ by definition of ϕ_1, then $\phi_1(x_0)\, y(x_0) + \phi(1-y(x_0)\, x_0) \leq 1$. Since ϕ is superadditive and monotone increasing, we get $\sum_{x\neq x_0} \phi(x)\, y(x) \leq \phi(\sum_{x\neq x_0} x\, y(x)) \leq \phi(1-y(x_0)\, x_0)$. From the two inequalities we conclude that ϕ_1 is a gDFF and dominates ϕ, which contradicts the maximality of ϕ. So the condition (iv) holds.

Suppose there is a function $\phi\colon \mathbb{R} \to \mathbb{R}$ satisfying all four conditions. Choose $x = 1$ and $k = 1$, we can get $\phi(1) \leq 1$ from (iv). Together with conditions $(i), (ii), (iii)$, it guarantees that ϕ is a gDFF by the definition of gDFFs. Assume that there is a gDFF ϕ_1 dominating ϕ and there exists x_0 such that $\phi_1(x_0) > \phi(x_0) = \inf_k\{\frac{1}{k}(1-\phi(1-kx_0))\colon k \in \mathbb{Z}_+\}$. So there exists some $k \in \mathbb{Z}_+$ such that

$$\begin{aligned} &\phi_1(x_0) > \frac{1}{k}(1-\phi(1-kx_0)) \\ \Leftrightarrow\ &k\phi_1(x_0) + \phi(1-kx_0) > 1 \\ \Rightarrow\ &k\phi_1(x_0) + \phi_1(1-kx_0) > 1. \end{aligned}$$

The last step contradicts the fact that ϕ_1 is a gDFF. Therefore, ϕ is maximal. □

Parallel to the restricted minimal and strongly minimal functions in the Yıldız–Cornuéjols model [**YC16**], "restricted maximal" and "strongly maximal" gDFFs are defined by strengthening the notion of maximality.

Definition 4.3.1. *We say that a gDFF ϕ is* implied via scaling *by a gDFF ϕ_1, if $\beta\phi_1 \geq \phi$ for some $0 \leq \beta \leq 1$. We call a gDFF $\phi\colon \mathbb{R} \to \mathbb{R}$* restricted maximal *if ϕ is not implied via scaling by a distinct gDFF ϕ_1. We say that a gDFF ϕ is* implied *by a gDFF ϕ_1, if $\phi(x) \leq \beta\phi_1(x) + \alpha x$ for some $0 \leq \alpha, \beta \leq 1$ and $\alpha + \beta \leq 1$. We call a gDFF $\phi\colon \mathbb{R} \to \mathbb{R}$* strongly maximal *if ϕ is not implied by a distinct gDFF ϕ_1.*

Note that restricted maximal gDFFs are maximal and strongly maximal gDFFs are restricted maximal. Based on the definition of strong maximality, $\phi(x) = x$ is implied by the zero function, so ϕ is not strongly maximal, though it is extreme. We include the characterizations of restricted maximal and strongly maximal gDFFs here, which only involve the standard symmetry condition instead of the generalized symmetry condition.

THEOREM 4.3.1. *A function $\phi\colon \mathbb{R} \to \mathbb{R}$ is a restricted maximal gDFF if and only if the following conditions hold:*

(i) $\phi(0) = 0$.
(ii) ϕ *is superadditive.*
(iii) $\phi(x) \geq 0$ *for all* $x \in \mathbb{R}_+$.
(iv) $\phi(x) + \phi(1 - x) = 1$ *for all* $x \in \mathbb{R}$.

PROOF. It is easy to show that ϕ is valid and restricted maximal if ϕ satisfies conditions $(i-iv)$. Suppose ϕ is a restricted maximal gDFF, then we only need to prove condition (iv), since restricted maximality implies maximality.

Suppose there exists some x such that $\phi(x)+\phi(1-x) < 1$. By the characterization of maximality, $\phi(x) = \inf_k\{\frac{1}{k}(1 - \phi(1 - kx))\colon k \in \mathbb{Z}_+\}$.

Case 1: Suppose there exists some $k \in \mathbb{N}$ such that $\phi(x) = \frac{1}{k}(1-\phi(1-kx))$. By superadditivity $k\phi(x) = 1-\phi(1-kx) = 1-\phi(1-x-(k-1)x) \geq 1-\phi(1-x)+\phi((k-1)x) \geq 1-\phi(1-x)+(k-1)\,\phi(x)$, which implies $\phi(x) + \phi(1 - x) \geq 1$, in contradiction to the assumption above.

Case 2: Suppose otherwise $\phi(x) < \frac{1}{k}(1 - \phi(1 - kx))$ for any positive integer k. Therefore, for any $\epsilon > 0$, there exists a corresponding $k_\epsilon \in \mathbb{N}$, such that

$$\phi(x) < \frac{1}{k_\epsilon}(1 - \phi(1 - k_\epsilon x)) < \phi(x) + \epsilon.$$

Then $\phi(k_\epsilon x) \leq 1 - \phi(1 - k_\epsilon x) < k_\epsilon\phi(x) + k_\epsilon\epsilon$, or equivalently $\frac{\phi(k_\epsilon x)}{k_\epsilon} < \phi(x) + \epsilon$. Since ϕ is superadditive, $\phi(x) \leq \frac{\phi(k_\epsilon x)}{k_\epsilon}$. Let ϵ go to 0 in the inequality $\phi(x) \leq \frac{\phi(k_\epsilon x)}{k_\epsilon} < \phi(x) + \epsilon$, and we have $\lim_{\epsilon\to 0} \frac{\phi(k_\epsilon x)}{k_\epsilon} = \phi(x)$. It is easy to see that $\lim_{\epsilon\to 0} k_\epsilon = +\infty$.

Next, we will show that $\phi(kx) = k\phi(x)$ for any positive integer k. Suppose $\bar{k}$ is the smallest integer such that $\frac{\phi(\bar{k}x)}{\bar{k}} = \phi(x) + \delta$ for some $\delta > 0$. Then for any $i \geq \bar{k}$, there exist $\lambda_i, r_i \in \mathbb{Z}_+$, such

that $i = \lambda_i \bar{k} + r_i$, $0 \leq r_i < \bar{k}$. Then

$$\begin{aligned}\phi(ix) &= \phi(\lambda_i \bar{k} x + r_i x) \geq \lambda_i \phi(\bar{k}x) + \phi(r_i x) \\ &\geq \lambda_i \bar{k}\phi(x) + \lambda_i \bar{k}\delta + r_i \phi(x) = i\phi(x) + (i - r_i)\delta.\end{aligned}$$

Therefore $\frac{\phi(ix)}{i} \geq \phi(x) + \delta - \frac{r_i}{i}\delta$ for any $i \geq \bar{k}$. Since r_i is bounded, $\frac{\phi(ix)}{i} \geq \phi(x) + \frac{\delta}{2}$ for any $i \geq 2\bar{k}$, which contradicts $\lim_{\epsilon \to 0} \frac{\phi(k_\epsilon x)}{k_\epsilon} = \phi(x)$. We have $\phi(kx) = k\phi(x)$ for any positive integer k. From Proposition 4.1.4 we know $\phi(1) = 1$, and we have

$$\begin{aligned}k\phi(x) = \phi(kx) &\geq (k-1)\phi(1) + \phi(1 - k(1-x)) \\ &\Leftrightarrow 1 - \phi(x) \leq \frac{1 - \phi(1 - k(1-x))}{k} \\ &\Rightarrow 1 - \phi(x) \leq \inf_k \frac{1 - \phi(1 - k(1-x))}{k} = \phi(1-x).\end{aligned}$$

The above inequality contradicts our original assumption.

In both cases, we have a contradiction if $\phi(x) + \phi(1-x) < 1$. Therefore, $\phi(x) + \phi(1-x) = 1$, which completes the proof. □

THEOREM 4.3.2. *A function $\phi\colon \mathbb{R} \to \mathbb{R}$ is a strongly maximal gDFF if and only if ϕ is a restricted maximal gDFF and $\lim_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = 0$.*

PROOF. Suppose ϕ is strongly maximal, we only need to show $\lim_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = 0$ since strong maximality implies restricted maximality. We first show that $\liminf_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = 0$. It is clear that $\liminf_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} \geq 0$ since ϕ is restricted maximal. Assume $\liminf_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = s > 0$, then there exist $\delta > 0$ and $s' < s$ (small enough) such that $\phi(x) \geq s'x$ for $x \in [0, \delta]$. Define a new function $\phi_1(x) = \frac{\phi(x) - s'x}{1 - s'}$, and ϕ is implied by ϕ_1. Note that ϕ_1 is a restricted maximal gDFF. The strong maximality of ϕ implies $\phi_1(x) = \phi(x) = x$. Therefore, $\phi(x) = x$ is not strongly maximal. This contradiction implies that $\liminf_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = 0$.

Next we show that $\lim_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = 0$. Suppose on the contrary there exists some positive s such that $\limsup_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = 3s > 0$. There exist two positive and decreasing sequences $(x_n)_{n=1}^\infty$ and $(y_n)_{n=1}^\infty$ approaching 0, such that $\phi(x_n) > 2sx_n$ and $\phi(y_n) < sy_n$. Fix y_1 and choose $0 < x_n < y_1$ and $k \in \mathbb{Z}_{++}$ such that $y_1 \geq kx_n \geq \frac{y_1}{2}$. Since ϕ is superadditive and nondecreasing, $\phi(y_1) \geq \phi(kx_n) \geq k\phi(x_n) > 2ksx_n \geq sy_1$, which contradicts the choice of y_1. Then we have $\limsup_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = \liminf_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = 0$, and $\lim_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = 0$ for a strongly maximal gDFF ϕ.

On the other hand, we assume ϕ is restricted maximal and $\lim_{\epsilon\to 0^+} \frac{\phi(\epsilon)}{\epsilon} = 0$. Suppose ϕ is implied by a gDFF ϕ_1 meaning $\phi(x) \le \beta\phi_1(x) + \alpha x$ and $\beta, \alpha \ge 0, \beta + \alpha \le 1$. Let $x = 1$, then $1 \le \beta\phi_1(1) + \alpha \le \beta + \alpha \le 1$. We know that $\beta = 1 - \alpha$. Note that $\beta\phi_1(x) + \alpha x$ is also a gDFF (a convex combination of two gDFFs ϕ_1 and x), then $\phi(x) = (1-\alpha)\phi_1(x) + \alpha x$ due to the maximality of ϕ. Divide by x from the above equation and take the lim inf as $x \to 0^+$, we can conclude $\alpha = 0$. So ϕ is strongly maximal. □

REMARK 4.3.1. *Let ϕ be a maximal gDFF that is not linear. By Proposition 4.1.4 we know that $\phi(1) = 1$. If ϕ is implied via scaling by a gDFF ϕ_1, or equivalently $\beta\phi_1 \ge \phi$ for some $0 \le \beta \le 1$, then $\beta\phi_1(1) \ge \phi(1) = 1$. Since $\beta \le 1$ and $\phi_1(1) \le 1$, we have $\beta = 1$ and ϕ is dominated by ϕ_1. The maximality of ϕ implies $\phi = \phi_1$, so ϕ is restricted maximal. Therefore, we have a simpler version of the characterization of maximal gDFFs.*

THEOREM 4.3.3. *A function $\phi\colon \mathbb{R} \to \mathbb{R}$ is a maximal gDFF if and only if $\phi(x) = sx$ for some $0 \le s < 1$ or ϕ is a restricted maximal gDFF, i.e., ϕ satisfies the following conditions:*

(i) $\phi(0) = 0$.

(ii) ϕ *is superadditive.*

(iii) $\phi(x) \ge 0$ *for all* $x \in \mathbb{R}_+$.

(iv) $\phi(x) + \phi(1-x) = 1$ *or* $\phi(x) = sx$, $0 \le s < 1$.

We use Zorn's lemma to show that maximal, restricted maximal and strongly maximal gDFFs exist, and they are potentially stronger than just valid gDFFs. The proof is analogous to the proof of [**YC16**, Theorem 1, Proposition 6, Theorem 9].

THEOREM 4.3.4. *(i) Every gDFF is dominated by a maximal gDFF.*

(ii) Every gDFF is implied via scaling by a restricted maximal gDFF.

(iii) Every gDFF is implied by a strongly maximal gDFF.

PROOF. *Part (i).* If the gDFF ϕ is already maximal, then it is dominated by itself. We assume ϕ is not maximal. Define a set $A = \{\text{ valid gDFF } \hat{\phi} : \hat{\phi}(x) \ge \phi(x) \text{ for } x \in \mathbb{R}\}$. We consider $(A, \le)$ as a partially ordered set, where the partial order $\phi_1 \le \phi_2$ is imposed by the pointwise inequality $\phi_1(x) \le \phi_2(x)$ for all $x \in \mathbb{R}$. Consider a chain C and a function $\phi_C(x) = \sup_{\phi' \in C} \phi'(x)$. We claim ϕ_C is an upper bound of the chain C and it is contained in A.

First, we prove ϕ_C is a well-defined function. For any fixed $r_0 \in \mathbb{R}$, based on the definition of gDFF, we know that for any $\phi' \in C$ it holds that $\phi'(r_0) + \phi(-r_0) \leq \phi'(r_0) + \phi'(-r_0) \leq 0$. Note that $\phi(-r_0)$ is a fixed constant and it forces that $\sup_{\phi' \in C} \phi'(r_0) < \infty$. So we know that $\phi_C(x) = \sup_{\phi' \in C} \phi'(x) < \infty$ for any $x \in \mathbb{R}$.

Next, we prove ϕ_C is a valid gDFF and dominates ϕ. It is clear that $\phi_C \geq \phi$, so we only need to show ϕ_C is a valid gDFF. Suppose on the contrary ϕ_C is not valid, then there exist $(x_i)_{i=1}^m$ such that $\sum_{i=1}^m x_i \leq 1$ and $\sum_{i=1}^m \phi_C(x_i) = 1 + \epsilon$ for some $\epsilon > 0$. Since there are only finite number of x_i, we can choose a function $\phi' \in C$ such that $\phi_C(x_i) < \phi'(x_i) + \frac{\epsilon}{m}$ for $i = 1, 2, \ldots, m$. Then $1 + \epsilon = \sum_{i=1}^m \phi_C(x_i) < \sum_{i=1}^m (\phi'(x_i) + \frac{\epsilon}{m}) \leq 1 + \epsilon$. The last step is due to the fact that ϕ' is a valid gDFF. From the contradiction we know that ϕ_C is a valid gDFF.

We have shown that every chain in the set A has a upper bound in A. By Zorn's lemma, we know there is a maximal element in the set A, which is the desired maximal gDFF.

Part (ii). By (i) we only need to show every maximal gDFF ϕ is implied via scaling by a restricted maximal gDFF. Based on Theorem 4.3.3, ϕ is either restricted maximal or a linear function. If ϕ is restricted maximal, then it is implied via scaling by itself. If ϕ is a linear function, then it is implied via scaling by $\phi'(x) = x$.

Part (iii). Suppose ϕ is a linear function, and $\phi(x) = s_0 x$ where $0 \leq s_0 \leq 1$. Observe that ϕ is implied by any strongly maximal gDFF ϕ_1, since we have $s_0 x \leq 0 \times \phi_1(x) + s_0 x$.

Now we assume that ϕ is nonlinear. By (ii) we only need to show every restricted maximal gDFF ϕ is implied by a strongly maximal gDFF. If ϕ is already strongly maximal, then it is implied by itself. Suppose ϕ is not strongly maximal.

First, we claim that $\lim_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon}$ exists. The proof of the claim follows the proof of Theorem 4.3.2 so we omit it here. Since ϕ is not strongly maximal, $\lim_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} > 0$ by Theorem 4.3.2. If $\lim_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = 1$, then ϕ is the linear functions $\phi(x) = x$. We can assume that $1 > \lim_{\epsilon \to 0^+} \frac{\phi(\epsilon)}{\epsilon} = s > 0$. Define a new function $\phi_1(x) = \frac{\phi - sx}{1-s}$ and we want to show ϕ_1 is a strongly maximal gDFF. Note that $\phi_1(0) = 0$, ϕ_1 is superadditive, $\phi_1(x) + \phi_1(1-x) = 1$ and $\lim_{\epsilon \to 0^+} \frac{\phi_1(\epsilon)}{\epsilon} = 0$. We only need to prove $\phi_1(x)$ is nonnegative if x is nonnegative and near 0. Suppose on the contrary there exist $r_0 > 0$ and $\epsilon > 0$ such that $\phi(r_0) = sr_0 - \epsilon$. There also exists a positive and decreasing sequence $(x_n)_{n=1}^\infty$ approaching 0 and satisfying $\frac{\phi(x_n)}{x_n} > s - \frac{\epsilon}{2r_0}$. Choose x_n small enough and $k \in \mathbb{Z}_{++}$ such that $r_0 \geq kx_n \geq r_0 - \frac{\epsilon}{2s}$. Since ϕ is superadditive and

nondecreasing, we have

$$sr_0 - \epsilon = \phi(r_0) \geq \phi(kx_n) \geq k\phi(x_n) > ksx_n - \frac{k\epsilon x_n}{2r_0} \geq sr_0 - \frac{\epsilon}{2} - \frac{\epsilon}{2} = sr_0 - \epsilon.$$

The above contradiction implies that $\phi(x) \geq sx$ for positive x near 0. Therefore ϕ_1 is strongly maximal and ϕ is implied by ϕ_1. □

4.4. Relation to cut-generating functions

4.4.1. Relation to Gomory–Johnson functions. In this section, we relate both cDFFs and gDFFs to the Gomory–Johnson cut-generating functions. In fact, new DFFs, especially extreme ones, can be discovered by converting Gomory–Johnson functions to DFFs. We first recall the single-row Gomory–Johnson model, which takes the following form:

$$x + \sum_{r \in \mathbb{R}} r\, y(r) = b, \quad b \notin \mathbb{Z}, b > 0, \tag{4.3}$$

$$x \in \mathbb{Z},\ y : \mathbb{R} \to \mathbb{Z}_+, \text{ and } y \text{ has finite support.}$$

Let $\pi\colon \mathbb{R} \to \mathbb{R}$ be a nonnegative function. Then by definition π is a valid Gomory–Johnson function if $\sum_{r \in \mathbb{R}} \pi(r)\, y(r) \geq 1$ holds for any feasible solution (x, y).

As maximal cDFFs and gDFFs are superadditive and minimal Gomory–Johnson functions are subadditive, underlying the conversion is that subtracting subadditive functions from linear functions gives superadditive functions; but the details are more complicated.

THEOREM 4.4.1. *Let π be a minimal piecewise linear Gomory–Johnson function corresponding to a row of the form (4.3) with the right hand side b. Assume π is continuous at 0 from the right. Then there exists $\delta > 0$, such that for all $0 < \lambda < \delta$, the function $\phi_\lambda\colon \mathbb{R} \to \mathbb{R}$, defined by $\phi_\lambda(x) = \frac{bx - \lambda\pi(bx)}{b - \lambda}$, is a maximal gDFF and its restriction $\phi_\lambda|_{[0,1]}$ is a maximal cDFF. These functions have the following properties.*

(i) π has k different slopes if and only if ϕ_λ has k different slopes. If $b > 1$, then π has k different slopes if and only if $\phi_\lambda|_{[0,1]}$ has k different slopes.

(ii) The gDFF ϕ_λ is extreme if π is also continuous with only 2 slope values where its positive slope s satisfies $sb > 1$ and $\lambda = \frac{1}{s}$. The cDFF $\phi_\lambda|_{[0,1]}$ is extreme if π and λ satisfy the previous conditions and $b > 3$.

PROOF. First, we prove ϕ_λ is a maximal gDFF if λ is small enough. As a minimal valid Gomory–Johnson function, π is $\mathbb{Z}$-periodic, $\pi(0) = 0$, π is subadditive and $\pi(x) + \pi(b - x) = 1$ for all $x \in \mathbb{R}$ [**BHK16a**]. Note that ϕ_λ is defined on $\mathbb{R}$, since π is $\mathbb{Z}$-periodic and defined on $\mathbb{R}$. It is not hard to check $\phi_\lambda(0) = 0$. Since ϕ_λ is obtained by subtracting a subadditive function from a linear function, it is superadditive.

The symmetry condition of ϕ_λ is due to the following equation:

$$\begin{aligned}\phi_\lambda(x) + \phi_\lambda(1 - x) &= \frac{bx - \lambda\pi(bx)}{b - \lambda} + \frac{b(1 - x) - \lambda\pi(b(1 - x))}{b - \lambda} \\ &= \frac{b - \lambda(\pi(bx) + \pi(b(1 - x)))}{b - \lambda} = 1.\end{aligned}$$

The last step is from the symmetry condition of π and $\pi(b) = 1$. Since π is piecewise linear and continuous at 0 from the right. Let s be the largest slope of π, then the largest slope of $\pi(bx)$ is bs. Choose $\delta = \frac{1}{s}$, then if $\lambda < \delta$, the slope of bx is always no smaller than the slope of $\lambda\pi(bx)$. There exists an $\epsilon > 0$ such that $\phi_\lambda(x) \geq 0$ for all $x \in (0, \epsilon)$. Therefore, ϕ_λ is a maximal gDFF by Theorem 4.1.2 and $\phi_\lambda|_{[0,1]}$ is a maximal cDFF by Theorem 4.1.1.

Part (i). Suppose π has slope s on the interval (a_i, a_{i+1}), then by calculation $\phi_\lambda(x) = \frac{bx - \lambda\pi(bx)}{b - \lambda}$ has slope $s' = \frac{b(1 - \lambda s)}{b - \lambda}$ on the interval $(\frac{a_i}{b}, \frac{a_{i+1}}{b})$. So if π has slope s_1, s_2 on interval (a_i, a_{i+1}) and (a_j, a_{j+1}) respectively, and ϕ_λ has slope s'_1, s'_2 on interval $(\frac{a_i}{b}, \frac{a_{i+1}}{b})$ and $(\frac{a_j}{b}, \frac{a_{j+1}}{b})$ respectively, then $s_1 = s_2$ if and only if $s'_1 = s'_2$. From the above fact we can conclude π has k different slopes if and only if ϕ_λ has k different slopes.

Since π is $\mathbb{Z}$-periodic, ϕ_λ is quasiperiodic with period $\frac{1}{b}$. If $b > 1$, the interval $[0, 1]$ contains a whole period, which has pieces with all different slope values. So π has k different slopes if and only if $\phi_\lambda|_{[0,1]}$ has k different slopes.

Part (ii). If $sb > 1$ and $\lambda = \frac{1}{s}$, then it is not hard to show ϕ_λ is also continuous piecewise linear with only 2-slope values, and $\phi_\lambda(x) = 0$ for $x \in [0, \frac{\epsilon}{b}]$, i,e., one slope value is 0. From the above results, we know ϕ_λ is a maximal gDFFs.

We use the idea of the extremality test in Section 4.2.3. Since π is extreme from the Gomory–Johnson 2-Slope Theorem [**GJ72a**], all intervals are covered and there are 2 covered components. Suppose $(x, y, x + y)$ is an additive vertex, which means $\pi(x) + \pi(y) = \pi(x + y)$. From arithmetic computation, $(\frac{x}{b}, \frac{y}{b}, \frac{x+y}{b})$ is an additive vertex, i.e., $\phi_\lambda(\frac{x}{b}) + \phi_\lambda(\frac{y}{b}) = \phi_\lambda(\frac{x+y}{b})$. So the additive faces for ϕ_λ are just a scaling of those for π. In regards to ϕ_λ, all intervals are covered and there are only

2 covered components, and $\phi_\lambda(1) = 1$ and $\phi_\lambda(x) = 0$ for $x \in [0, \frac{\epsilon}{b}]$ guarantee that the interval $[0, 1]$ contains the 2 covered components.

Assume $\phi_\lambda = \frac{\phi_1+\phi_2}{2}$, where ϕ_1 and ϕ_2 are maximal gDFFs. By Theorem 4.1.2 and definition, $\phi_1(x) = \phi_2(x) = 0$ for $x \in [0, \frac{\epsilon}{b}]$ and $\phi_1(1) = \phi_2(1) = 1$. The functions ϕ_1 and ϕ_2 satisfy the additivity where ϕ_λ satisfies the additivity, otherwise one of ϕ_1 and ϕ_2 violates the superadditivity. So the additive faces of ϕ_λ are still additive faces of ϕ_1 and ϕ_2. By the Interval Lemma [**BHK16a**] and values at point $\frac{\epsilon}{b}$ and 1, we can show ϕ_1 and ϕ_2 both have 2 covered components and these covered components are the same as those of ϕ_λ. Thus ϕ_1 and ϕ_2 are both continuous 2-slope functions and one slope value is 0, due to nondecreasing condition. Suppose the 2 covered components within $[0, 1]$ are C_1 and C_2, where C_1 and C_2 are disjoint unions of closed intervals. We assume ϕ_1 and ϕ_2 have slope 0 on C_1 and slope s_1 and s_2 on C_2 respectively. The condition $\phi_1(1) = \phi_2(1) = 1$ implies that $0 \times |C_1| + s_1 \times |C_2| = 1$ and $0 \times |C_1| + s_2 \times |C_2| = 1$, where $|C_1|$ and $|C_2|$ denote the measure of C_1 and C_2. So we have $s_1 = s_2$. All these properties guarantee that ϕ_1 and ϕ_2 are equal to each other, therefore ϕ_λ is extreme.

We assume $b > 3$. If all intervals are covered for the restriction $\phi_\lambda|_{[0,1]}$, then we can use the same arguments to show $\phi_\lambda|_{[0,1]}$ is extreme. So we only need to show all intervals are covered by additive faces in the triangular region: $R = \{(x, y) \colon x, y, x + y \in [0, 1]\}$. Maximality of $\phi_\lambda|_{[0,1]}$, especially the symmetry condition, implies that if $(x, y, x + y)$ is an additive vertex, so is $(1 - x - y, y, 1 - x)$. The fact implies that the covered components are symmetric about $x = \frac{1}{2}$, i.e., x is covered $\Leftrightarrow$ $1 - x$ is covered and they are in the same covered components. From the scaling of additive faces of π, the additive faces of $\phi_\lambda|_{[0,1]}$ contained in the square $[0, \frac{1}{b}]^2$ cover the interval $[0, \frac{1}{b}]$, and the additive faces of $\phi_\lambda|_{[0,1]}$ contained in the square $[\frac{1}{b}, \frac{2}{b}] \times [0, \frac{1}{b}]$ cover the interval $[\frac{1}{b}, \frac{2}{b}]$. Similarly, we can use additive faces contained in $\lceil \frac{b}{2} \rceil = \lceil \frac{1}{2} / \frac{1}{b} \rceil$ such whole squares to cover the interval $[0, \frac{1}{2}]$. The condition $b > 3$ guarantees that those $\lceil \frac{b}{2} \rceil$ whole squares are contained in the region R. Together with the symmetry of covered components, we can conclude all intervals are covered, thus $\phi_\lambda|_{[0,1]}$ is extreme.

This concludes the proof of the theorem. □

REMARK 4.4.1. *(1) A construction of k-slope extreme Gomory–Johnson functions has been found for any arbitrary $k \in \mathbb{N}$ [**BCDSP16**]. Therefore, there exist maximal gDFFs and cDFFs with an arbitrary number of slopes.*

(2) The restriction $\phi_\lambda|_{[0,1]}$ is not always extreme as a cDFF even if ϕ_λ is extreme as a gDFF. See an example in Remark 4.5.2.

(3) Note that ϕ_λ is quasiperiodic since π is $\mathbb{Z}$-periodic. However, not all maximal gDFFs are quasiperiodic (See Theorem 4.1.4). Therefore, the conversion is not surjective.

4.4.2. Relation to Yıldız and Cornuéjols cut-generating functions. In this subsection, we focus on gDFFs since they have the extended domain $\mathbb{R}$. We define an infinite dimensional space Y called "the space of nonbasic variables" as $Y = \{y : y\colon \mathbb{R} \to \mathbb{Z}_+ \text{ and } y \text{ has finite support}\}$, and we refer to the zero function as the origin of Y. In this section, we study valid inequalities of certain subsets of the space Y and connect gDFFs to a particular family of cut-generating functions.

In the paper of Yıldız and Cornuéjols [**YC16**], the authors considered the following generalization of the Gomory–Johnson model:

$$x = f + \sum_{r\in\mathbb{R}} r\, y(r), \tag{4.4}$$

$$x \in S,\ y : \mathbb{R} \to \mathbb{Z}_+, \text{ and } y \text{ has finite support},$$

where S can be any nonempty subset of $\mathbb{R}$. A function $\pi\colon \mathbb{R} \to \mathbb{R}$ is called a *valid cut-generating function* if the inequality $\sum_{r\in\mathbb{R}} \pi(r)\, y(r) \geq 1$ holds for all feasible solutions (x, y) to (4.4). In order to ensure that such cut-generating functions exist, they only consider the case $f \notin S$. Otherwise, if $f \in S$, then $(x, y) = (f, 0)$ is a feasible solution and there is no function π which can make the inequality $\sum_{r\in\mathbb{R}} \pi(r)\, y(r) \geq 1$ valid. Note that $y \in Y$ for any feasible solution (x, y) to (4.4), and all valid inequalities in the form of $\sum_{r\in\mathbb{R}} \pi(r)\, y(r) \geq 1$ to (4.4) are inequalities which separate the origin of Y.

We consider two different but related models in the form of (4.4). Let $f = -1$, $S = \{0\}$, and the feasible region $Y_{=1} = \{y : \sum_{r\in\mathbb{R}} r\, y(r) = 1,\ y\colon \mathbb{R} \to \mathbb{Z}_+ \text{ and } y \text{ has finite support}\}$. Let $f = -1$, $S = (-\infty, 0]$, and the feasible region $Y_{\leq 1} = \{y : \sum_{r\in\mathbb{R}} r\, y(r) \leq 1,\ y\colon \mathbb{R} \to \mathbb{Z}_+ \text{ and } y \text{ has finite support}\}$. It is immediate to check that the latter model is the relaxation of the former. Therefore $Y_{=1} \subsetneq Y_{\leq 1}$ and any valid inequality for $Y_{\leq 1}$ is also valid for $Y_{=1}$.

Jeroslow [**Jer79**], Blair [**Bla78**] and Bachem et al. [**BJS82**] studied minimal valid inequalities of the set $Y_{=b} = \{y : \sum_{r\in\mathbb{R}} r\, y(r) = b,\ y\colon \mathbb{R} \to \mathbb{Z}_+ \text{ and } y \text{ has finite support}\}$. Note that $Y_{=b}$ is the set of feasible solutions to (4.4) for $S = \{0\}$, $f = -b$. The notion "minimality" they used is in

fact the restricted minimality in the Yıldız–Cornuéjols model. In this section, we use the terminology introduced by Yıldız and Cornuéjols. Jeroslow [**Jer79**] showed that finite-valued subadditive (restricted minimal) functions are sufficient to generate all necessary valid inequalities of $Y_{=b}$ for bounded mixed integer programs. Kılınç-Karzan and Yang [**KKY15**] discussed whether finite-valued functions are sufficient to generate all necessary inequalities for the convex hull description of disjunctive sets. Interested readers are referred to [**KKY15**] for more details on the sufficiency question. Blair [**Bla78**] extended Jeroslow's result to rational mixed integer programs. Bachem et al. [**BJS82**] characterized restricted minimal cut-generating functions under some continuity assumptions, and showed that restricted minimal functions satisfy the symmetry condition.

Yıldız–Cornuéjols cut-generating functions provide valid inequalities which separate the origin, but clearly there exist other types of valid inequalities. If we let $f \in S$, then there does not exist a valid inequality separating the origin, but we can still consider those which do not separate the origin.

In terms of the relaxation $Y_{\leq 1}$, gDFFs can generate the valid inequalities in the form of $\sum_{r\in\mathbb{R}} \phi(r)\, y(r) \leq 1$, and such inequalities do not separate the origin. Note that there is no valid inequality separating the origin since $0 \in Y_{\leq 1}$. The gDFFs can also be viewed as valid functions in the following pure integer linear programming model:

$$1 \geq \sum_{r\in\mathbb{R}} r\, y(r), \tag{4.5}$$

$$y : \mathbb{R} \to \mathbb{Z}_+, \text{ and } y \text{ has finite support.}$$

Notice that ϕ is a valid gDFF if the inequality $\sum_{r\in\mathbb{R}} \phi(r)\, y(r) \leq 1$ holds for all feasible solutions y to (4.5).

Cut-generating functions provide valid inequalities which separate the origin for $Y_{=1}$, but such inequalities are not valid for $Y_{\leq 1}$. In terms of inequalities that do not separate the origin, any inequality in the form of $\sum_{r\in\mathbb{R}} \phi(r)\, y(r) \leq 1$ generated by some gDFF ϕ is valid for $Y_{\leq 1}$ and hence valid for $Y_{=1}$, since the model of $Y_{\leq 1}$ is the relaxation of that of $Y_{=1}$. Clearly, there also exist valid inequalities which do not separate the origin for $Y_{=1}$ but are not valid for $Y_{\leq 1}$.

Yıldız and Cornuéjols [**YC16**] introduced the notions of minimal, restricted minimal and strongly minimal cut-generating functions. We consider the cut-generating functions to the model

(4.4) when $f = -1$, $S = \{0\}$, and we restate the definitions of minimality of such cut-generating functions. A valid cut-generating function π is called *minimal* if it does not dominate another valid cut-generating function π'. A cut-generating function π' implies a cut-generating function π via scaling if there exists $\beta \geq 1$ such that $\pi \geq \beta\pi'$. A valid cut-generating function π is *restricted minimal* if there is no other cut-generating function π' implying π via scaling. A cut-generating function π' implies a cut-generating function π if there exist α, β, and $\beta \geq 0, \alpha + \beta \geq 1$ such that $\pi(x) \geq \beta\pi'(x) + \alpha x$. A valid cut-generating function π is *strongly minimal* if there is no other cut-generating function π' implying π. Yıldız and Cornuéjols also characterized minimal and restricted minimal functions without additional assumptions. As for the strong minimality and extremality, they mainly focused on the case where $f \in \overline{\text{conv}(S)}$ and $\overline{\text{conv}(S)}$ is full-dimensional. We instead discuss the strong minimality and extremality when $f = -1$, $S = \{0\}$ in Remark 4.4.2.

We show that gDFFs are closely related to cut-generating functions for $Y_{=1}$. The main idea is that valid inequalities generated by cut-generating functions for $Y_{=1}$ can be lifted to valid inequalities generated by gDFFs for the relaxation $Y_{\leq 1}$. The procedure involves adding a multiple of the defining equality $\sum_{r\in\mathbb{R}} r\, y(r) = 1$ to a valid inequality, which is called "tilting" by Aráoz et al. [**AEGJ03**].

We include the characterizations of minimal and restricted minimal cut-generating functions for $Y_{=1}$ below. Bachem et al. had the same characterization [**BJS82**, Theorem] as Theorem 4.4.3 under continuity assumptions at the origin.

THEOREM 4.4.2 ([**YC16**, Theorem 2]). *A function $\pi\colon \mathbb{R} \to \mathbb{R}$ is a minimal cut-generating function for $Y_{=1}$ if and only if $\pi(0) = 0$, π is subadditive, and $\pi(x) = \sup_k\{\frac{1}{k}(1-\pi(1-kx)) : k \in \mathbb{N}\}$.*

THEOREM 4.4.3 ([**YC16**, Proposition 5]). *A function $\pi\colon \mathbb{R} \to \mathbb{R}$ is a restricted minimal cut-generating function for $Y_{=1}$ if and only if π is minimal and $\pi(1) = 1$.*

The following theorem describes the conversion between gDFFs and cut-generating functions for $Y_{=1}$. Unlike Gomory–Johnson cut-generating functions, Yıldız–Cornuéjols cut-generating functions can be converted to gDFFs and the other way around.

THEOREM 4.4.4. *Given a valid/maximal/restricted maximal gDFF ϕ, then for every $0 < \lambda < 1$, the following function is a valid/minimal/restricted minimal cut-generating function for $Y_{=1}$:*

$$\pi_\lambda(x) = \frac{x - (1-\lambda)\,\phi(x)}{\lambda}.$$

Given a valid/minimal/restricted minimal cut-generating function π for $Y_{=1}$, which is Lipschitz continuous at $x = 0$, then there exists $\delta > 0$ such that for all $0 < \lambda < \delta$ the following function is a valid/maximal/restricted maximal gDFF:

$$\phi_\lambda(x) = \frac{x - \lambda\,\pi(x)}{1 - \lambda}, \quad 0 < \lambda < 1.$$

PROOF. *Part (i).* The proof of valid functions.

We want to show that π_λ is a a valid cut-generating function for $Y_{=1}$. Suppose there is a function $y : \mathbb{R} \to \mathbb{Z}_+$, y has finite support, and $\sum_{r\in\mathbb{R}} r\,y(r) = 1$. We want to show that for $\lambda \in (0, 1)$:

$$\begin{aligned}
&\sum_{r\in\mathbb{R}} \pi_\lambda(r)\,y(r) \geq 1 \\
\Leftrightarrow\ &\sum_{r\in\mathbb{R}} \frac{r - (1-\lambda)\,\phi(r)}{\lambda}\,y(r) \geq 1 \\
\Leftrightarrow\ &\sum_{r\in\mathbb{R}} (r - (1-\lambda)\,\phi(r))\,y(r) \geq \lambda \\
\Leftrightarrow\ &\sum_{r\in\mathbb{R}} r\,y(r) - (1-\lambda)\sum_{r\in\mathbb{R}} \phi(r)\,y(r) \geq \lambda \\
\Leftrightarrow\ &\sum_{r\in\mathbb{R}} \phi(r)\,y(r) \leq 1.
\end{aligned}$$

The last step is derived from $\sum_{r\in\mathbb{R}} r\,y(r) = 1$ and ϕ is a gDFF.

On the other hand, the Lipschitz continuity of π at 0 guarantees that $\phi_\lambda(x) \geq 0$ for $x \geq 0$ if λ is small enough. Then the proof for validity of ϕ_λ is analogous to the proof above.

Part (ii). The proof of maximal/minimal functions.

As stated in Theorem 4.4.2, π is minimal if and only if $\pi(0) = 0$, π is subadditive and $\pi(x) = \sup_k\{\frac{1}{k}(1 - \pi(1 - kx)) : k \in \mathbb{N}\}$, which is called the generalized symmetry condition. If $\pi_\lambda(x) =$

$\frac{x-(1-\lambda)\,\phi(x)}{\lambda}$, then $\pi_\lambda(0) = 0$ and π_λ is subadditive.

$$\begin{aligned}
&\sup_k\{\frac{1}{k}(1-\pi_\lambda(1-kx)) : k \in \mathbb{Z}_+\}\\
=\ &\sup_k\{\frac{1}{k}(1-\frac{1-kx-(1-\lambda)\,\phi(1-kx)}{\lambda}) : k \in \mathbb{Z}_+\}\\
=\ &\sup_k\{\frac{kx-(1-\lambda)(1-\phi(1-kx))}{k\lambda} : k \in \mathbb{Z}_+\}\\
=\ &\sup_k\{\frac{x}{\lambda}-\frac{1-\lambda}{\lambda}\frac{1}{k}(1-\phi(1-kx)) : k \in \mathbb{Z}_+\}\\
=\ &\frac{x}{\lambda}-\frac{1-\lambda}{\lambda}\inf_k\{\frac{1}{k}(1-\phi(1-kx)) : k \in \mathbb{Z}_+\}\\
=\ &\frac{x}{\lambda}-\frac{1-\lambda}{\lambda}\phi(x)\\
=\ &\pi_\lambda(x).
\end{aligned}$$

Therefore, π_λ is minimal.

On the other hand, given a minimal cut-generating function π, let $\phi_\lambda(x) = \frac{x-\lambda\,\phi(x)}{1-\lambda}$, then it is easy to see the superadditivity and $\phi_\lambda(0) = 0$. The generalized symmetry can be proven similarly. The Lipschitz continuity of π at 0 implies that $\phi_\lambda(x) \geq 0$ for any $x \geq 0$ if λ is chosen properly.

Part (iii). The proof of restricted maximal/minimal functions.

As stated in Theorem 4.4.3, π is restricted minimal if and only if $\pi(0) = 0$, π is subadditive and $\pi(x) = \sup_k\{\frac{1}{k}(1-\pi(1-kx)) : k \in \mathbb{Z}_+\}$, and $\pi(1) = 1$. Given a restricted maximal gDFF ϕ, we have $\phi(1) = 1$, which implies $\pi_\lambda(1) = 1$.

On the other hand, a restricted minimal π satisfying $\pi(1) = 1$, then $\phi_\lambda(1) = 1$. Based on the maximality of ϕ_λ, we know ϕ_λ is restricted maximal. □

REMARK 4.4.2. *We discuss the distinctions between these two families of functions.*

(i) It is not hard to prove that extreme gDFFs are always maximal. However, unlike cut-generating functions for $Y_{=1}$, extreme gDFFs are not always restricted maximal. For instance, $\phi(x) = 0$ is an extreme gDFF but not restricted maximal.

*(ii) By applying the proof of [**YC16**, Proposition 28], we can show that no strongly minimal cut-generating function for $Y_{=1}$ exists. However, there exist strongly maximal gDFFs by Theorem 4.3.4. Moreover, we can use the same conversion formula in Theorem 4.4.4 to convert a restricted minimal cut-generating function to a strongly maximal gDFF (see Theorem 4.4.5*

below). In fact, it suffices to choose a proper λ *such that* $\lim_{\epsilon\to 0^+} \frac{\phi_\lambda(\epsilon)}{\epsilon} = 0$ *by the characterization of strongly maximal gDFFs (Theorem 4.3.2).*

(iii) There is no extreme piecewise linear cut-generating function π *for* $Y_{=1}$ *which is Lipschitz continuous at* $x = 0$*, except for* $\pi(x) = x$*. If* π *is such an extreme function, then for any* λ *small enough, we claim that* ϕ_λ *is an extreme gDFF. Suppose* $\phi_\lambda = \frac{1}{2}\phi^1 + \frac{1}{2}\phi^2$ *and let* $\pi^1_\lambda, \pi^2_\lambda$ *be the corresponding cut-generating functions of* ϕ^1, ϕ^2 *by Theorem 4.4.4. Note that* $\pi = \frac{1}{2}(\pi^1_\lambda + \pi^2_\lambda)$*, which implies* $\pi = \pi^1_\lambda = \pi^2_\lambda$ *and* $\phi_\lambda = \phi^1_\lambda = \phi^2_\lambda$*. Thus* ϕ_λ *is extreme. By Lemma 4.5.2 and the extremality of* ϕ_λ*, we know* $\phi_\lambda(x) = x$ *or there exists* $\epsilon > 0$*, such that* $\phi_\lambda(x) = 0$ *for* $x \in [0, \epsilon)$*. If* $\phi_\lambda(x) = x$*, then* $\pi(x) = x$*. Otherwise,* $\lim_{x\to 0^+} \frac{\phi_\lambda(x)}{x} = 0$ *for any small enough* λ*. The equation*

$$0 = \lim_{x\to 0^+} \frac{\phi_\lambda(x)}{x} = \lim_{x\to 0^+} \frac{x - \lambda\pi(x)}{(1-\lambda)x} = \frac{1 - \lambda \lim_{x\to 0^+} \frac{\pi(x)}{x}}{1-\lambda}$$

implies $\lim_{x\to 0^+} \frac{\pi(x)}{x} = \frac{1}{\lambda}$ *for any small enough* λ*, which is not possible. Therefore,* π *cannot be extreme except for* $\pi(x) = x$*.*

THEOREM 4.4.5. *Given a non-linear restricted minimal cut-generating function* π *for* $Y_{=1}$*, which is Lipschitz continuous at* 0*, then there exists* $\lambda > 0$ *such that the following function is a strongly maximal gDFF:*

$$\phi_\lambda(x) = \frac{x - \lambda\,\pi(x)}{1-\lambda}.$$

PROOF. From the subadditivity, $\pi(1) = 1$ and Lipschitz continuity of π,

$$1 \le \liminf_{\epsilon\to 0^+} \frac{\pi(\epsilon)}{\epsilon} < +\infty$$

Since π is non-linear, then $s = \liminf_{\epsilon\to 0^+} \frac{\pi(\epsilon)}{\epsilon} > 1$. Let $\lambda = \frac{1}{s}$, then it is immediate to check $\liminf_{\epsilon\to 0^+} \frac{\phi_\lambda(\epsilon)}{\epsilon} = 0$. From the second part in the proof of Theorem 4.3.2, we know $\liminf_{\epsilon\to 0^+} \frac{\phi_\lambda(\epsilon)}{\epsilon} = 0$ suffices in order to prove ϕ_λ is strongly maximal. □

4.5. Two-slope theorem for general DFFs

In this section, we prove a 2-slope theorem for extreme gDFFs, in the spirit of the 2-slope theorem of Gomory and Johnson [**GJ72a**, **GJ72b**]. First, we introduce the lemma showing that extreme gDFFs have certain structures. Similar to Lemma 4.2.1 and Lemma 4.2.2, by studying the superadditivity of maximal gDFFs, it is not hard to prove the following lemma.

LEMMA 4.5.1. *Piecewise linear maximal gDFFs are continuous at* 0 *from the right.*

LEMMA 4.5.2. *Let* ϕ *be a piecewise linear extreme gDFF.*

(i) If ϕ *is strictly increasing, then* $\phi(x) = x$.

(ii) If ϕ *is not strictly increasing, then there exists* $\epsilon > 0$, *such that* $\phi(x) = 0$ *for* $x \in [0, \epsilon)$.

PROOF. Similar to the proof of Lemma 4.2.2, we can assume $\phi(x) = sx$, $x \in [0, a_1)$. If $s = 1$, then $\phi(x) = x$. If $s = 0$, then ϕ is not strictly increasing therefore (ii) holds.

Next, we assume $0 < s < 1$. Define a function:

$$\phi_1(x) = \frac{\phi(x) - sx}{1 - s}.$$

Clearly $\phi_1(x) = 0$ for $x \in [0, a_1)$. The function ϕ_1 is superadditive because it is obtained by subtracting a linear function from a superadditive function. We have that

$$\begin{aligned}
\phi_1(x) &= \frac{\phi(x) - sx}{1 - s} \\
&= \frac{1}{1-s}[\inf_k\{\frac{1}{k}(1 - \phi(1 - kx)) : k \in \mathbb{Z}_+\} - sx] \\
&= \frac{1}{1-s}[\inf_k\{\frac{1}{k}(1 - [(1 - s)\phi_1(1 - kx) + s(1 - kx)]) : k \in \mathbb{Z}_+\} - sx] \\
&= \frac{1}{1-s}[\inf_k\{\frac{1}{k}[(1 - s) + skx - (1 - s)\phi_1(1 - kx)] : k \in \mathbb{Z}_+\} - sx] \\
&= \frac{1}{1-s}\inf_k\{\frac{1}{k}[(1 - s) - (1 - s)\phi_1(1 - kx)] : k \in \mathbb{Z}_+\} \\
&= \inf_k\{\frac{1}{k}(1 - \phi_1(1 - kx)) : k \in \mathbb{Z}_+\}.
\end{aligned}$$

The above equation shows that ϕ_1 satisfies the generalized symmetry condition in Proposition 4.3.1. Therefore, ϕ_1 is also a maximal gDFF. The condition $\phi(x) = sx + (1-s)\phi_1(x)$ implies ϕ is not extreme, since it can be expressed as a convex combination of two different maximal gDFFs: x and ϕ_1. □

From Lemma 4.5.2, we know 0 must be one slope value of a piecewise linear extreme gDFF ϕ, except for $\phi(x) = x$. Next, we introduce the 2-slope theorem for extreme gDFFs. The proof of the following two-slope theorem follows closely that of the Gomory–Johnson's two-slope theorem.

THEOREM 4.5.1 (Two-Slope Theorem for gDFFs). *Let ϕ be a continuous piecewise linear strongly maximal gDFF with only 2 slope values, then ϕ is extreme.*

PROOF. Since ϕ is strongly maximal with 2 slope values, we know one slope value must be 0 by Theorem 4.3.2. Suppose $\phi = \frac{1}{2}(\phi_1 + \phi_2)$, where ϕ_1, ϕ_2 are two maximal gDFFs. From Proposition 4.1.4, we know $\phi(1) = 1$, which implies $\phi_1(1) = \phi_2(1) = 1$. Let s be the other slope value of ϕ. Due to superadditivity of ϕ, s is the limiting slope of ϕ at 0^- and 0 is the limiting slope of ϕ at 0^+. More precisely, there exist $\epsilon, \delta > 0$ such that $\phi(x) = sx$ for $x \in [-\epsilon, 0]$ and $\phi(x) = 0$ for $x \in [0, \delta]$. We want to show ϕ_1, ϕ_2 have slope 0 where ϕ has slope 0, and ϕ_1, ϕ_2 have slope s where ϕ has slope s.

Case 1: Suppose $[a, b]$ is a closed interval where ϕ has slope value 0. Choose $\delta' = \min(\delta, \frac{b-a}{2}) > 0$. Let $I = [0, \delta']$, $J = [a, b - \delta']$, $K = [a, b]$, then I, J, K are three non-empty and proper intervals. Clearly $\phi(x) + \phi(y) = \phi(x + y)$ for $x \in I, y \in J$. Since ϕ_1, ϕ_2 are also superadditive, they satisfy the equality where ϕ satisfy the equality. In other words, $\phi_i(x) + \phi_i(y) = \phi_i(x + y)$ for $x \in I$, $y \in J$, $i = 1, 2$. By the Interval Lemma, ϕ_1 is affine over $[a, b]$ and $[0, \delta']$ with the same slope value l_1. Similarly, ϕ_2 is affine over $[a, b]$ and $[0, \delta']$ with the same slope value l_2. It is clear that $l_1 = l_2 = 0$ since ϕ_1, ϕ_2 are increasing and $0 = \frac{1}{2}(l_1 + l_2)$.

Case 2: Suppose $[c, d]$ is a closed interval where ϕ has slope value s. Choose $\epsilon' = \min(\epsilon, \frac{d-c}{2})$. Let $I = [-\epsilon', 0]$, $J = [c + \epsilon', d]$, $K = [c, d]$, it is clear that $\phi(x) + \phi(y) = \phi(x + y)$ for $x \in I, y \in J$. Similarly we can prove that ϕ_i is affine over $[c, d]$ and $[-\epsilon', 0]$ with the same slope value s_i $(i = 1, 2)$.

Consider the interval $[0 = a_0, a_1, \ldots, a_n = 1]$, where ϕ has slope 0 over $[a_k, a_{k+1}]$ with k even and slope s over $[a_k, a_{k+1}]$ with k odd. Then ϕ_i have slope 0 over $[a_k, a_{k+1}]$ with k even and slope s_i over $[a_k, a_{k+1}]$ with k odd. Let L_0 and L_s be the total length of intervals where ϕ has slope 0 and s, respectively. Then $s \cdot L_s + 0 \cdot L_0 = 1$. It is possible that ϕ_i has jumps at breakpoints a_k, but it can only jump up since ϕ_i is increasing. Suppose $h_i \geq 0$ are the total jumps of ϕ_i at discontinuous points. From $\phi_i(1) = 1$ we can obtain the following equation:

$$s_i \cdot L_s + 0 \cdot L_0 + h_i = 1 \quad (i = 1, 2).$$

Note that $s = \frac{1}{2}(s_1 + s_2)$ and $s \cdot L_s + 0 \cdot L_0 = 1$. So $s_1 = s_2 = s$ and $h_1 = h_2 = 0$ which implies ϕ_1, ϕ_2 are continuous and $\phi_1 = \phi_2 = \phi$. Thus, ϕ is extreme. □

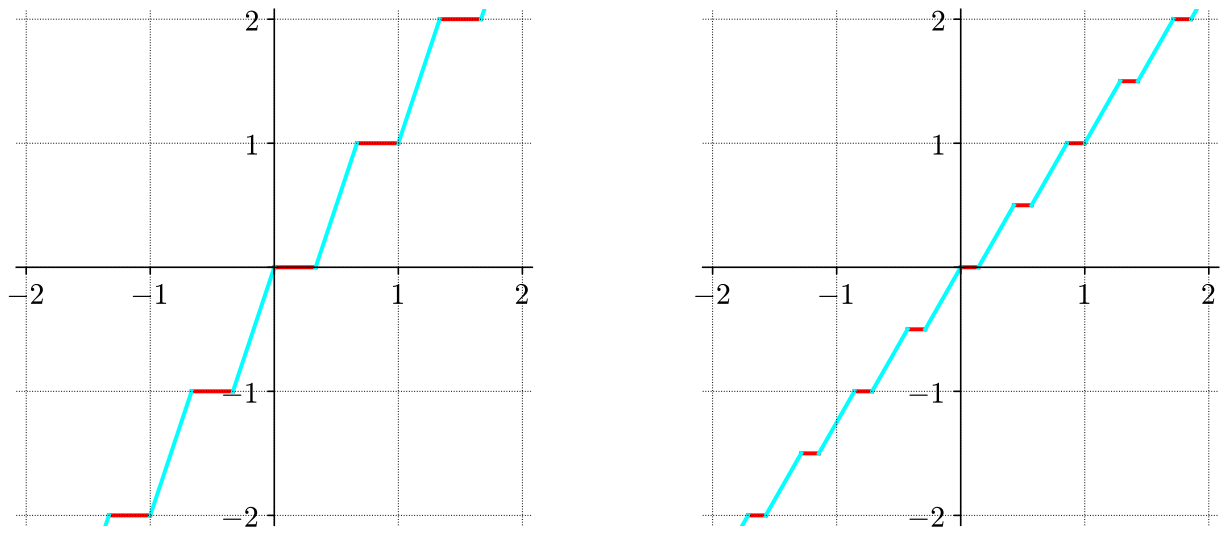

FIGURE 4.2. Graphs of $\phi_{BJ,1}$ [**ACdCR16**, Example 3.1] for $C = 3/2$ (left) and $C = 7/3$ (right).

REMARK 4.5.1. *Alves et al. [**ACdCR16**] showed the following functions by Burdet and Johnson with one parameter $C \geq 1$ are maximal gDFFs, where $\{a\}$ represents the fractional part of a:*

$$\phi_{BJ,1}(x;C) = \frac{\lfloor Cx \rfloor + \max(0, \frac{\{Cx\}-\{C\}}{1-\{C\}})}{\lfloor C \rfloor}.$$

We can prove that they are extreme. If $C \in \mathbb{N}$, then $\phi_{BJ,1}(x) = x$. If $C \notin \mathbb{N}$, $\phi_{BJ,1}$ is a continuous 2-slope maximal gDFF with one slope value 0, therefore it is extreme by Theorem 4.5.1. Figure 4.2 shows two examples of $\phi_{BJ,1}$ and they are constructed by the Python function `phi_bj_1_gdff`.

REMARK 4.5.2. *However, the analogous result does not hold for cDFFs. In other words, the restriction $\phi|_{[0,1]}$ is not always extreme as a cDFF even if ϕ is extreme as a gDFF. In fact, $\phi_{BJ,1}(x;C)|_{[0,1]}$ is not extreme as a cDFF for $1 < C < 2$, though it is a continuous 2-slope maximal cDFF with one slope value 0. We found an interesting counterexample by computer-based search; it is shown in Figure 4.3 and Figure 4.4.*

4.6. Restricted maximal general DFFs are almost extreme

In the previous section, we have shown that any continuous 2-slope strongly maximal gDFF is extreme. In this section, we prove that extreme gDFFs are dense in the set of continuous restricted maximal gDFFs. Equivalently, for any given continuous restricted maximal gDFF ϕ, there exists an

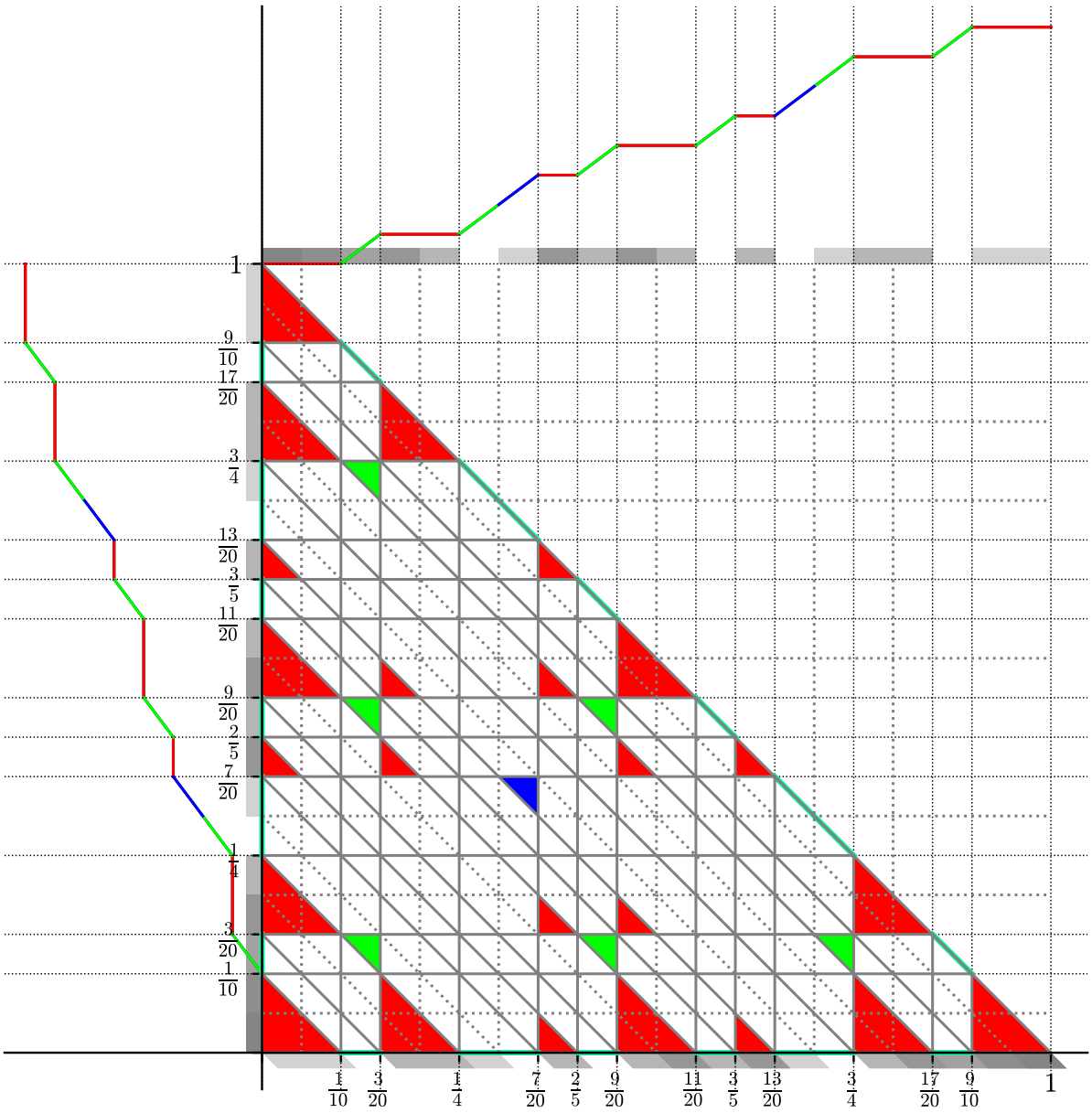

FIGURE 4.3. Continuous 2-slope maximal non-extreme cDFF `w_2slope_3covered_nonextreme` with 3 connected covered components. We use 3 different colors to color additive faces to represent 3 different covered components. The colors on the function are consistent with the colors of additive faces. We plot the function on the left and upper border. The shadows represent covered components from the projections of additive faces in 3 directions.

extreme gDFF ϕ_{ext} which approximates ϕ as close as desired (with the infinity norm). The idea of the proof is inspired by the approximation theorem of Gomory–Johnson functions [**BHM16**]. We

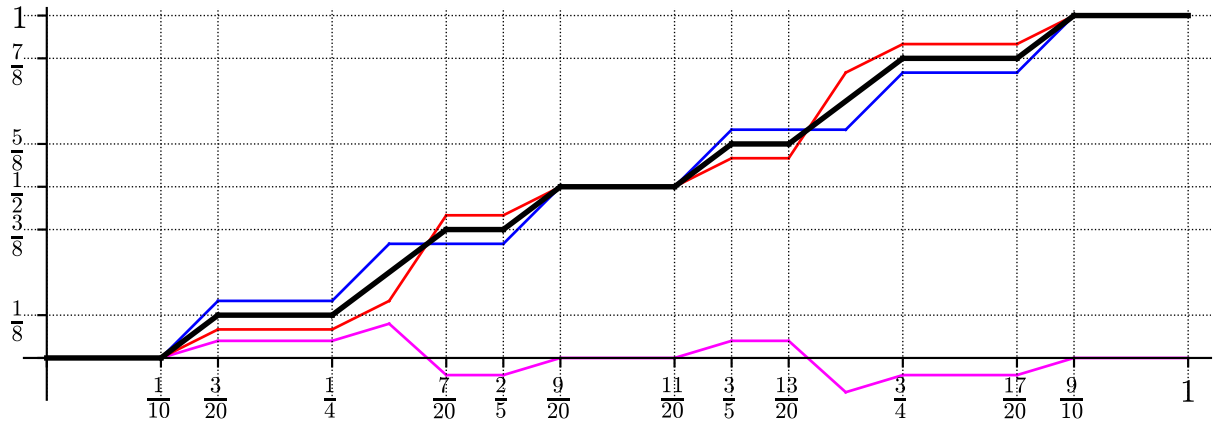

FIGURE 4.4. Continuous 2-slope maximal non-extreme cDFF `w_2slope_3covered_nonextreme` (in black), an effective perturbation function $\tilde{\pi}$ (magenta), and functions $\phi^{\pm} = \phi \pm \epsilon\tilde{\phi}$ (blue and red).

first introduce the main theorem in this section. The approximation[1] is implemented for piecewise linear functions with finitely many pieces.

THEOREM 4.6.1 (Approximation Theorem). *Let ϕ be a continuous restricted maximal gDFF, then for any $\epsilon > 0$, there exists an extreme gDFF ϕ_{ext} such that $\|\phi - \phi_{\text{ext}}\|_\infty < \epsilon$.*

REMARK 4.6.1. *The result cannot be extended to maximal gDFF. Note that $\phi(x) = sx$ is maximal but not extreme for $0 < s < 1$. Any non-trivial extreme gDFF ϕ' satisfies $\phi'(1) = 1$, and $\phi'(1) - \phi(1) = 1 - s > 0$ and $1 - s$ is a fixed positive constant. Therefore, $\phi(x) = sx$ cannot be arbitrarily approximated by an extreme gDFF.*

We briefly explain the structure of the proof. First, we approximate a continuous restricted maximal gDFF ϕ by a piecewise linear maximal gDFF ϕ_{pwl}. Next, we perturb ϕ_{pwl} such that the new maximal gDFF ϕ_{loose} satisfies $\nabla\phi_{\text{loose}}(x, y) > \gamma > 0$ for "most" $(x, y) \in \mathbb{R}^2$. After applying the 2-slope fill-in procedure to ϕ_{loose}, we get a superadditive 2-slope function $\phi_{\text{fill-in}}$, which is not symmetric anymore. Finally, we symmetrize $\phi_{\text{fill-in}}$ to get the desired ϕ_{ext}.

4.6.1. Approximate ϕ using piecewise linear functions ϕ_{pwl}. By studying the superadditivity of maximal gDFFs near the origin, it is not hard to prove Lemma 4.6.1. By choosing a large enough $q \in \mathbb{N}$ and interpolating the function over $\frac{1}{q}\mathbb{Z}$ we can obtain Lemma 4.6.2.

LEMMA 4.6.1. *Any continuous restricted maximal gDFF ϕ is uniformly continuous.*

[1]See the constructor `two_slope_approximation_gdff_linear`.

PROOF. Since ϕ is continuous at 0 and nondecreasing, for any $\epsilon > 0$, there exists $\delta > 0$ such that $-\delta < t \leq 0$ implies $-\epsilon < \phi(t) \leq 0$. For any x, y with $-\delta < x - y < 0$, by superadditivity we have $0 \geq \phi(x) - \phi(y) \geq \phi(x - y) > -\epsilon$. So ϕ is uniformly continuous. □

LEMMA 4.6.2. *Let ϕ be a continuous restricted maximal gDFF, then for any $\epsilon > 0$, there exists a piecewise linear continuous restricted maximal gDFF ϕ_{pwl}, such that $\|\phi - \phi_{\text{pwl}}\|_\infty < \frac{\epsilon}{3}$.*

PROOF. By Lemma 4.6.1, ϕ is uniformly continuous. For any $\epsilon > 0$, there exists $\delta > 0$ such that $|x - y| < \delta$ implies $|\phi(x) - \phi(y)| < \frac{\epsilon}{3}$. Choose $q \in \mathbb{N}$ large enough such that $\frac{1}{q} < \delta$, then $0 \leq \phi(\frac{n+1}{q}) - \phi(\frac{n}{q}) < \frac{\epsilon}{3}$ for any integer n. We claim that the interpolation of $\phi|_{\frac{1}{q}\mathbb{Z}}$ is the desired ϕ_{pwl}.

We first prove $\|\phi - \phi_{\text{pwl}}\|_\infty < \frac{\epsilon}{3}$. For any $x \in \mathbb{R}$, suppose $\frac{n}{q} \leq x < \frac{n+1}{q}$ for some integer n. Due to the choice of q and δ,

$$\phi(x) - \phi_{\text{pwl}}(x) \leq \phi(\frac{n+1}{q}) - \phi_{\text{pwl}}(\frac{n}{q}) = \phi(\frac{n+1}{q}) - \phi(\frac{n}{q}) < \frac{\epsilon}{3}$$

Similarly we can prove $\phi(x) - \phi_{\text{pwl}}(x) > -\frac{\epsilon}{3}$. So $\|\phi - \phi_{\text{pwl}}\|_\infty < \frac{\epsilon}{3}$.

Since $\phi|_{\frac{1}{q}\mathbb{Z}}$ is superadditive and satisfies the symmetry condition, then ϕ_{pwl} is also superadditive and satisfies the symmetry condition due to piecewise linearity of ϕ_{pwl}. Therefore, ϕ_{pwl} is the desired function. □

4.6.2. Perturbation function $\phi_{s,\delta}$. Next, we introduce a parametric family of restricted maximal gDFFs $\phi_{s,\delta}$ which will be used to perturb ϕ_{pwl}. Define

$$\phi_{s,\delta}(x) = \begin{cases} sx - s\delta & \text{if } x < -\delta \\ 2sx & \text{if } -\delta \leq x < 0 \\ 0 & \text{if } 0 \leq x < \delta \\ \frac{1}{1-2\delta}x - \frac{\delta}{1-2\delta} & \text{if } \delta \leq x < 1 - \delta \cdot \\ 1 & 1 - \delta \leq x < 1 \\ 2sx - 2s + 1 & 1 \leq x < 1 + \delta \\ sx - s + 1 + s\delta & x \geq 1 + \delta \end{cases}$$

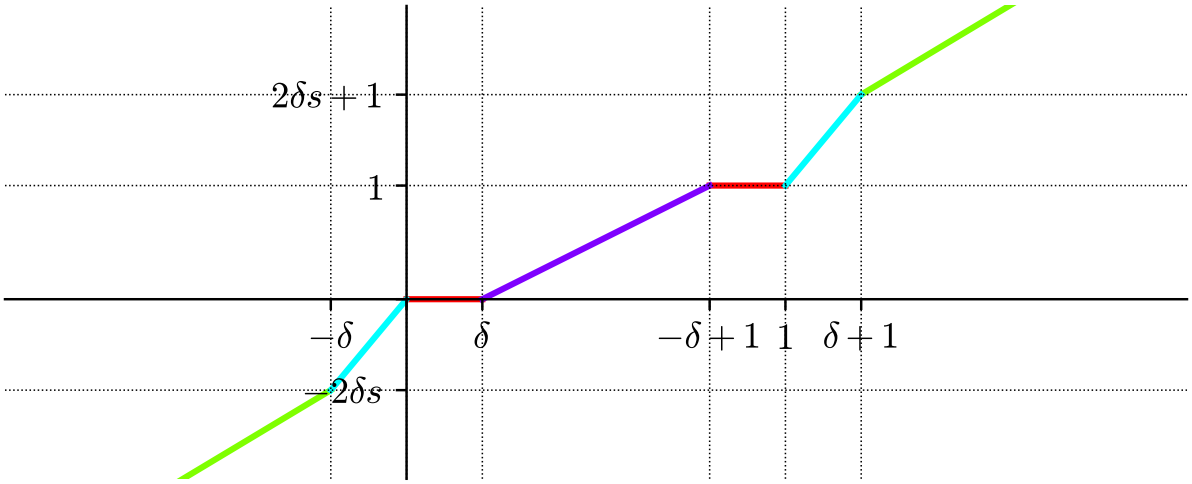

FIGURE 4.5. The graph of $\phi_{s,\delta}$ for $s = 2$ and $\delta = \frac{1}{5}$

The function $\phi_{s,\delta}$ is a continuous piecewise linear function, which has breakpoints: $-\delta, 0, \delta, 1-\delta, 1, 1+\delta$ and slope values: $s, 2s, 0, \frac{1}{1-s\delta}, 0, 2s, s$ in each affine piece. Figure 4.5 is the graph of one $\phi_{s,\delta}$ function constructed by the Python function `phi_s_delta`.

Let $E_\delta = \{(x,y) \in \mathbb{R}^2 : -\delta < x < \delta \text{ or } -\delta < y < \delta \text{ or } 1-\delta < x+y < 1+\delta\}$.

LEMMA 4.6.3. *The function $\phi_{s,\delta}$ is a continuous restricted maximal gDFF and $\nabla\phi_{s,\delta}(x,y) \geq \delta$ for $(x,y) \notin E_\delta$, if $s > 1$ and $0 < \delta < \min\{\frac{s-1}{2s}, \frac{1}{3}\}$.*

Verifying the above properties of $\phi_{s,\delta}$ is a routine computation by analyzing the superadditivity slack at every vertex in the two-dimensional polyhedral complex (cf. Section 4.2.1) of $\phi_{s,\delta}$. We explain why it suffices to check the superadditive slack at finitely many vertices in the two-dimensional polyhedral complex.

First, we generalize the definition of the two-dimensional polyhedral complex to piecewise linear functions with unbounded domain. Let $\phi\colon \mathbb{R} \to \mathbb{R}$ be a piecewise linear function with finitely many pieces with breakpoints $x_1 < x_2 < \cdots < x_n$. To express the domains of linearity of $\nabla\phi(x,y)$, and thus domains of additivity and strict superadditivity, we introduce the two-dimensional polyhedral complex $\Delta\mathcal{P}$, similar to the definition in Section 4.2.1 for cDFFs. The faces F of the complex are defined as follows. Let $I, J, K \in \mathcal{P}$, so each of I, J, K is either a breakpoint of ϕ or a closed interval delimited by two consecutive breakpoints including $\pm\infty$. Then $F = F(I,J,K) = \{\,(x,y) \in \mathbb{R}\times\mathbb{R} : x \in I,\, y \in J,\, x+y \in K\,\}$. Let $F \in \Delta\mathcal{P}$ and observe that the piecewise linearity of ϕ induces piecewise linearity of $\nabla\phi$.

LEMMA 4.6.4. *Let $\phi\colon \mathbb{R} \to \mathbb{R}$ be a continuous piecewise linear function with finitely many pieces with breakpoints $x_1 < x_2 < \cdots < 0 < \cdots < x_n$ and ϕ has the same slope s on $(-\infty, x_1]$ and $[x_n, \infty)$. Consider a one-dimensional unbounded face F where one of I, J, K is a finite breakpoint and the other two are unbounded closed intervals, $(-\infty, x_1]$ or $[x_n, \infty)$. Then $\nabla\phi(x, y)$ is a constant along the face F.*

PROOF. We only provide the proof for one case, the proofs for other cases are similar.

Suppose $I = \{x_i\}$, $J = K = [x_n, \infty)$. The vertex of F is $(x, y) = (x_i, x_n)$ if $x_i \geq 0$ and $(x, y) = (x_i, x_n - x_i)$ if $x_i < 0$. If $x_i \geq 0$, we claim that $\nabla\phi(x, y) = \nabla\phi(x_i, x_n)$ for $(x, y) \in F$. We have

$$\nabla\phi(x, y) = \phi(x_i + y) - \phi(x_i) - \phi(y) = \phi(x_i + x_n) - \phi(x_i) - \phi(x_n) = \nabla\phi(x_i, x_n).$$

The second step in the above equation is due to ϕ is affine on $[x_n, \infty)$ and $x_i + x_n, y \geq x_n$.

If $x_i < 0$, we claim that $\nabla\phi(x, y) = \nabla\phi(x_i, x_n - x_i)$ for $(x, y) \in F$. We can deduce that

$$\begin{aligned}
\nabla\phi(x, y) &= \phi(x_i + y) - \phi(x_i) - \phi(y) \\
&= (\phi(x_n) + s(x_i + y - x_n)) - \phi(x_i) - (\phi(x_n - x_i) + s(x_i + y - x_n)) \\
&= \phi(x_n) - \phi(x_i) - \phi(x_n - x_i) = \nabla\phi(x_i, x_n - x_i).
\end{aligned}$$

The second step in the above equation is due to ϕ has slope s on $[x_n, \infty)$ and $x_n - x_i, x_i + y \geq x_n$.

Case 2: Suppose $K = \{x_i\}$, $I = [x_n, \infty)$ and $J = (-\infty, x_1]$. The vertex of F is $(x, y) = (x_n, x_i - x_n)$ if $x_i \leq x_1 + x_n$ and $(x, y) = (x_i - x_1, x_1)$ if $x_i > x_1 + x_n$.

If $x_i \leq x_1 + x_n$, we claim that $\nabla\phi(x, y) = \nabla\phi(x_n, x_i - x_n)$ for $(x, y) \in F$. Similarly we have

$$\begin{aligned}
\nabla\phi(x, y) &= \phi(x_i) - \phi(x) - \phi(x_i - x) \\
&= \phi(x_i) - (\phi(x_n) + s(x - x_n)) - (\phi(x_i - x_n) - s(x - x_n)) \\
&= \phi(x_i) - \phi(x_n) - \phi(x_i - x_n) = \nabla\phi(x_n, x_i - x_n).
\end{aligned}$$

The second step in the above equation is due to ϕ has the same slope s on $(-\infty, x_1]$ and $[x_n, \infty)$ and $x \geq x_n$, $y = x_i - x \leq x_i - x_n \leq x_1$.

If $x_i > x_1 + x_n$, we claim that $\nabla\phi(x,y) = \nabla\phi(x_i - x_1, x_1)$ for $(x,y) \in F$, by the following equation:

$$\begin{aligned}\nabla\phi(x,y) &= \phi(x_i) - \phi(x_i - y) - \phi(y) \\ &= \phi(x_i) - (\phi(x_i - x_1) + s(x_1 - y)) - (\phi(x_1) - s(x_1 - y)) \\ &= \phi(x_i) - \phi(x_i - x_1) - \phi(x_1) = \nabla\phi(x_i - x_1, x_1).\end{aligned}$$

The second step in the above equation is due to ϕ has the same slope s on $(-\infty, x_1]$ and $[x_n, \infty)$ and $y \geq x_1$, $x = x_i - y \geq x_i - x_1 \geq x_n$.

Therefore, $\nabla\phi(x,y)$ is a constant for (x,y) in any fixed one-dimensional unbounded face. □

By using the piecewise linearity of $\nabla\phi$, we can prove the following lemma. Thus, it suffices to check the superadditivity slack at finitely many vertices in the two-dimensional polyhedral complex to prove the desired properties of $\phi_{s,\delta}$, i.e. $\nabla\phi_{s,\delta}(x,y) \geq \delta$ for $(x,y) \notin E_\delta$, if $s > 1$ and $0 < \delta < \min\{\frac{s-1}{2s}, \frac{1}{3}\}$.

LEMMA 4.6.5. *Define the two-dimensional polyhedral complex $\Delta\mathcal{P}$ of the function $\phi_{s,\delta}$. If $\nabla\phi_{s,\delta}(x,y) \geq \delta$ for any zero-dimensional face $(x,y) \notin E_\delta$, then $\nabla\phi_{s,\delta}(x,y) \geq \delta$ for $(x,y) \notin E_\delta$.*

PROOF. Observe that $\mathbb{R}^2 - E_\delta$ is the union of finite two-dimensional faces. So we only need to show $\nabla\phi_{s,\delta}(x,y) \geq \delta$ for $(x,y) \notin E_\delta$ and (x,y) in some two-dimensional face F.

If F is bounded, then $\nabla\phi_{s,\delta}(x,y) \geq \delta$ since the inequality holds for vertices of F and $\nabla\phi$ is affine over F.

Suppose that F is unbounded and is enclosed by some bounded and some unbounded one-dimensional faces. For those bounded one-dimensional faces, $\nabla\phi_{s,\delta}(x,y) \geq \delta$ holds since the inequality holds for vertices. For any unbounded one-dimensional face F', by Lemma 4.6.4, the $\nabla\phi$ is constant and equals to the value at the vertex of F'. We have showed that $\nabla\phi_{s,\delta}(x,y) \geq \delta$ holds for any (x,y) in the enclosing one-dimensional faces, then the inequality holds for $(x,y) \in F$ due to the piecewise linearity of $\nabla\phi$. □

REMARK 4.6.2. *In the software [**KZHW20**], we define a parametric family of functions $\phi_{s,\delta}$ with two variables s and δ. From the definition of $\phi_{s,\delta}$, it is clear that $\phi_{s,\delta}$ satisfies the symmetry condition. Although $\phi_{s,\delta}$ is defined in the unbounded domain $\mathbb{R}$, $\nabla\phi$ only depends on the values at the vertices of $\Delta\mathcal{P}$ which is a bounded and finite set, based on the above lemma. In order to*

show the superadditivity and $\nabla\phi_{s,\delta}(x,y) \geq \delta$ for $(x,y) \notin E_\delta$, only $\nabla\phi_{s,\delta}$ at all vertices of $\Delta\mathcal{P}$ needs to be checked. The Python function `phi_s_delta_is_superadditive_almost_strict` *verifies the claim for given numerical values of s and δ that satisfy the hypotheses of Lemma 4.6.3. Using the method of parametric metaprogramming introduced in [**KZ16**], the documentation tests of the Python function* `phi_s_delta_check_claim` *verify the claim for the full parametric family, providing an automatic proof of Lemma 4.6.3.*

4.6.3. Approximate ϕ_{pwl} using ϕ_{loose}.

LEMMA 4.6.6. *Let ϕ_{pwl} be a piecewise linear continuous restricted maximal gDFF, then for any $\epsilon > 0$, there exists a piecewise linear continuous restricted maximal gDFF ϕ_{loose} satisfying: (i) $\|\phi_{\text{loose}} - \phi_{\text{pwl}}\|_\infty < \frac{\epsilon}{3}$; (ii) there exist $\delta, \gamma > 0$ such that $\nabla\phi_{\text{loose}}(x,y) \geq \gamma$ for (x,y) not in E_δ.*

PROOF. By Proposition 4.1.3, let $t = \lim_{x\to\infty} \frac{\phi_{\text{pwl}}(x)}{x}$, then $tx - t + 1 \leq \phi_{\text{pwl}}(x) \leq tx$. We can assume $t > 1$, otherwise ϕ_{pwl} is the identity function and the result is trivial. Choose $s = t$ and δ small enough such that $0 < \delta < \min\{\frac{s-1}{2s}, \frac{1}{3}, \frac{1}{q}\}$, where q is the denominator of breakpoints of ϕ_{pwl} in previous lemma. We know that the limiting slope of maximal gDFF $\phi_{t,\delta}$ is also t and $tx - t + 1 \leq \phi_{t,\delta}(x) \leq tx$, which implies $\|\phi_{t,\delta} - \phi_{\text{pwl}}\|_\infty \leq t - 1$.

Define $\phi_{\text{loose}} = (1 - \frac{\epsilon}{3(t-1)})\,\phi_{\text{pwl}} + \frac{\epsilon}{3(t-1)}\,\phi_{t,\delta}$. It is immediate to check ϕ_{loose} is restricted maximal, and $\|\phi_{\text{loose}} - \phi_{\text{pwl}}\|_\infty < \frac{\epsilon}{3}$ is due to $\|\phi_{t,\delta} - \phi_{\text{pwl}}\|_\infty \leq t - 1$. Based on the property of $\phi_{t,\delta}$, $\nabla\phi_{\text{loose}}(x,y) = (1 - \frac{\epsilon}{3(t-1)})\nabla\phi_{\text{pwl}}(x,y) + \frac{\epsilon}{3(t-1)}\nabla\phi_{t,\delta}(x,y) \geq \frac{\epsilon}{3(t-1)}\nabla\phi_{t,\delta}(x,y) \geq \gamma = \frac{\epsilon\delta}{3(t-1)}$ for (x,y) not in E_δ. □

4.6.4. Approximate ϕ_{loose} using extreme function ϕ_{ext}.

LEMMA 4.6.7. *Given a piecewise linear continuous restricted maximal gDFF ϕ_{loose} satisfying properties in previous lemma, there exists an extreme gDFF ϕ_{ext} such that $\|\phi_{\text{loose}} - \phi_{\text{ext}}\|_\infty < \frac{\epsilon}{3}$.*

PROOF. Let $s^+ \geq 0$ be the largest slope of ϕ_{loose} and $\phi_{\text{loose}}(x) = s^+x$ for $x \in [-\delta, 0]$ where δ is chosen from previous lemma. Choose $q' \in \mathbb{N}_+$ such that $\frac{1}{q'}s^+ < \min\{\frac{\epsilon}{3}, \frac{\gamma}{3} = \frac{\epsilon\delta}{9(t-1)}\}$ and the breakpoints of ϕ_{loose} and $\frac{1}{2}$ are contained in $U = \frac{1}{q'}\mathbb{Z}$. Note that we can always choose a rational δ to ensure that the last step is feasible. Define a function $g\colon \mathbb{R} \to \mathbb{R}$ and a 2-slope function $\phi_{\text{fill-in}}\colon [0,1] \to [0,1]$:

$$g(x) = \begin{cases} 0 & \text{if } x \geq 0 \\ s^+x & \text{if } x < 0 \end{cases},$$

$$\phi_{\text{fill-in}}(x) = \max_{u \in U}\{\phi_{\text{loose}}(u) + g(x-u)\}.$$

We claim that $\phi_{\text{fill-in}}$ is a continuous 2-slope superadditive function and $\phi_{\text{fill-in}} \leq \phi_{\text{loose}}$, $\phi_{\text{fill-in}}|_U = \phi|_U$. The proof is similar to that of [**GJ72a**, Theorem 3.3]. $|\phi_{\text{fill-in}}(x) - \phi_{\text{loose}}(x)| \leq \frac{1}{q}s^+ < \frac{\epsilon}{3}$ implies that $\|\phi_{\text{loose}} - \phi_{\text{fill-in}}\| < \frac{1}{q'}s^+ < \frac{\epsilon}{3}$. However, $\phi_{\text{fill-in}}$ does not necessarily satisfy the symmetry condition. If we symmetrize it and define the following function:

$$\phi_{\text{ext}}(x) = \begin{cases} \phi_{\text{fill-in}}(x) & \text{if } x \leq \frac{1}{2} \\ 1 - \phi_{\text{fill-in}}(1-x) & \text{if } x > \frac{1}{2} \end{cases}.$$

We claim that ϕ_{ext} is the desired function. It is immediate to check $\phi_{\text{ext}}(0) = 0$, ϕ_{ext} is a 2-slope continuous function and it automatically satisfies the symmetry condition. Since we use slope 0 and s^+ to do the fill-in procedure, the limiting slope of ϕ_{ext} at 0^+ is 0. Notice that $\|\phi_{\text{loose}} - \phi_{\text{ext}}\| = \|\phi_{\text{loose}} - \phi_{\text{fill-in}}\| < \frac{1}{q'}s^+ < \frac{\epsilon}{3}$ because they both satisfy the symmetry condition. So we only need to prove ϕ_{ext} is superadditive.

Case 1: If (x, y) is not in E_δ, $\nabla\phi_{\text{ext}}(x, y) \geq \nabla\phi_{\text{loose}}(x, y) - \frac{\epsilon\delta}{9(t-1)} - \frac{\epsilon\delta}{9(t-1)} - \frac{\epsilon\delta}{9(t-1)} \geq \frac{\epsilon\delta}{3(t-1)} - \frac{\epsilon\delta}{3(t-1)} = 0$.

Case 2: If $0 \leq x \leq \delta$, there are also three sub cases.

(i) If $y, x + y \leq \frac{1}{2}$, then $\nabla\phi_{\text{ext}}(x, y) = \nabla\phi_{\text{fill-in}}(x, y) \geq 0$.

(ii) If $y \leq \frac{1}{2}$ and $x + y > \frac{1}{2}$, then $\nabla\phi_{\text{ext}}(x, y) = 1 - \phi_{\text{fill-in}}(1 - x - y) - \phi_{\text{fill-in}}(x) - \phi_{\text{fill-in}}(y) \geq 1 - \phi_{\text{loose}}(1 - x - y) - \phi_{\text{loose}}(x) - \phi_{\text{loose}}(y) \geq 0$. Here we use the fact that $\phi_{\text{loose}} \geq \phi_{\text{fill-in}}$ and ϕ_{loose} is a maximal gDFF.

(iii) If $y, x + y > \frac{1}{2}$, then $\nabla\phi_{\text{ext}}(x, y) = (1 - \phi_{\text{fill-in}}(1 - x - y)) - \phi_{\text{fill-in}}(x) - (1 - \phi_{\text{fill-in}}(1 - y)) = \phi_{\text{fill-in}}(1 - y) - \phi_{\text{fill-in}}(1 - x - y) - \phi_{\text{fill-in}}(x) \geq 0$ due to superadditivity of $\phi_{\text{fill-in}}$.

Case 3: If $0 > x \geq -\delta$, based on the choice of δ and s^+, we know $\phi_{\text{ext}}(x) = s^+x$ for $0 > x \geq -\delta$. For any $y \in \mathbb{R}$, $\phi_{\text{ext}}(x + y) - \phi_{\text{ext}}(y) \geq s^+x = \phi_{\text{ext}}(x)$ since ϕ_{ext} is a 2-slope function and s^+ is the larger slope.

Similarly we can prove $\nabla\phi_{\text{ext}}(x, y) \geq 0$ if $-\delta \leq y \leq \delta$.

Case 4: If $1 - \delta \leq x + y \leq 1 + \delta$, let $\beta = 1 - x - y$ and $-\delta \leq \beta \leq \delta$, so by case 2 and 3, $\phi_{\text{ext}}(\beta) + \phi_{\text{ext}}(x) \leq \phi_{\text{ext}}(\beta + x)$. Then we have $\phi_{\text{ext}}(x + y) = \phi_{\text{ext}}(1 - \beta) = 1 - \phi_{\text{ext}}(\beta) = 1 - \phi_{\text{ext}}(\beta) + \phi_{\text{ext}}(x) - \phi_{\text{ext}}(x) \geq 1 - \phi_{\text{ext}}(\beta + x) + \phi_{\text{ext}}(x) = 1 - \phi_{\text{ext}}(1 - y) + \phi_{\text{ext}}(x) = \phi_{\text{ext}}(y) + \phi_{\text{ext}}(x)$.

We have shown that ϕ_{ext} is superadditive, then it is a continuous 2-slope strongly maximal gDFF. By the Two-Slope Theorem (Theorem 4.5.1), ϕ_{ext} is extreme. □

Combine the previous lemmas, and we have $\|\phi - \phi_{\text{ext}}\|_\infty \leq \|\phi - \phi_{\text{pwl}}\|_\infty + \|\phi_{\text{pwl}} - \phi_{\text{loose}}\|_\infty + \|\phi_{\text{loose}} - \phi_{\text{ext}}\|_\infty < 3 \times \frac{\epsilon}{3} = \epsilon$. We can conclude the Approximation Theorem. Observe that we always use 0 as one slope value in the fill-in procedure. It is due to the fact (Lemma 4.5.2) that almost all extreme gDFFs have 0 as the limiting slope at 0^+.

4.7. Conclusion

In this chapter, we have studied two types of dual-feasible functions: classical DFFs and general DFFs. Our results are mainly inspired by the characterizations of DFFs and Gomory–Johnson cut-generating functions. Specifically, there is a natural mutual transformation relation between subadditivity and superadditivity. The cut-generating functions have been studied extensively for general integer optimization problems, whereas DFFs are relatively restricted to certain combinatorial optimization problems. We apply the research on cut-generating functions and prove analogous results for DFFs, and we also write a software for DFFs following the framework of Gomory–Johnson cut-generating functions. Our research connects both fields. We not only foster the DFF community by applying results of cut-generating function, but also provide a tunnel of knowledge in the sense that a new result in one field can be potentially transferred to the other field.

There are several future research directions regarding DFFs. First, we only study analytical properties of DFFs and have not applied them in any computational experiment. One future work is to test how a richer family of DFFs can improve computational performance of solving combinatorial optimization problems. Ideally, the efficiently computed bound generated by DFFs can be improved given more available DFFs. On the other hand, for certain combinatorial optimization problems and DFFs with known good computational performance, one can study if the cutting plane algorithm can be improved by a good cut selection strategy. Another research direction is regarding the multi-dimensional generalization. Multi-row cut-generating functions was first proposed in late 1960s [**Gom65**] and have received renewed attention since late 2000s [**Esp08**, **Poi12**, **Zam09**]. As the generalization of single-row cut-generating functions, multi-row cut-generating functions are used to generate cuts from multiple rows of the simplex tableau simultaneously. Multi-dimensional DFFs have been used in vector packing [**AdCCR14**], orthogonal packing [**FS04**], and tree decomposition

problems [**ACdCR16**]. It is worth investigating the relation between multi-dimensional DFFs and cut-generating functions.

Bibliography

[AAF98] C. Adjiman, I. Androulakis, and C. Floudas, *A global optimization method, αBB, for general twice-differentiable constrained NLPs-II. implementation and computational results*, Computers and Chemical engineering **22** (1998), 1159–1179.

[AC91] D. Applegate and W. Cook, *A computational study of the job-shop scheduling problem*, ORSA Journal on computing **3** (1991), no. 2, 149–156.

[ACdCR16] C. Alves, F. Clautiaux, J. V. de Carvalho, and J. Rietz, *Dual-feasible functions for integer programming and combinatorial optimization: Basics, extensions and applications*, EURO Advanced Tutorials on Operational Research, Springer, 2016.

[Ach07] T. Achterberg, *Conflict analysis in mixed integer programming*, Discrete Optimization **4** (2007), no. 1, 4–20.

[AdCCR14] C. Alves, J. V. de Carvalho, F. Clautiaux, and J. Rietz, *Multidimensional dual-feasible functions and fast lower bounds for the vector packing problem*, European Journal of Operational Research **233** (2014), no. 1, 43–63.

[AEGJ03] J. Aráoz, L. Evans, R. E. Gomory, and E. L. Johnson, *Cyclic group and knapsack facets*, Mathematical Programming, Series B **96** (2003), 377–408.

[AW13] T. Achterberg and R. Wunderling, *Mixed integer programming: Analyzing 12 years of progress*, Facets of combinatorial optimization, Springer, 2013, pp. 449–481.

[Bak16] M. Baker, *Is there a reproducibility crisis? a Nature survey lifts the lid on how researchers view the crisis rocking science and what they think will help*, Nature **533** (2016), no. 7604, 452–455.

[BB19] L. Berk and D. Bertsimas, *Certifiably optimal sparse principal component analysis*, Mathematical Programming Computation **11** (2019), no. 3, 381–420.

[BBM04] S. Bollapragada, M. R. Bussieck, and S. Mallik, *Scheduling commercial videotapes in broadcast television*, Operations Research **52** (2004), no. 5, 679–689.

[BCDSP16] A. Basu, M. Conforti, M. Di Summa, and J. Paat, *Extreme functions with an arbitrary number of slopes*, Integer Programming and Combinatorial Optimization: 18th International Conference, IPCO 2016, Liège, Belgium, June 1–3, 2016, Proceedings (Q. Louveaux and M. Skutella, eds.), Springer International Publishing, Cham, 2016, pp. 190–201.

[BF74] M. B. Brown and A. B. Forsythe, *Robust tests for the equality of variances*, Journal of the American Statistical Association **69** (1974), no. 346, 364–367.

[BGG+12] T. Berthold, G. Gamrath, A. M. Gleixner, S. Heinz, T. Koch, and Y. Shinano, *Solving mixed integer linear and nonlinear problems using the SCIP optimization suite*, ZIB-Report 12-27 (2012), 1–23.

[BHK14] A. Basu, R. Hildebrand, and M. Köppe, *Equivariant perturbation in Gomory and Johnson's infinite group problem. I. The one-dimensional case*, Mathematics of Operations Research **40** (2014), no. 1, 105–129.

[BHK16a] ———, *Light on the infinite group relaxation I: foundations and taxonomy*, 4OR **14** (2016), no. 1, 1–40.

[BHK16b] ———, *Light on the infinite group relaxation II: sufficient conditions for extremality, sequences, and algorithms*, 4OR **14** (2016), no. 2, 107–131.

[BHKM13] A. Basu, R. Hildebrand, M. Köppe, and M. Molinaro, *A $(k+1)$-slope theorem for the k-dimensional infinite group relaxation*, SIAM Journal on Optimization **23** (2013), no. 2, 1021–1040.

[BHM16] A. Basu, R. Hildebrand, and M. Molinaro, *Minimal cut-generating functions are nearly extreme*, Integer Programming and Combinatorial Optimization: 18th International Conference, IPCO 2016, Liège, Belgium, June 1–3, 2016, Proceedings (Q. Louveaux and M. Skutella, eds.), Springer International Publishing, Cham, 2016, pp. 202–213.

[BHT17] L. Bulej, V. Horkỳ, and P. Tůma, *Do we teach useful statistics for performance evaluation?*, Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, ACM, 2017, pp. 185–189.

[BIS16] W. Bruns, B. Ichim, and C. Söger, *The power of pyramid decomposition in normaliz*, J. Symb. Comput. **74** (2016), no. C, 513–536.

[Bix12] R. E. Bixby, *A brief history of linear and mixed-integer programming computation*, Documenta Mathematica (2012), 107–121.

[BJS82] A. Bachem, E. L. Johnson, and R. Schrader, *A characterization of minimal valid inequalities for mixed integer programs*, Operations Research Letters **1** (1982), no. 2, 63–66.

[BK17] M. Bansal and K. Kianfar, *Planar maximum coverage location problem with partial coverage and rectangular demand and service zones*, INFORMS Journal on Computing **29** (2017), no. 1, 152–169.

[Bla78] C. E. Blair, *Minimal inequalities for mixed integer programs*, Discrete Mathematics **24** (1978), no. 2, 147–151.

[BS08] E. Balas and A. Saxena, *Optimizing over the split closure*, Mathematical Programming **113** (2008), no. 2, 219–240.

[CATVS17] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, *EMNIST: Extending MNIST to handwritten letters*, 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 2921–2926.

[CBLA19] D. E. D. Costa, C.-P. Bezemer, P. Leitner, and A. Andrzejak, *What's wrong with my benchmark results? studying bad practices in JMH benchmarks*, IEEE Transactions on Software Engineering (2019).

[CCD+13] M. Conforti, G. Cornuéjols, A. Daniilidis, C. Lemaréchal, and J. Malick, *Cut-generating functions and S-free sets*, Mathematics of Operations Research **40** (2013), no. 2, 253–275.

[Chv80] V. Chvátal, *Hard knapsack problems*, Operations Research **28** (1980), no. 6, 1402–1411.

[CK16] B. Chattopadhyay and K. Kelley, *Estimation of the coefficient of variation with minimum risk: A sequential method for minimizing sampling error and study cost*, Multivariate Behavioral Research **51** (2016), no. 5, 627–648.

[CM+82] Y. Chow, A. Martinsek, et al., *Bounded regret of a sequential procedure for estimation of the mean*, The Annals of Statistics **10** (1982), no. 3, 909–914.

[CP16] C. Collberg and T. A. Proebsting, *Repeatability in computer systems research*, Communications of the ACM **59** (2016), no. 3, 62–69.

[CW20] F. Campelo and E. F. Wanner, *Sample size calculations for the experimental comparison of multiple algorithms on multiple problem instances*, Journal of Heuristics (2020), 1–33.

[CY+81] Y. Chow, K. Yu, et al., *The performance of a sequential procedure for the estimation of the mean*, The Annals of Statistics **9** (1981), no. 1, 184–189.

[DE96] T. J. DiCiccio and B. Efron, *Bootstrap confidence intervals*, Statistical science (1996), 189–212.

[Den12] L. Deng, *The MNIST database of handwritten digit images for machine learning research*, IEEE Signal Processing Magazine **29** (2012), no. 6, 141–142.

[DGB07] W. Dubitzky, M. Granzow, and D. P. Berrar, *Fundamentals of data mining in genomics and proteomics*, Springer Science & Business Media, 2007.

[DGG10] S. Dash, M. Goycoolea, and O. Günlük, *Two-step MIR inequalities for mixed integer programs*, INFORMS Journal on Computing **22** (2010), no. 2, 236–249.

[DGL10] S. Dash, O. Günlük, and A. Lodi, *MIR closures of polyhedral sets*, Mathematical Programming **121** (2010), no. 1, 33–60.

[DM02] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming **91** (2002), no. 2, 201–213.

[DMY92] T. J. DiCiccio, M. A. Martin, and G. A. Young, *Analytical approximations for iterated bootstrap confidence intervals*, Statistics and Computing **2** (1992), no. 3, 161–171.

[DSB17] D. Doran, S. Schulz, and T. R. Besold, *What does explainable AI really mean? a new conceptualization of perspectives*, arXiv preprint arXiv:1710.00794 (2017).

[EPS+18] Z. Epstein, B. H. Payne, J. H. Shen, A. Dubey, B. Felbo, M. Groh, N. Obradovich, M. Cebrián, and I. Rahwan, *Closing the AI knowledge gap*, arXiv preprint arXiv:1803.07233 (2018).

[ESP88] R. L. Edgeman, R. C. Scott, and R. J. Pavur, *A modified Kolmogorov-Smirnov test for the inverse Gaussian density with unknown parameters*, Communications in Statistics-Simulation and Computation **17** (1988), no. 4, 1203–1212.

[Esp08] D. G. Espinoza, *Computing with multi-row gomory cuts*, Proceedings of the 13th international conference on Integer programming and combinatorial optimization (Berlin, Heidelberg), IPCO'08, Springer-Verlag, 2008, pp. 214–224.

[FBS12] J. Freire, P. Bonnet, and D. Shasha, *Computational reproducibility: state-of-the-art, challenges, and database research opportunities*, Proceedings of the 2012 ACM SIGMOD international conference on management of data, 2012, pp. 593–596.

[Fei06] D. G. Feitelson, *Experimental computer science: The need for a cultural change*, Internet version: http://www. cs. huji. ac. il/˜ feit/papers/exp05. pdf (2006).

[FMMS14] M. Ferro, A. R. Mury, L. F. Manfroi, and B. Schlze, *High performance computing evaluation a methodology based on scientific application requirements*, arXiv preprint arXiv:1412.1297 (2014).

[FN95] D. G. Feitelson and B. Nitzberg, *Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860*, workshop on job scheduling strategies for parallel processing, Springer, 1995, pp. 337–360.

[FS04] S. P. Fekete and J. Schepers, *A general framework for bounds for higher-dimensional orthogonal packing problems*, Mathematical Methods of Operations Research **60** (2004), no. 2, 311–329.

[Gan13] C. Gandrud, *Reproducible research with R and R studio*, CRC Press, 2013.

[GBE07] A. Georges, D. Buytaert, and L. Eeckhout, *Statistically rigorous java performance evaluation*, ACM SIGPLAN Notices, vol. 42, ACM, 2007, pp. 57–76.

[GEG+17] A. Gleixner, L. Eifler, T. Gally, G. Gamrath, P. Gemander, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. Miltenberger, et al., *The SCIP optimization suite 5.0*, (2017).

[GJ72a] R. E. Gomory and E. L. Johnson, *Some continuous functions related to corner polyhedra, I*, Mathematical Programming **3** (1972), 23–85.

[GJ72b] ———, *Some continuous functions related to corner polyhedra, II*, Mathematical Programming **3** (1972), 359–389.

[GM79] M. Ghosh and N. Mukhopadhyay, *Sequential point estimation of the mean when the distribution is unspecified*, Communications in Statistics-Theory and Methods **8** (1979), no. 7, 637–652.

[Gom65] R. E. Gomory, *On the relation between integer and noninteger solutions to linear programs*, Proc. Nat. Acad. Sci. U.S.A. **53** (1965), 260–265.

[Gom69a] ———, *Some polyhedra related to combinatorial problems*, Linear Algebra and Appl. **2** (1969), 451–558.

[Gom69b] ———, *Some polyhedra related to combinatorial problems*, Linear Algebra and its Applications **2** (1969), 451–558.

[GS12] M. R. Green and J. Sambrook, *Molecular cloning*, A Laboratory Manual 4th (2012).

[GZ12] A. Ghasemi and S. Zahediasl, *Normality tests for statistical analysis: a guide for non-statisticians*, International journal of endocrinology and metabolism **10** (2012), no. 2, 486.

[HB15] T. Hoefler and R. Belli, *Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results*, Proceedings of the international conference for high performance computing, networking, storage and analysis, ACM, 2015, p. 73.

[HCA15] S. Hunold and A. Carpen-Amarie, *MPI benchmarking revisited: Experimental design and reproducibility*, arXiv preprint arXiv:1505.07734 (2015).

[HKZ18a] R. Hildebrand, M. Köppe, and Y. Zhou, *Equivariant perturbation in Gomory and Johnson's infinite group problem. VII. Inverse semigroup theory, closures, decomposition of perturbations*, e-print, 61 pages, 2018, 1811.06189.

[HKZ18b] ———, *Equivariant perturbation in Gomory and Johnson's infinite group problem. VIII. Grid-free extremality test—general algorithm and implementation*, manuscript, 2018.

[HKZ18c] C. Y. Hong, M. Köppe, and Y. Zhou, *Equivariant perturbation in Gomory and Johnson's infinite group problem (V). Software for the continuous and discontinuous 1-row case*, Optimization Methods and Software **33** (2018), no. 3, 475–498.

[HP88] M. D. Hughes and S. J. Pocock, *Stopping rules and estimation problems in clinical trials*, Statistics in medicine **7** (1988), no. 12, 1231–1242.

[Jer79] R. G. Jeroslow, *Minimal inequalities*, Mathematical Programming **17** (1979), no. 1, 1–15.

[JL84] R. G. Jeroslow and J. K. Lowe, *Modelling with integer variables*, Mathematical Programming at Oberwolfach II, Springer, 1984, pp. 167–184.

[JS11] S. Jukna and G. Schnitger, *Yet harder knapsack problems*, Theoretical computer science **412** (2011), no. 45, 6351–6358.

[KB16] W.-Y. Ku and J. C. Beck, *Mixed integer programming models for job shop scheduling: A computational analysis*, Computers & Operations Research **73** (2016), 165–173.

[KJ13] T. Kalibera and R. Jones, *Rigorous benchmarking in reasonable time*, ACM SIGPLAN Notices **48** (2013), no. 11, 63–74.

[KKY15] F. Kılınç-Karzan and B. Yang, *Sufficient conditions and necessary conditions for the sufficiency of cut-generating functions*, Tech. report, December 2015, `http://www.andrew.cmu.edu/user/fkilinc/files/draft-sufficiency-web.pdf`.

[KL10] S. Kosuch and A. Lisser, *Upper bounds for the 0-1 stochastic knapsack problem and a B&B algorithm*, Annals of Operations Research **176** (2010), no. 1, 77–93.

[KMS+11] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann, *Algorithm selection and scheduling*, International Conference on Principles and Practice of Constraint Programming, Springer, 2011, pp. 454–469.

[Kor85] R. E. Korf, *Depth-first iterative-deepening: An optimal admissible tree search*, Artificial intelligence **27** (1985), no. 1, 97–109.

[KRKP+16] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, et al., *Jupyter notebooks-a publishing format for reproducible computational workflows.*, ELPUB, 2016, pp. 87–90.

[KW19] M. Köppe and J. Wang, *Dual-feasible functions for integer programming and combinatorial optimization: Algorithms, characterizations, and approximations*, Discrete Applied Mathematics (2019).

[KZ15] M. Köppe and Y. Zhou, *An electronic compendium of extreme functions for the Gomory–Johnson infinite group problem*, Operations Research Letters **43** (2015), no. 4, 438–444.

[KZ16] ———, *Toward computer-assisted discovery and automated proofs of cutting plane theorems*, Combinatorial Optimization: 4th International Symposium, ISCO 2016, Vietri sul Mare, Italy, May 16–18, 2016, Revised Selected Papers (R. Cerulli, S. Fujishige, and A. R. Mahjoub, eds.), Springer International Publishing, Cham, 2016, pp. 332–344.

[KZ17] ———, *New computer-based search strategies for extreme functions of the Gomory–Johnson infinite group problem*, Mathematical Programming Computation **9** (2017), no. 3, 419–469.

[KZ18] ———, *Equivariant perturbation in Gomory and Johnson's infinite group problem. VI. The curious case of two-sided discontinuous minimal valid functions*, Discrete Optimization (2018), 1–22.

[KZHW20] M. Köppe, Y. Zhou, C. Y. Hong, and J. Wang, *cutgeneratingfunctionology: Python code for computation and experimentation with cut-generating functions*, `https://github.com/mkoeppe/cutgeneratingfunctionology`, August 2020, Version 1.5.

[Lil67] H. W. Lilliefors, *On the Kolmogorov-Smirnov test for normality with mean and variance unknown*, Journal of the American statistical Association **62** (1967), no. 318, 399–402.

[Lil05] D. J. Lilja, *Measuring computer performance: a practitioner's guide*, Cambridge university press, 2005.

[LJ04] Y. Luo and L. K. John, *Efficiently evaluating speedup using sampled processor simulation*, IEEE Computer Architecture Letters **3** (2004), no. 1, 6–6.

[LK82] A. M. Law and W. D. Kelton, *Confidence intervals for steady-state simulations ii: A survey of sequential procedures*, Management Science **28** (1982), no. 5, 550–562.

[LP03] L. Liberti and C. C. Pantelides, *Convex envelopes of monomials of odd degree*, Journal of Global Optimization **25** (2003), no. 2, 157–168.

[LT13] A. Lodi and A. Tramontani, *Performance variability in mixed-integer programming*, Theory driven by influential applications, INFORMS, 2013, pp. 1–12.

[Lue83] G. S. Lueker, *Bin packing with items uniformly distributed over intervals [a,b]*, Proceedings of the 24th Annual Symposium on Foundations of Computer Science (Washington, DC, USA), SFCS '83, IEEE Computer Society, 1983, pp. 289–297.

[LY95] S. M. Lee and G. A. Young, *Asymptotic iterated bootstrap confidence intervals*, The Annals of Statistics (1995), 1301–1330.

[LY96] ———, *Sequential iterated bootstrap confidence intervals*, Journal of the Royal Statistical Society: Series B (Methodological) **58** (1996), no. 1, 235–251.

[Mar09] F. Margot, *Testing cut generators for mixed-integer linear programming*, Mathematical Programming Computation **1** (2009), no. 1, 69–95.

[Mas04] J. R. Mashey, *War of the benchmark means: time for a truce*, ACM SIGARCH Computer Architecture News **32** (2004), no. 4, 1–14.

[Mey76] R. R. Meyer, *Mixed integer minimization models for piecewise-linear functions of a single variable*, Discrete Mathematics **16** (1976), no. 2, 163–171.

[MF17] C. Moreno and S. Fischmeister, *Accurate measurement of small execution times–getting around measurement errors*, IEEE Embedded Systems Letters **9** (2017), 17–20.

[MGR12] Y. Merzifonluoğlu, J. Geunes, and H. E. Romeijn, *The static stochastic knapsack problem with normally distributed item sizes*, Mathematical Programming **134** (2012), no. 2, 459–489.

[mip18] *MIPLIB 2017*, 2018, http://miplib.zib.de.

[Mit20] H. D. Mittelmann, *Benchmarking optimization software-a (hi) story*, SN Operations Research Forum, vol. 1, Springer, 2020, pp. 1–6.

[MJ51] F. J. Massey Jr, *The Kolmogorov-Smirnov test for goodness of fit*, Journal of the American statistical Association **46** (1951), no. 253, 68–78.

[Mon17] D. C. Montgomery, *Design and analysis of experiments*, John wiley & sons, 2017.

[MWA06] B. Moghaddam, Y. Weiss, and S. Avidan, *Spectral bounds for sparse PCA: Exact and greedy algorithms*, Advances in neural information processing systems, 2006, pp. 915–922.

[Nag80] H. Nagao, *On stopping times of sequential estimations of the mean of a log-normal distribution*, Annals of the Institute of Statistical Mathematics **32** (1980), no. 3, 369–375.

[Nem13] G. L. Nemhauser, *Integer programming: The global impact*, (2013).

[NS04] A. Neumaier and O. Shcherbina, *Safe bounds in linear and mixed-integer linear programming*, Mathematical Programming **99** (2004), no. 2, 283–296.

[OLCO+17] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, *PMLB: a large benchmark suite for machine learning evaluation and comparison*, BioData mining **10** (2017), no. 1, 1–13.

[Ols05] U. Olsson, *Confidence intervals for the mean of a log-normal distribution*, Journal of Statistics Education **13** (2005), no. 1.

[PBV08] P. M. Pardalos, V. L. Boginski, and A. Vazacopoulos, *Data mining in biomedicine*, vol. 7, Springer Science & Business Media, 2008.

[Pen11] R. D. Peng, *Reproducible research in computational science*, Science **334** (2011), no. 6060, 1226–1227.

[Per18] J. M. Perkel, *Data visualization tools drive interactivity and reproducibility in online publishing*, Nature **554** (2018), no. 7690, 133–134.

[Poi12] L. Poirrier, *Multi-row approaches to cutting plane generation*, Ph.D. thesis, University of Liege, Belgium (2012).

[PT09] D. Payne and C. C. Trumbach, *Data mining: proprietary rights, people and proposals*, Business Ethics: A European Review **18** (2009), no. 3, 241–252.

[PVB+19] A. V. Papadopoulos, L. Versluis, A. Bauer, N. Herbst, J. Von Kistowski, A. Ali-eldin, C. Abad, J. N. Amaral, P. Tůma, and A. Iosup, *Methodological principles for reproducible performance evaluation in cloud computing*, IEEE Transactions on Software Engineering (2019).

[RACC12] J. Rietz, C. Alves, J. Carvalho, and F. Clautiaux, *Computing valid inequalities for general integer programs using an extension of maximal dual feasible functions to negative arguments*, 1st International Conference on Operations Research and Enterprise Systems-ICORES 2012, 2012.

[RAdCC14] J. Rietz, C. Alves, J. M. V. de Carvalho, and F. Clautiaux, *On the properties of general dual-feasible functions*, pp. 180–194, Springer International Publishing, Cham, 2014.

[Ram13] K. Ram, *Git can facilitate greater reproducibility and increased transparency in science*, Source code for biology and medicine **8** (2013), no. 1, 1–8.

[RG04] D. Rasch and V. Guiard, *The robustness of parametric statistical methods*, Psychology Science **46** (2004), 175–208.

[RHGM18] M. Raasveldt, P. Holanda, T. Gubner, and H. Mühleisen, *Fair benchmarking considered difficult: Common pitfalls in database performance testing*, Proceedings of the Workshop on Testing Database Systems, ACM, 2018, p. 2.

[Rud19] C. Rudin, *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*, Nature Machine Intelligence **1** (2019), no. 5, 206–215.

[S+16] W. A. Stein et al., *Sage Mathematics Software (Version 7.1)*, The Sage Development Team, 2016, `http://www.sagemath.org`.

[SBO+07] S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.-R. MÃžller, F. Pereira, C. E. Rasmussen, et al., *The need for open source software in machine learning*, Journal of Machine Learning Research **8** (2007), no. Oct, 2443–2466.

[SBZ+08] M. Sayeed, H. Bae, Y. Zheng, B. Armstrong, R. Eigenmann, and G. Saied, *Measuring high-performance computing with real applications*, Computing in Science & Engineering **10** (2008), no. 4, 60.

[Sin14] D. I. Singham, *Selecting stopping rules for confidence interval procedures*, ACM Transactions on Modeling and Computer Simulation (TOMACS) **24** (2014), no. 3, 1–18.

[SP99] E. Smith and C. Pantelides, *A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs*, Computers & Chemical Engineering **23** (1999), no. 4, 457 – 478.

[Sta66] N. Starr, *On the asymptotic efficiency of a sequential procedure for estimating the mean*, The Annals of Mathematical Statistics **37** (1966), no. 5, 1173–1185.

[SW72] N. Starr and M. Woodroofe, *Further remarks on sequential estimation: the exponential case*, The Annals of Mathematical Statistics (1972), 1147–1154.

[TS05] M. Tawarmalani and N. V. Sahinidis, *A polyhedral branch-and-cut approach to global optimization*, Mathematical Programming **103** (2005), no. 2, 225–249.

[TSS02] M. Tawarmalani, N. V. Sahinidis, and N. Sahinidis, *Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications*, vol. 65, Springer Science & Business Media, 2002.

[TSW10] A. Tafazzoli, N. M. Steiger, and J. R. Wilson, *N-Skart: A nonsequential skewness-and autoregression-adjusted batch-means procedure for simulation analysis*, IEEE Transactions on Automatic Control **56** (2010), no. 2, 254–264.

[VAB+97] P. H. Vance, A. Atamturk, C. Barnhart, E. Gelman, E. L. Johnson, A. Krishna, D. Mahidhara, G. L. Nemhauser, and R. Rebello, *A heuristic branch-and-price approach for the airline crew pairing problem*, preprint (1997).

[Van00] F. Vanderbeck, *Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem*, Operations Research **48** (2000), no. 6, 915–926.

[Var79] Y. Vardi, *Asymptotic optimal sequential estimation: the Poisson case*, The Annals of Statistics (1979), 1040–1051.

[Vie15] J. P. Vielma, *Mixed integer linear programming formulation techniques*, Siam Review **57** (2015), no. 1, 3–57.

[VK12] J. Vitek and T. Kalibera, *R3: Repeatability, reproducibility and rigor*, ACM SIGPLAN Notices **47** (2012), no. 4a, 30–36.

[Voo17] P. Voosen, *The AI detectives*, 2017.

[Wel62] B. Welford, *Note on a method for calculating corrected sums of squares and products*, Technometrics **4** (1962), no. 3, 419–420.

[WHG18] A. Wang, C. L. Hanselman, and C. E. Gounaris, *A customized branch-and-bound approach for irregular shape nesting*, Journal of Global Optimization **71** (2018), no. 4, 935–955.

[WMDC83] B. W. Woodruff, A. H. Moore, E. J. Dunne, and R. Cortes, *A modified Kolmogorov-Smirnov test for Weibull distributions with unknown location and scale parameters*, IEEE transactions on Reliability **32** (1983), no. 2, 209–213.

[YC16] S. Yıldız and G. Cornuéjols, *Cut-generating functions for integer variables*, Mathematics of Operations Research **41** (2016), no. 4, 1381–1403, http://dx.doi.org/10.1287/moor.2016.0781.

[YH19] A. Yousef and H. Hamdy, *Three-stage estimation of the mean and variance of the normal distribution with application to an inverse coefficient of variation with computer simulation*, Mathematics **7** (2019), no. 9, 831.

[Zac66] S. Zacks, *Sequential estimation of the mean of a log-normal distribution having a prescribed proportional closeness*, The Annals of Mathematical Statistics (1966), 1688–1696.

[Zam09] G. Zambelli, *On degenerate multi-row Gomory cuts*, Operations Research Letters **37** (2009), no. 1, 21–22.

[Zho17] Y. Zhou, *Infinite-dimensional relaxations of mixed-integer optimization problems*, Ph.D. thesis, University of California, Davis, Graduate Group in Applied Mathematics, May 2017.