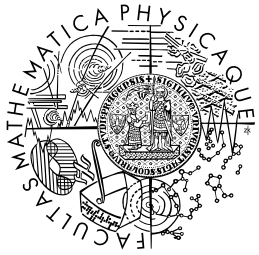


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Jakub Komárek

Experimental Analysis of Scaling Methods for LP

Computer Science Institute of Charles University

Supervisor of the bachelor thesis: Mgr. Martin Koutecký, Ph.D.
Study programme: Computer Science
Study branch: General Computer Science

Prague 2023

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my thesis supervisor Mgr. Martin Koutecký, Ph.D. for providing much valuable advice and support. Among all the other people who supported me, I am especially grateful for my girlfriend Maky for her understanding and helping attitude.

Title: Experimental Analysis of Scaling Methods for LP

Author: Jakub Komárek

Institute: Computer Science Institute of Charles University

Supervisor: Mgr. Martin Koutecký, Ph.D., Computer Science Institute of Charles University

Abstract: In his recent work, Dadush et al. introduced the condition number κ for constraint matrices of linear programming and devised an algorithm to approximately optimize κ by scaling columns of the constraint matrix. We follow up on his work by implementing this scaling algorithm. We have used our implementation to obtain an approximate rescaling of some linear programming instances available from public datasets. Finally, this work shows results of experiments evaluating the effects of the obtained rescalings on the runtime of some available linear programming solvers.

Keywords: linear programming scaling circuit imbalance measure

Contents

1. Introduction

Linear programming is a well-established discipline in optimization and modern computer science with many applications in practice. In one of his articles [Bix12], Bixby describes the immense theoretical advancements in algorithms for solving linear programs since its early years, far outreaching the advancements of the machines running them. We are therefore well-aware of the usefulness of theoretical research in this area and we strive to contribute to its advancement.

One of the possible approaches at improving linear programming algorithms is the use of scaling. Dadush et al. [Dad+20] have shown that the complexity of an LP solving algorithm of Vavasis and Ye [VY96] can be bounded in terms of a certain condition number κ of the constraint matrix A. This condition number can be changed by scaling individual columns of A by positive real values. The authors have further devised an algorithm to find a positive diagonal matrix D such that the value κ after applying such rescaling is not too far from the minimum one. To be precise, let κ^* be the minimum κ that can be obtained by applying a column rescaling. Then, applying the rescaling matrix D returned by the algorithm yields a condition number $\kappa \leq (\kappa^*)^3$.

This thesis follows up on these results. In contrast to showing theoretical improvements of the algorithm, we are interested in exploring its applicability in practice and describing the effects of the condition number on performance of practical LP solvers, including ones not based on interior-point methods.

1.1 Scaling

In an effort to avoid numerical instability issues when solving LPs on realistic hardware, one may try to change the scaling of columns of the problem matrix. For example, if values in a column represent very large distances in meters, it might be useful to scale down the values, e.g., represent the distances in kilometers.

Scaling methods have been previously studied, e.g., by Tomlin [Tom75] and are commonly used by practical LP solvers. In this thesis we are, however, exploring a less traditional approach to scaling presented in the article by Dadush et al. [Dad+20] and clearly described in later sections.

Note that while scaling rows of the problem matrix is also possible, it will not be the focus of this thesis. We will therefore restrict ourselves only to scaling of columns.

1.2 The Circuit Imbalance Measure

This thesis revolves around the matrix condition numbers κ and κ^*, as defined in the article by Dadush et al. [Dad+20]. These numbers play a crucial role in both implementation and analysis of their scaling-invariant algorithm for linear programming whose running time depends only on the constraint matrix. For a comprehensive overview of the algorithm see the full article by Dadush et al. [Dad+20]. Specifically, the procedure for approximating κ^* described in Section 2.4 was originally devised as a subroutine of this linear programming

algorithm and this algorithm remains our main motivation for research in this area.

As the name suggests, the circuit imbalance measure relies on the concept of *circuits* of a vector matroid. For reader not familiar with these notions, we refer to Section 2.3, where some background and notation is given.

For every circuit C of the vector matroid of A we can consider its *elementary vector*, which is an element of the kernel of A that has C as its support. The *circuit imbalance measure* κ of matrix A is the maximum ratio of absolute values of nonzero elements over all elementary vectors of the vector matroid of A.

If we scale the columns of A, the set of circuits $\mathcal{C}$ does not change. However, the elementary vectors do change and so κ can also change. If we view κ as a measure of how ill-conditioned the matrix is, it makes sense to ask what is the best value of κ we can obtain by choosing a suitable column scaling. This is exactly what the number κ^* means.

1.3 Our Contribution

Our first contribution is an implementation of the algorithm for computing an approximate optimal rescaling with respect to the circuit imbalance measure. To the best of my knowledge, this is the first implementation of the algorithm. We have discovered several issues that complicate implementing the algorithm in practice and that are not apparent from the theoretical description. We try to address these issues and propose reasonable solutions.

Secondly, we used the algorithm implementation to obtain an approximate optimal rescaling of many problems in the benchmark sets Netlib [Gay85] and MIPLIB [Gle+21] (here we consider the linear relaxations, since the scaling approach is only applicable for purely continuous problems). Because of the high complexity of the algorithm and size of the mentioned problems there remain problem instances for which the computation of the rescaling using our implementation would need more computational resources than what we deemed as reasonable for the purposes of this thesis.

As our third contribution, we used some available LP solvers to evaluate the impact of the computed rescalings on the time it takes the solvers to solve the instances. We have used the solvers GLPK[1], Gurobi [Gur23] and SCIP [Bes+21] using all available solving methods and using both exact and inexact arithmetic where available.

[1]Open source software available at `http://www.gnu.org/software/glpk/glpk.html`

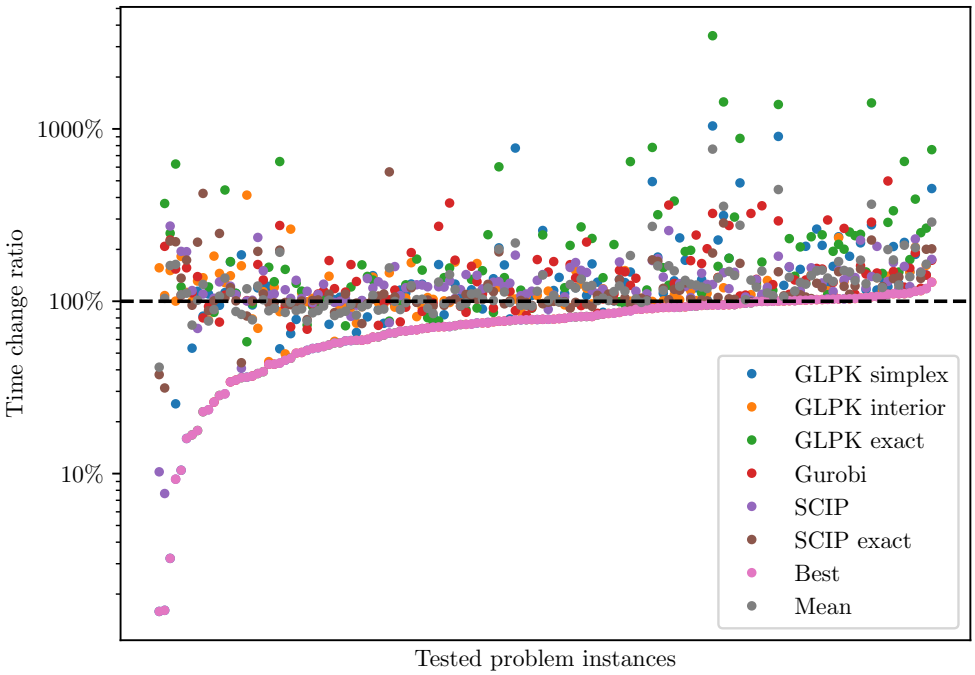


Figure 1.1: Results of the experiment.

2. Preliminaries

In this chapter we provide the necessary background and definitions used throughout the rest of the thesis. First, we introduce the notation conventions this thesis follows (Section 2.1). Next, we include the background on *linear programming* (Section 2.2), *matroid theory* (Section 2.3) and the algorithm for optimizing the circuit imbalance measure (Section 2.4), which is central to this thesis.

2.1 Notation

We denote the sets of all real and natural numbers respectively by $\mathbb{R}$ and $\mathbb{N}$. As is customary in computer science, we consider zero to be a natural number.

Matrices are written as $A \in \mathbb{R}^{m \times n}$ and vectors as $\mathbf{x} \in \mathbb{R}^n$. All vectors are regarded as column vectors, the notation $\mathbf{x}^\top$ is used to denote row vectors when needed. Vector elements are provided with indices and not written in bold font, as in $\mathbf{x} = (x_1, x_2, \ldots, x_n)^\top$. Whenever used with the index in bold font, $\mathbf{x_i}$ does *not* denote the i-th coordinate, but it is a vector itself. The notation $\mathrm{Diag}(\mathbf{d})$ means a diagonal matrix with elements of $\mathbf{d}$ on the diagonal.

When using the $\leq$ relation and derived relations ($\geq$, $<$, $>$, $=$) on vectors, it is understood to mean that the relation holds for every coordinate of the vectors. More precisely, if $\mathbf{x}$ and $\mathbf{y}$ are vectors of the same dimension, the notation $\mathbf{x} \leq \mathbf{y}$ means that $x_i \leq y_i$ for every coordinate i.

Sets are denoted with capital letters. We use the union symbol strictly for the union of the sets, e.g., $A \cup B$. When expressing a set with an extra added element, we use the notation $A \cup \{e\}$. For set difference we use the notation $A \setminus B$. We use notation $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

If $\mathbb{F}$ is arbitrary field, then the coordinate space whose elements are n-tuples of elements of $\mathbb{F}$ is denoted $\mathbb{F}^n$.

2.2 Linear Programming

In this section, we introduce the notion of *linear programming*. This notion is central to this thesis, as it provides the framework for our further results.

Definition 1 (Linear program in standard form)**.** *Let $A \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^n$. Then a linear program in standard form (LP) is the problem*

$$\begin{aligned} \min\ & \mathbf{c}^\top \mathbf{x} \\ A\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0}. \end{aligned}$$

Solving LP problems is an extensively studied subject in the field of computer science and many different algorithms have been proposed to tackle it. A particularly convenient and widely known fact is that LPs can be solved in polynomial time, as has been first shown by Khachiyan [Kha79].

For a more in-depth overview of LP we refer the reader to the textbook of Schrijver [Sch99].

2.2.1 Equivalent Forms

The standard LP form is convenient for researching and describing algorithms for solving LP's, because it gives a simple description of the constraints. However, for formulating real-world problems as LP's it may be easier to describe the problem using a different form that allows to express real-world constraints more naturally. We introduce a modelling form that is sometimes used in practical software packages for working with LP's.

Definition 2 (Modelling form)**.** *Let* $A_{\text{ub}} \in \mathbb{R}^{m_1 \times n}$*,* $A_{\text{eq}} \in \mathbb{R}^{m_2 \times n}$*,* $\mathbf{b_{ub}} \in \mathbb{R}^{m_1}$*,* $\mathbf{b_{eq}} \in \mathbb{R}^{m_2}$*,* $\mathbf{c} \in \mathbb{R}^n$ *and* $\mathbf{l}, \mathbf{u} \in (\mathbb{R} \cup \{-\infty, \infty\})^n$ *such that* $\mathbf{l} \leq \mathbf{u}$*. Then, a linear program in modelling form is the problem*

$$\begin{aligned} \min\ & \mathbf{c}^\top \mathbf{x} \\ A_{\text{ub}}\mathbf{x} &\leq \mathbf{b_{ub}} \\ A_{\text{eq}}\mathbf{x} &= \mathbf{b_{eq}} \\ \mathbf{l} \leq \mathbf{x} &\leq \mathbf{u}. \end{aligned}$$

It is not difficult to see that every LP in standard form is already in modelling form. What is, however, less clear, is that also every LP in modelling form can be efficiently converted to an equivalent LP in standard form, rendering the two forms easily interchangeable.

Proposition 1. *Solving an LP in modelling form is reducible to solving an LP in standard form in time* $\mathcal{O}(m_1 n + m_2 n)$ *where* m_1, m_2, n *have meanings as in Definition 2.*

This proposition can be proven using standard LP form reduction techniques. For examples of such techniques, see e.g., the textbook of Gärtner and Matoušek [GM07, Section 4.1].

2.2.2 Solving Scaled Instances

When constructing a scaled LP instance, it is required that we can obtain an optimum of the original program from an optimum of a scaled instance. The following proposition implies that this is in fact possible.

Proposition 2. *Let* $A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^n$ *and let* $D \in \mathbb{R}^{n \times n}$ *be a diagonal matrix with positive entries on the diagonal. Let* P *be the standard-form LP*

$$\begin{aligned} \min\ & \mathbf{c}^\top \mathbf{x} \\ A\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

and Q *the standard-form LP*

$$\begin{aligned} \min\ & \mathbf{c}^\top D\mathbf{x} \\ (AD)\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0}. \end{aligned}$$

Then $\mathbf{x}'$ *is an optimum of* P *if and only if* $D^{-1}\mathbf{x}'$ *is an optimum of* Q*.*

Proof. Let $\mathbf{x}'$ be an optimum of P. Because D has positive diagonal entries, D^{-1} also has positive diagonal entries, so $D^{-1}\mathbf{x}' \geq \mathbf{0}$ holds. The equality constraint is also satisfied, which follows from $(AD)(D^{-1}\mathbf{x}') = A(DD^{-1})\mathbf{x}' = A\mathbf{x}' = \mathbf{b}$. This proves that $D^{-1}\mathbf{x}'$ is feasible for Q.

What is left to show is the optimality of $D^{-1}\mathbf{x}'$ for Q. For contradiction consider that Q has a better solution $\mathbf{x}''$ such that $\mathbf{c}^\top D\mathbf{x}'' < \mathbf{c}^\top D(D^{-1}\mathbf{x}')$. Then, $D\mathbf{x}''$ is feasible for P and it holds that $\mathbf{c}^\top(D\mathbf{x}'') < \mathbf{c}^\top\mathbf{x}'$, contradicting that $\mathbf{x}'$ is an optimum of P.

The proof of the opposite direction is symmetric. □

2.3 Matroids

While it is possible to describe the objective our scaling algorithm is optimizing using notions from elementary linear algebra, it is best described using the notion of circuits from matroid theory. This is a well-studied entity and several results regarding circuits together with other results from matroid theory are useful in understanding and implementing the rescaling algorithm. This section strives to introduce such results to the reader. Unless stated otherwise, the definitions and results closely follow the book by Oxley [Oxl92].

Definition 3 (Matroid)**.** *A* matroid M *is an ordered pair* (E, I) *consisting of a finite set* E *and a collection* $\mathcal{I}$ *of subsets of* E *satisfying the following three conditions:*

(I1) $\emptyset \in \mathcal{I}$.

(I2) *If* $I \in \mathcal{I}$ *and* $I' \subseteq I$*, then* $I' \in \mathcal{I}$*. (Hereditary property)*

(I3) *If* I_1 *and* I_2 *are in* $\mathcal{I}$ *and* $|I_1| < |I_2|$*, then there is an element* e *of* $I_2 - I_1$ *such that* $I_1 \cup \{e\} \in \mathcal{I}$*. (Augmentation property)*

The members of $\mathcal{I}$ *are the* independent sets *of* M *and* E *is the* ground set *of* M*. A subset of* E *that is not in* $\mathcal{I}$ *is called* dependent*.*

Notation. We use the notation $E(M)$ and $\mathcal{I}(M)$ to refer to elements and independent sets of M. Where the matroid M is clear from context, we only use the symbols E and $\mathcal{I}$.

The only matroid instance we interest ourselves in the scope of this thesis is the *vector matroid* as defined below.

Definition 4 (Vector matroid)**.** *Let* $\mathbb{F}$ *be an arbitrary field and let* $A \in \mathbb{F}^{m\times n}$*. Let* E *be the set of column labels of* A*, and let* $\mathcal{I}$ *be the set of subsets* X *of* E *for which the multiset of columns labeled by* X *is linearly independent in the vector space* $\mathbb{F}^m$*. Then* $(E, \mathcal{I})$ *is called the* vector matroid *of* A *and is denoted by* $M(A)$*.*

The use of the term vector *matroid* is justified by the following proposition.

Proposition 3 (Vector matroid is a matroid [Oxl92, Proposition 1.1.1])**.** *For any field* $\mathbb{F}$ *and any matrix* $A \in \mathbb{F}^{m\times n}$*, the vector matroid of* A *is a matroid.*

The fact that the matroid augmentation property holds for vector spaces is a famous result in linear algebra known as the Steinitz Exchange Theorem. We state it here for completeness, for the proof the reader may see the referenced book.

Theorem 4 (Steinitz exchange theorem [Jän94, Section 3.6])**.** *If a vector space V has a basis of p vectors and if $\mathbf{v_1}, \ldots, \mathbf{v_r}$ are linearly independent in V, then there exists a basis of p vectors for V, in which $\mathbf{v_1}, \ldots, \mathbf{v_r}$ all occur.*

It is now time to introduce some definitions used throughout this thesis, namely those of a matroid circuit and fundamental circuit.

Definition 5 (Circuit)**.** *A set $C \subseteq E(M)$ is a* circuit *if and only if C is a minimal dependent set of M. We shall denote the set of circuits of M by $\mathcal{C}(M)$.*

Definition 6 (Basis)**.** *Any maximal independent set in a matroid M is a* basis *of M.*

Proposition 5 (Fundamental circuit)**.** *Let B be a basis of a matroid M. If $e \in E(M) \setminus B$, then $B \cup \{e\}$ contains a unique circuit, denoted by $C(e, B)$. Moreover, $e \in C(e, B)$. We call this circuit the* fundamental circuit *of e with respect to B.*

The set of all fundamental circuits of M with respect to the basis B will be denoted by $\mathcal{F}_B(M)$.

Notation. Where the matroid is clear from context, we only use the symbols $\mathcal{C}$ and $\mathcal{F}_B$ instead of $\mathcal{C}(M)$ and $\mathcal{F}_B(M)$.

For the proof of Proposition 5 we will find handy the following result:

Proposition 6 ([Oxl92, Lemma 1.1.3])**.** *The set $\mathcal{C}$ of circuits of a matroid has the following property:*

If C_1 and C_2 are distinct members of $\mathcal{C}$ and $e \in C_1 \cap C_2$, then there is a member C_3 of $\mathcal{C}$ such that $C_3 \subseteq (C_1 \cup C_2) \setminus \{e\}$.

We are now set to present the proof of Proposition 5.

Proof of Proposition 5. Given that B is maximal independent and e is not in B, the set $B \cup \{e\}$ is dependent. There must then be a subset $C \subseteq B \cup \{e\}$ which is inclusion-minimal dependent and is therefore a circuit. Because B is independent, all its subsets are also independent thanks to the matroid hereditary property and therefore all dependent subsets of $B \cup \{e\}$ must contain e. That in turn implies $e \in C$.

It is left to show that this circuit is unique. Consider that there exists another circuit $C' \subseteq B \cup \{e\}$. Applying the same argument as above, it must hold that $e \in C'$. By Proposition 6 there exists a circuit C'' s.t. $C'' \subseteq (C \cup C') \setminus \{e\} \subseteq B$, giving us a dependent subset of B and thus contradicting that B is a basis. □

Another notion to define is matroid connectivity and components. Prior to introducing these definitions, for each element e of a matroid M let $\gamma(e)$ be the set

$$\gamma(e) := \{e\} \cup \{f \in E(M) \mid M \text{ has a circuit containing both } e \text{ and } f\}.$$

Define the relation γ on E by $e \ \gamma \ f$ if and only if $e \in \gamma(f)$. Without proof we claim that this relation is an equivalence, for a proof refer to the book by Oxley [Oxl92, Proposition 4.1.2].

It is now time to properly define the notions introduced in Section 1.2. In the article by Dadush et al. [Dad+20], some of the following notions are expressed in terms of a linear subspace. To keep things simple, we modify the definitions to work directly with matrix A.

Definition 7 (Elementary vector)**.** *An elementary vector for a circuit $C \in \mathcal{C}(M(A))$ is a vector $\mathbf{g^C} \in \mathrm{Ker}(A)$ such that $\mathrm{supp}(\mathbf{g^C}) = C$.*

Note that the elementary vector is unique up to scalar multiplication.

Prior to defining κ itself, we first introduce pairwise circuit imbalance measures κ_{ij}. These will not only come in handy when defining κ itself, but they also play a key role in the procedure in Section 2.4.

Definition 8 (Pairwise circuit imbalance measure [Dad+20, Definition 2.7])**.** *The pairwise circuit imbalance measure for coordinate pair $i, j \in [n], i \neq j$ is the value*

$$\kappa_{ij}^A(C) := \frac{\left|g_j^C\right|}{|g_i^C|}, \quad \kappa_{ij}^A := \max\{\kappa_{ij}^A(C) \mid C \in \mathcal{C}(A), i, j \in C\}.$$

By convention we set $\kappa_{ij} = 0$ if there is no circuit supporting i and j.

Definition 9 (Circuit imbalance measure)**.** *The circuit imbalance measure is defined as*

$$\kappa_A := \max\{\kappa_{ij}^A \mid i, j \in [n]\}.$$

Notation. For convenience, we will also be using the notation

$$\kappa_A(C) := \max\{\kappa_{ij}^A(C) \mid i, j \in C\}$$

that satisfies that $\kappa_A = \max\{\kappa_A(C) \mid C \in \mathcal{C}(M(A))\}$.

Definition 10. *The set $\mathbf{D}$ is the set of $n \times n$ real diagonal matrices with positive entries on the diagonal.*

Definition 11 (Optimal circuit imbalance measure)**.** *The* optimal circuit imbalance measure *for A is the number*

$$\kappa_A^* := \inf\{\kappa_{AD} \mid D \in \mathbf{D}\}.$$

It is not a priori clear that the infimum is indeed attained. Proposition 7 gives justification that it is (without proof).

Proposition 7. *There exists $D \in \mathbf{D}$ such that $\kappa_A^* = \kappa_{AD}$. Consequently,*

$$\kappa_A^* = \min\{\kappa_{AD} \mid D \in \mathbf{D}\}.$$

Notation. If clear from context, we will sometimes omit the A subscript or superscript from κ and its variants.

Let us now introduce some more background from matroid theory.

Definition 12 (Matroid components)**.** *For a matroid M, its* components *are the equivalence classes of the relation γ on $E(M)$.*

Definition 13 (Matroid connectivity)**.** *A matroid is* connected *if and only if it has a single component.*

Let us explicitly state a simple corollary that will be later very useful for us.

Corollary 8. *The matroid M is connected if and only if, for every pair of distinct elements of $E(M)$, there is a circuit containing both.*

Matroid connectivity can be also characterized by connectivity of its fundamental circuit graph.

Definition 14 (Fundamental circuit graph)**.** *Let B be a basis of a matroid M. The* fundamental circuit graph with respect to the basis B *is the undirected graph $\mathcal{F}G_B = (V, E)$ defined as $V := [n]$ and*

$$E := \left\{ \{i,j\} \in \binom{[n]}{2} \,\middle|\, i \text{ and } j \text{ are both contained in some fundamental circuit} \right\}.$$

In their article [Dad+20, Proposition 2.19], Dadush et al. give several characterizations of matroid separability. The two following results straightforwardly follow from these characterizations.

Proposition 9. *A matroid M is connected if and only if its fundamental circuit graph $\mathcal{F}G_B$ is connected for any choice of the basis B.*

Lemma 10. *The connected components of $\mathcal{F}G_B$ correspond to components of M.*

Again omitting a technical proof which the reader can find in the article by Dadush et al. [Dad+20], we present one more important proposition revolving around circuits.

Proposition 11 ([Dad+20, Lemma 2.21])**.** *Let B be a basis of the matroid $M = ([n], \mathcal{I})$, and let $U = \{u_1, u_2, \ldots, u_\ell\} \subseteq B$, and $V = \{v_1, v_2, \ldots, v_\ell, v_{\ell+1}\} \subseteq [n] \setminus B$. Assume $C(B, v_1) \cap U = \{u_1\}$, $C(B, v_{\ell+1}) \cap U = \{u_\ell\}$, and for each $2 \leq t \leq \ell$, $C(B, v_t) \cap U = \{u_{t-1}, u_t\}$. Then, $(B \setminus U) \cup V$ contains a unique circuit C, and $V \subseteq C$.*

2.4 Algorithm for Optimizing the Circuit Imbalance Measure

In this section, we show a way to approximate κ and give an overview of the scaling algorithm from the article by Dadush et al. [Dad+20] that is implemented and used in experiments in later chapters. The purpose of the algorithm is to find the diagonal matrix D such that κ_{AD} is not much larger that κ_A^*.

First, we leverage Corollary 8 to restrict ourselves to matrices whose vector matroid is connected.

Notation. For $B \subseteq [n]$, we will let A_B denote the submatrix of A obtained by selecting exactly columns with indices in B and similarly we will let $\mathbf{v}_B$ denote the subvector of $\mathbf{v}$ obtained by selecting exactly the entries of $\mathbf{v}$ with indices in B.

Proposition 12. *Let $A \in \mathbb{R}^{m\times n}$ and let $V_1, V_2, \ldots, V_k$ be the components of $M(A)$. Then, it holds that*

$$\kappa_A = \max\{\kappa_{A_{V_\ell}} \mid \ell \in [k]\}. \tag{2.1}$$

Proof. We will show a bijection $f : \mathcal{C}(M(A)) \to \bigcup_{\ell\in[k]} \mathcal{C}(M(A_{V_\ell}))$ for which for every circuit C of A in component V_ℓ and every $i, j \in C$ holds that

$$\kappa_A(C) = \kappa_{A_{V_\ell}}(f(C)).$$

That will yield (2.1) because

$$\begin{aligned}
\kappa_A &= \max\{\kappa^A_{ij} \mid i, j \in [n]\} \\
&= \max\{\kappa^A_{ij}(C) \mid C \in \mathcal{C}(A), i, j \in C\} \\
&= \max\{\kappa_A(C) \mid C \in \mathcal{C}(A)\} \\
&= \max\{\kappa_{A_{V_\ell}}(f(C)) \mid C \in \mathcal{C}(A), \ell \in [k]\} \\
&= \max\{\max\{\kappa_{A_{V_\ell}}(C) \mid C \in \mathcal{C}(A_{V_\ell})\} \mid \ell \in [k]\} \\
&= \max\{\kappa_{A_{V_\ell}} \mid \ell \in [k]\}.
\end{aligned}$$

Now, it suffices to show the bijection f. Let us choose an arbitrary circuit C with its elementary vector $\mathbf{g}^{\mathbf{C}}$. By Corollary 8, C lies inside a single component V_ℓ. Then, $\mathbf{g}^{\mathbf{C}}_{V_\ell} \in \mathrm{Ker}(A_{V_\ell})$ is a vector corresponding to a circuit in $\mathcal{C}(M(A_{V_\ell}))$. We therefore define

$$f(C) := \mathrm{supp}(\mathbf{g}^{\mathbf{C}}_{V_\ell}).$$

As $\mathbf{g}^{\mathbf{C}}$ and $\mathbf{g}^{\mathbf{C}}_{V_\ell}$ have the same non-zero entries, it must hold

$$\{\kappa^A_{ij}(C) \mid i, j \in C\} \setminus \{0\} = \{\kappa^{A_{V_\ell}}_{ij}(f(C)) \mid i, j \in f(C)\}$$

and consequently also

$$\kappa_A(C) = \kappa_{A_{V_\ell}}(f(C)).$$

The very last remaining thing is to show that f is indeed bijective. The mapping is clearly injective, as vectors corresponding to different vectors have different supports. Now choose any $\ell \in [k]$ and $C \in \mathcal{C}(M(A_{V_\ell}))$ and let $\mathbf{g} \in \mathrm{Ker}(A_{V_\ell})$ be the corresponding vector. By "lifting" the vector back to the original $\mathbb{R}^n$ space, we get a vector $\mathbf{h} \in \mathrm{Ker}(A)$ such that $f(\mathbf{h}) = \mathbf{g}$. Hence, the mapping is also surjective. □

As a corollary, this is not only useful for obtaining the value of κ_A from results for matrices of individual components, but also for finding a good rescaling from rescalings of individual components.

Corollary 13. *Let $D_1 \in \mathbb{R}^{|V_1|\times|V_1|}, D_2 \in \mathbb{R}^{|V_2|\times|V_2|}, \ldots, D_k \in \mathbb{R}^{|V_k|\times|V_k|}$ be rescaling matrices for individual components. Let $\hat{D}_1, \hat{D}_2, \ldots, \hat{D}_2 \in \mathbb{R}^{n\times n}$ be the rescaling matrices lifted back to the space $\mathbb{R}^{n\times n}$. More precisely, let $\gamma(\alpha)$ be the element of $M(A)$ corresponding to $\alpha \in V_i$. Then, let*

$$(\hat{D}_i)_{\gamma(\alpha)\gamma(\beta)} := (D_i)_{\alpha\beta} \quad \forall \alpha, \beta \in V_i$$

and let all the undefined values be zero.

Now combine the lifted matrices into a single matrix D. Since at most one of $\hat{D}_i$ has a non-zero value at any position α, β, this can be expressed simply as

$$D := \sum_{i \in [k]} \hat{D}_i.$$

Then, it holds that

$$\kappa_{AD} = \max\{\kappa_{A_{V_\ell}} \mid \ell \in [k]\}.$$

In the rest of this section we can therefore assume for $M(A)$ to be connected.

As noted in the article by Dadush et al. [Dad+20], it follows from an article by Tuncel [Tun99] that approximating κ up to a factor of $2^{\mathcal{O}(m)}$ is NP-hard. The algorithm provided in the following text offers to find a $\hat{\kappa}_A$ satisfying

$$\hat{\kappa}_A \leq \kappa_A \leq n(\kappa_A^*)^2 \hat{\kappa}_A$$

in polynomial time. Note that this is not a contradiction to the NP-hardness claim, because the bound depends on κ_A and $\hat{\kappa}_A$, which can both be arbitrarily large.

The idea of approximating κ is to first show that when possessing the values κ_{ij}^A, one can efficiently compute the value of κ. The values κ_{ij} are also NP-hard to compute, but it is possible to efficiently find their proxies and use them to obtain a good enough approximation $\hat{\kappa}$. In the way it is stated in the article by Dadush et al. [Dad+20], the theorem focuses on concepts that are not substantial in the setting of this thesis. We therefore formulate the theorem in a way that is more suited to our situation.

Theorem 14 ([Dad+20, Theorem 2.5])**.** *For every $i, j \in [n]$ let $\hat{\kappa}_{ij}$ satisfy*

$$\hat{\kappa}_{ij} \leq \kappa_{ij} \leq (\kappa^*)^2 \hat{\kappa}_{ij}. \tag{2.2}$$

Define $\hat{\kappa} := \max\{\hat{\kappa}_{ij} \mid i, j \in [n]\}$. Then it holds that

$$\hat{\kappa} \leq \kappa \leq n(\kappa^*)^2 \hat{\kappa}.$$

Notation. From now on, we will be using the notation $\hat{\kappa}_{ij}$ for any value satisfying (2.2). While possibly many values satisfy this, the same value is always considered when $\hat{\kappa}_{ij}$ occurs multiple times with the same indices.

That leaves us with the task to find the pairwise approximations $\hat{\kappa}_{ij}$. The idea is actually simple: find any circuit that contains both i and j and estimate κ_{ij} based on this circuit.

Proposition 15 ([Dad+20, Corollary 2.13])**.** *Let $i, j \in [n]$ and let C be any circuit such that $i, j \in C$. Then,*

$$\kappa_{ij}(C) \leq \kappa_{ij} \leq (\kappa^*)^2 \kappa_{ij}(C).$$

We put the task of finding such circuit aside for now, as it will be the central topic of Subsection 2.4.1.

Computing the diagonal matrix which witnesses an approximation $\hat{\kappa}^*$ of κ^* is more involved, but can still be done efficiently. This is again described in the article by Dadush et al. [Dad+20, Theorem 2.5], we just formulate the result in a more algorithm-centric way.

Definition 15 (Circuit ratio digraph)**.** *The circuit ratio digraph is a directed graph* $\mathcal{C}G_{\bullet}^{\bullet} := ([n], E)$*, where*

$$E := \{(i,j) \in [n]^2 \mid i \neq j, \kappa_{ij} > 0\}.$$

Remark. From Proposition 15 it follows that $\kappa_{ij} > 0$ if and only if $\hat{\kappa}_{ij} > 0$. The edge set can therefore be equivalently defined as

$$E := \{(i,j) \in [n]^2 \mid i \neq j, \hat{\kappa}_{ij} > 0\}.$$

Theorem 16 ([Dad+20, Theorem 2.5])**.** *Consider the optimization program*

$$\begin{aligned} &\min t \\ &\hat{\kappa}_{ij} d_j / d_i \leq t \quad \forall (i,j) \in E(\mathcal{C}G_{\bullet}^{\bullet}) \\ &\mathbf{d} > 0. \end{aligned} \tag{2.3}$$

Let $\hat{\mathbf{d}}$ *be the optimal value of* $\mathbf{d}$ *for* (2.3) *and* $D := \mathrm{Diag}(\hat{\mathbf{d}})$*. Then,*

$$\kappa_{AD} \leq (\kappa_A^*)^3.$$

Remark. Note that (2.3) can be made into a linear program by substituting all variables by their logarithms.

Notation. In the following text, let $\hat{\mathbf{d}}$ always denote the optimal value of $\mathbf{d}$ for (2.3).

This optimization problem can be solved in polynomial time using linear program solving techniques. There also exists an equivalent combinatorial description for the rescaling, which will yield additional insights.

Theorem 17 ([Dad+20, Theorem 2.5])**.** *Consider a weighted directed graph* G *given by assigning*

$$\begin{aligned} V(G) &:= [n], \\ E(G) &:= \{(i,j) \in [n]^2 \mid i \neq j, \hat{\kappa}_{ij} > 0\}, \\ w_G(e) &:= \hat{\kappa}_{ij} \quad \forall e \in E(G). \end{aligned}$$

Define $\hat{t}$ *as the maximum geometric mean of a cycle in* G*, that is*

$$\hat{t} := \max \left\{ \left(\sum_{e \in C} w_G(e) \right)^{\frac{1}{|C|}} \,\middle|\, C \text{ is a cycle in } G \right\}.$$

Let H *be a weighted directed graph given by assigning*

$$\begin{aligned} V(H) &:= V(G) \cup \{r\}, \\ E(H) &:= E(G) \cup \{(r,i) \mid i \in [n]\}, \\ w_H(e) &:= \begin{cases} \log \hat{t} - \log \hat{\kappa}_{ij} & \text{for } e \in E(H) \text{ s.t. } r \notin e, \\ 0 & \text{for } e \in E(H) \text{ s.t. } r \in e. \end{cases} \end{aligned}$$

Note that this graph has negative edge weights, but has no negative cycles. This is the case, because $\hat{t}$ *is defined as the maximum geometric mean of weights over all cycles in* G*. Graph* H *has the same cycles as graph* G*, so for every cycle* C *of* H

it must hold that $|C| \log \hat{t} \geq \sum_{(i,j)\in C} \log \hat{\kappa}_{ij}$. *Therefore, the Bellman-Ford shortest path algorithm can be used to determine the length of the shortest path from* r *to* i *in* H. *Let* s_i *denote this value.*

Define values

$$d_i := \exp(s_i) \quad \forall i \in [n].$$

Then, it holds that $\mathbf{d} = \hat{\mathbf{d}}$.

The proof of Theorem 17 in the article by Dadush et al. [Dad+20] gives interesting insight that allows us to exactly determine the value of κ^* for some matrices and see that not every matrix has $\hat{\kappa} = 1$:

Proposition 18. *For every* $i, j \in [n], i \neq j$ *it holds that*

$$\kappa_{ij}\kappa_{ji} \leq (\kappa^*)^2.$$

Lemma 19. *For every* $\bar{\kappa} > 1$ *the matrix*

$$A := \begin{pmatrix} \bar{\kappa} & 1 & \bar{\kappa} & 0 \\ \bar{\kappa} & \bar{\kappa}^2 & 0 & \bar{\kappa}^2 \end{pmatrix}$$

satisfies $\kappa_A^* = \kappa_A = \bar{\kappa}$.

Proof. The vector matroid $M(A)$ has exactly four circuits with elementary vectors

$$\begin{aligned} \mathbf{g_1} &= (\bar{\kappa}^2, -\bar{\kappa}, 1 - \bar{\kappa}^2, 0)^\top, \\ \mathbf{g_2} &= (1, -\bar{\kappa}, 0, \bar{\kappa} - 1)^\top, \\ \mathbf{g_3} &= (-\bar{\kappa}, 0, \bar{\kappa}, 1)^\top, \\ \mathbf{g_4} &= (0, \bar{\kappa}, -1, -\bar{\kappa})^\top. \end{aligned}$$

By taking the maximum over all pairwise circuit imbalance measures, we obtain $\kappa_A^* \leq \kappa_A = \bar{\kappa}$. We can further observe that $\kappa_{12} = \kappa_{21} = \bar{\kappa}$. By Proposition 18 it also holds that

$$\bar{\kappa}^2 = \kappa_{12}\kappa_{21} \leq (\kappa^*)^2.$$

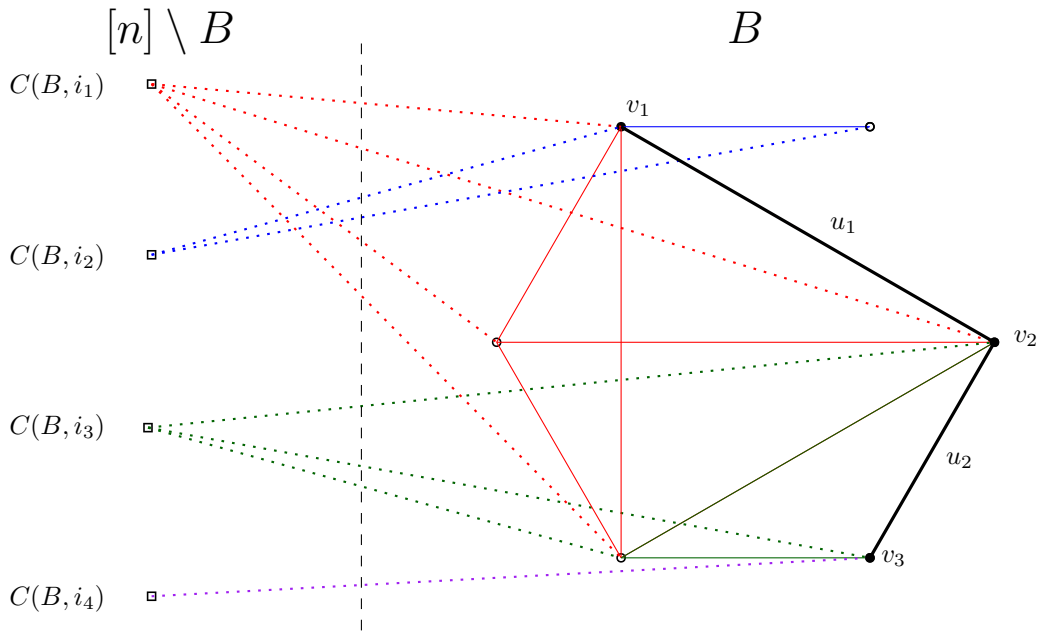
Combining the two inequalities yields $\kappa_A^* \leq \kappa_A = \bar{\kappa} \leq \kappa_A^*$ and consequently $\kappa_A^* = \kappa_A = \bar{\kappa}$. □

2.4.1 Finding Circuits

The last remaining thing to do is to find circuits C_{ij} that include both i and j. Note that we only consider connected matroids, so by Corollary 8 such circuits always exist. We provide an algorithm to find these circuits described in the article by Dadush et al. [Dad+20, Theorem 2.14].

Let us suppose we have a basis B of $M(A)$ and we start with the set of all fundamental circuits $\mathcal{F}_B$ with respect to B. If there is a circuit $C \in \mathcal{F}_B$ with $i, j \in \mathcal{F}_B$, we set $C_{ij} := C$ and we are done. Otherwise, we will find such circuit.

Figure 2.1: An example of G with shortest path between $C(B, i_1)$ and $C(B, i_4)$ drawn in black. Only the solid lines form the actual graph, the dotted lines only show which elements of B correspond to which fundamental circuit. Every color (other than black) corresponds to one fundamental circuit.



We construct an undirected graph G with $V(G) = B$ and edges between elements that are both contained in a fundamental circuit. Because of Proposition 9, G is connected.

First assume $i, j \notin B$. We can find a shortest path in G between any vertex in the set $C(B, i) \setminus \{i\}$ and any vertex in the set $C(B, j) \setminus \{j\}$. This can be represented by the sequences $V := \{v_1, v_2, \ldots, v_{\ell+1}\} \subseteq [n] \setminus B$, $v_1 = i, v_{\ell+1} = j$, and $U = \{u_1, u_2, \ldots, u_\ell\} \subseteq B$. The set V corresponds to vertices of G and the set U to edges of G. More precisely, we set $v_i := b$ for some $b \in B$ such that i-th edge on the path connects vertices in $C(B, b)$.

An illustration of this is shown in Figure 2.1. According to Proposition 11, the set $S := (B \setminus U) \cup V$ contains a unique circuit C that contains all v_t's, including i and j.

If $\{i, j\} \cap B \neq \emptyset$, without loss of generality it can be assumed that $i \in B$. Then, for any choice of $V = \{v_1, v_2, \ldots, v_{\ell+1}\}$ and $U = \{u_1, u_2, \ldots, u_\ell\}$ as in Proposition 11 such that $i \in C(B, v_1)$ and $i \notin C(B, v_t)$ for $t > 1$, the unique circuit in $(B \setminus U) \cup V$ also contains i. If both $i, j \in B$, we proceed in a similar manner with j. For more details see the original algorithm description in the article by Dadush et al. [Dad+20].

It remains to show how to find the unique circuit in S together with its elementary vector. There are standard tools of linear algebra that can be utilized for this task, but the article by Dadush et al. [Dad+20] also provides an alternative solution. We provide both of these solutions here and comparing their efficiency and suitability in our practical setting will be an important topic of Section 3.2.

Lemma 20. *Let $A \in \mathbb{R}^{n \times m}$ and let $S \subseteq [n]$ such that $M(A_S)$ contains a unique circuit C. Then, $\mathrm{Ker}(A_S)$ has dimension 1. We let the basic vector of $\mathrm{Ker}(A_S)$*

(unique up to scalar multiplication) be denoted by $\mathbf{v}$*. Then,* $\mathbf{v}$ *is an elementary vector for* C *and it holds that* $C = \mathrm{supp}(\mathbf{v})$*.*

Proof. We will only prove that $\dim(\mathrm{Ker}(A_S)) = 1$, the rest of the lemma statement follows in a straightforward way. Let C be a circuit of $M(A_S)$ and let $\mathbf{v_1}$ be its elementary vector. For contradiction, suppose that $\dim(\mathrm{Ker}(A_S)) \geq 2$. That implies that there exists another vector $\mathbf{v_2} \in \mathrm{Ker}(A_S)$ that is linearly independent from $\mathbf{v_1}$.

Let i be any element such that $i \in C$. If $(v_2)_i = 0$, define $\mathbf{v_3} := \mathbf{v_2}$. Otherwise, let $\mathbf{v_3} := \mathbf{v_2} - \frac{(v_1)_i}{(v_2)_i}\mathbf{v_1}$. In both cases, $i \notin \mathrm{supp}(\mathbf{v_3})$. Because $\mathbf{v_3} \in \mathrm{Ker}(A_S)$, $\mathrm{supp}(\mathbf{v_3})$ is a dependent set in $M(A_S)$. Let C' be any inclusion-wise minimal dependent subset of $\mathrm{supp}(\mathbf{v_3})$.

Because $i \in C$ and $i \notin C'$, it must hold that $C \neq C'$, which is a contradiction with the claim that $M(A_S)$ has exactly one circuit. □

Proposition 21 ([Dad+20, Inside proof of Theorem 2.14])**.** *Let us define an elementary vector* $\mathbf{g}$ *for circuit* C*. Start by setting* $g_{v_1} := 1$*. Using Gauss-Jordan elimination, convert the matrix* A *to the form* $A' = (I_m|H)$*. It then holds that for each* $t \in [\ell]$*, the row of* A'_S *corresponding to* u_t *contains only two nonzero entries:* $A'_{u_t v_t}$ *and* $A'_{u_t v_{t+1}}$*. Propagate the value* g_{v_1} *by assigning* $g_{v_2}, g_{v_3}, \ldots, g_{v_{\ell+1}}$*. Uniquely extend* $\mathbf{g}$ *to the indices in* $B \cap S$ *for any basis* B *so that* $\mathbf{g} \in \mathrm{Ker}(A)$*. Then,* $\mathbf{g}$ *is an elementary vector for* C *and it holds that* $C = \mathrm{supp}(\mathbf{g})$*.*

2.4.2 Algorithm Overview

To not only give all the ingredients, but also to give a high-level overview of how these ingredients should be put together, we provide a coarse outline of the main steps of the algorithm.

Data: Matrix $A \in \mathbb{R}^{n \times n}$
Result: Matrix $D \in \mathbf{D}$ such that κ_{AD} is not too far from κ^*_A
split A into $A_{V_1}, \ldots, A_{V_\ell}$, where $V_1, \ldots, V_\ell$ are the components of $M(A)$;
foreach $A' := A_{V_i}$ **do**
  find circuits C_{ij} of A' for every $i \neq j$ together with elementary vectors $\mathbf{g}_{ij}$;
  find an approximate rescaling matrix D_i of A' using these circuits and elementary vectors ;
end
combine the matrices $D_1, D_2, \ldots, D_\ell$ into a single matrix D ;

Algorithm 1: High-level overview of the algorithm

We do not overwhelm the reader with time complexities of the individual steps. It is, however, important to note that the whole algorithm can be implemented in time $\mathcal{O}(n^2m^2 + n^3)$.

2.5 LP Dataset

To be able to evaluate algorithms related to linear programming, we will be using a publicly available LP collection [Gay85] that is a part of Netlib repository. For brevity, we will be referring to this dataset as "Netlib" throughout this thesis.

There exists a similar, larger repository MIPLIB [Gle+21] for mixed integer programs (MIP). The ideas in this work are only suitable for linear programming problems and cannot be directly applied to mixed integer programs. To be able to utilize this dataset, we will be considering *LP relaxations* of problems in MIPLIB, that is their modifications where real values are allowed for all variables, instead of requiring integrality for some of them.

2.5.1 Problem Format

Both Netlib and MIPLIB problems are in MPS format, which is the industry standard for linear programming problems. That implies that the MPS format is capable of expressing integrality constraints on variables. As we are only interested in linear programming problems (that have no such constraints), we can safely assume that the MPS files used in our experiments do not use this capability.

We will not describe the internal details of the MPS format here, but it needs to be noted that the internal representation is very similar to a sparsely represented modelling LP form described in Subsection 2.2.1. Reading the problem file into a convenient representation will be discussed later in Section 3.1. Most of the problems from both datasets have many zeroes and therefore the sparse representation is efficient.

The Netlib dataset does not directly provide the MPS files for the problems, but compressed versions of the MPS files are provided instead for most problems. These files can be decompressed using the utility `emps`, also available in Netlib, to obtain the actual problem MPS files. Several of the problems are only available as fortran programs generating the problem file in MPS. There are only several such problems, so we exclude these problems from our dataset for simplicity.

3. Algorithm Implementation

This chapter strives to describe the details of our implementation of the approximation algorithm, to explain the choices that were made along the way and compare them to alternative approaches. We also describe the challenges that we discovered in the process.

The SageMath [The22] software will be used for the implementation of the algorithm. SageMath uses Python and can be used directly from Python code, which gives the advantage of a variety of available scientific packages. SageMath itself has support for working with both densely and sparsely represented matrices over both exact and inexact fields, working with graphs and it has some support for working with matroids. These capabilities make SageMath a good fit for our application.

With Python being an interpreted language, it might be possible to achieve smaller run-times by using a more low-level programming language. However, a large portion of the run-time of our implementation is spent inside library functions that have their heavily optimized low-level implementations. For this reason, the performance gain cannot be expected to be too large. Following this observation, and also taking into account the possible greater implementation complexity of alternative approaches, we have not further explored this possibility.

3.1 Reading LP Problems

As mentioned in Subsection 2.5.1, the Netlib LP dataset provides its problems in the MPS format that has nearly identical representation as the modelling LP form. The algorithm from Section 2.4, however, works with an LP in the standard form. That gives two different challenges for this section to deal with. The first one is solving how to read the problem matrix from the MPS file to a format that can be conveniently used within SageMath.

The second problem is solving the form mismatch. As the reader learned in Subsection 2.2.1, this problem could be remediated by performing an efficient conversion of the problem representation to one in standard form. We present a more efficient approach that allows us to run the algorithm with a smaller problem matrix.

3.1.1 Reading Problem Matrix from MPS

Several well-known and maintained Python libraries allow reading linear programs from the MPS format. The problem with most of these libraries is that it is not easy to access the problem matrix itself. In order to avoid writing an MPS parser from scratch, we utilize a Python package `scikit-glpk`[1], licensed under the GNU General Public Licence v3.0. This package provides Python bindings for GLPK and allows to read the problem into modelling form as sparse SciPy matrices and vectors, that can be easily converted to sparse matrices and vectors native to SageMath.

[1] Available from `https://github.com/mckib2/scikit-glpk`

Prior to our work, this package was not able to read all MPS files that are used in Netlib and MIPLIB. To allow efficient reading all of the problems, we have made some changes to the library. The changes were contributed back to the original package in the form of GitHub pull requests[2] and are now a part of the original codebase.

The first modification of the library was a simple bug fix. The bug was caused by earlier modifications of the library that rendered the MPS reading and writing capabilities defunct.

The second modification solved the fact that the MPS reader inside GLPK sometimes internally represents constraints bounded from both sides, i.e.

$$b_1 \leq \mathbf{a}^\top \mathbf{x} \leq b_2.$$

As this does not directly correspond to a constraint in modelling form, the library did not support reading the problems whose internal GLPK representation contained such a constraint. We have modified the library to split any such constraint into two constraints

$$\begin{aligned} b_1 &\leq \mathbf{a}^\top \mathbf{x} \\ b_2 &\geq \mathbf{a}^\top \mathbf{x}. \end{aligned}$$

Lastly, we have significantly improved the library performance for working with sparse matrices. The original implementation required the use of a dense representation for combining two matrices in one and only then converted the resulting matrix into a sparsely represented one. We used a sparse implementation for the combining step, avoiding the need for costly conversion to the dense representation altogether.

3.1.2 Solving Form Mismatch

The standard procedure to convert an LP in modelling form to an LP in standard form turns inequality constraints into equality constraints and turns generic bounds into non-negativity bounds. The key insight is that converting the bounds is not needed, as this step does not change the value of κ nor the value of κ^*.

Definition 16 (Semi-standard LP Form)**.** *Let* $A \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$ *and* $\mathbf{l}, \mathbf{u} \in (\mathbb{R} \cup \{-\infty, \infty\})^n$. *Then a linear program in* semi-standard form *is the problem*

$$\begin{aligned} \min\ & \mathbf{c}^\top \mathbf{x} \\ & A\mathbf{x} = \mathbf{b} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}. \end{aligned}$$

We can observe that the standard algorithm for converting an LP in semi-standard form to an LP in standard form only uses the following operations:

[2] `https://github.com/mckib2/scikit-glpk/pull/24`, `https://github.com/mckib2/scikit-glpk/pull/25`, `https://github.com/mckib2/scikit-glpk/pull/23`

(O1) Take a variable x_i and express it as $x_i = x_i^+ - x_i^-$ for new variables $x_i^+, x_i^- \geq 0$. Let, without loss of generality,

$$A = \left(\begin{array}{ccc|c} & & & \vdots \\ & \bar{A} & & A_{*i} \\ & & & \vdots \end{array}\right), \qquad \mathbf{x} = \left(\begin{array}{c} \vdots \\ \hat{\mathbf{x}} \\ \vdots \\ \hline x_i \end{array}\right).$$

The problem matrix after this operation is then

$$A' = \left(\begin{array}{ccc|c} & & & \vdots & \vdots \\ & \bar{A} & & A_{*i} & -A_{*i} \\ & & & \vdots & \vdots \end{array}\right).$$

(O2) Take a bound $\pm x \geq l$ and turn it into a constraint. If we assume the same layout of $\mathbf{x}$ as in (O1), the problem matrix after this operation is

$$A' = \left(\begin{array}{ccc|c} & & & \vdots & 0 \\ & \bar{A} & & A_{*i} & \vdots \\ & & & \vdots & 0 \\ \hline 0 & \cdots & 0 & \pm 1 & 1 \end{array}\right).$$

We will show that performing neither operation changes the values κ, κ^*. By induction it then follows that performing the conversion algorithm also does not change these values.

Lemma 22. $\kappa_A \geq 1$.

Proof. Let us choose any $C \in M(A)$. For all $i, j \in \binom{C}{2}$, both values $\kappa_{ij}^A(C)$ and $\kappa_{ji}^A(C)$ are considered when determining the value of $\kappa_A(C)$. Because $\kappa_{ij}^A = 1/\kappa_{ji}^A$, it must be the case that $\max\{\kappa_{ij}^A(C), \kappa_{ji}^A(C)\} \geq 1$. Therefore, the maximum over all pairs of i, j is also greater than 1, meaning that $\kappa_A(C) \geq 1$ for any circuit C. By taking maximum over all circuits, we get the lemma statement. □

Lemma 23. *Performing operation (O1) does not change the value of κ.*

Proof. Let A, A' be problem matrices before and after performing the operation respectively. Firstly we will show the inequality $\kappa_A \leq \kappa_{A'}$. Let $C \in \mathcal{C}(M(A))$ be the circuit that $\kappa_A = \kappa_A(C)$ and let $\mathbf{g}^{\mathbf{C}} = (\bar{\mathbf{g}}^{\mathbf{C}}, g_i)^\top$ be its elementary vector. Define the vector $\mathbf{g}^{\mathbf{C}'} := (\bar{\mathbf{g}}^{\mathbf{C}}, g_i, 0)^\top$. For this vector it holds that

$$\left(\begin{array}{ccc|c} & & & \vdots & \vdots \\ & \bar{A} & & A_{*i} & -A_{*i} \\ & & & \vdots & \vdots \end{array}\right) \left(\begin{array}{c} \vdots \\ \bar{\mathbf{g}}^{\mathbf{C}} \\ \vdots \\ \hline g_i \\ 0 \end{array}\right) = A\mathbf{g}^{\mathbf{C}} = \mathbf{0}.$$

We will show that $C' = \mathrm{supp}(\mathbf{g}^{\mathbf{C}'})$ is a circuit of $M(A')$. From the above it directly follows that C' is a dependent set. For contradiction let us now assume that there exists a dependent set of $M(A')$ denoted by $C'' \subset C'$. Let $(\bar{\mathbf{g}}^{\mathbf{C}''}, 0)^\top$ be an elementary vector for this set. Then, $\mathrm{supp}(\bar{\mathbf{g}}^{\mathbf{C}''}) \subset C$ is an independent set of $M(A)$, which contradicts that C is a circuit.

Because $\mathbf{g}^{\mathbf{C}'}$ has the same non-zero values as $\mathbf{g}^{\mathbf{C}}$, it holds that $\kappa_A(C) = \kappa_{A'}(C')$. From this it follows that

$$\kappa_A = \kappa_A(C) = \kappa_{A'}(C') \leq \kappa_{A'}.$$

For the other inequality, let us consider any $C' \in \mathcal{C}(M(A'))$ together with its elementary vector $\mathbf{g}^{\mathbf{C}'}$. Because the last two columns of A' are linearly dependent, $\mathbf{g}^{\mathbf{C}'}$ can only have one of two forms:

1. Either $\mathbf{g}^{\mathbf{C}'} = (\bar{\mathbf{g}}^{\mathbf{C}'}, g_i, 0)^\top$ or $\mathbf{g}^{\mathbf{C}'} = (\bar{\mathbf{g}}^{\mathbf{C}'}, 0, -g_i)^\top$. In that case, $(\bar{\mathbf{g}}^{\mathbf{C}'}, g_i)^\top$ is an elementary vector for some vector $C \in \mathcal{C}(M(A))$ and it holds that $\kappa_{A'}(C') = \kappa_A(C)$.
2. Otherwise, $\mathbf{g}^{\mathbf{C}'} = (\mathbf{0}, g_i, g_i)^\top$. Then clearly $\kappa_{A'}(C') = 1$.

For any circuit C' it therefore holds that $\kappa_{A'}(C') \leq \max\{\kappa_A, 1\}$. By using the previously stated Lemma 22, we get the inequality $\kappa_{A'}(C') \leq \kappa_A$.

Consider C'' such that $\kappa_{A'} = \kappa_{A'}(C'')$. Then it is holds that

$$\kappa_{A'} = \kappa_{A'}(C'') \leq \kappa_A. \qquad \square$$

Lemma 24. *For any $D \in \mathbf{D}$, $\mathcal{C}(M(A)) = \mathcal{C}(M(AD))$.*

Proof. Multiplying vectors by nonzero constants in a set of vectors does not change the linear dependence of the set. Therefore, rescaling columns with matrix D does not change dependent sets nor inclusion-wise minimal dependent sets. $\square$

Lemma 25. *Performing operation (O1) does not change the value of κ^*.*

Proof. Let A, A' be matrices as in the previous proof. We will start by showing that $\kappa^*_{A'} \leq \kappa^*_A$. Consider $D \in \mathbf{D}$ such that $\kappa^*_A = \kappa_{AD}$. Define

$$D' := \left(\begin{array}{ccc|c} & & & 0 \\ & \bar{D} & & \vdots \\ & & & 0 \\ & & & d_i \\ \hline 0 & \cdots & 0 & d_i \end{array}\right).$$

Consider the circuit $C' \in \mathcal{C}(M(A'D'))$ such that $\kappa_{A'D'} = \kappa_{A'D'}(C')$ with an elementary vector $\mathbf{g}^{\mathbf{C}'}$.

Using Lemma 24, we again know that this circuit can be of one of two types:

1. If $\mathbf{g}^{\mathbf{C}'} = (\bar{\mathbf{g}}^{\mathbf{C}'}, g_i, 0)^\top$ or $\mathbf{g}^{\mathbf{C}'} = (\bar{\mathbf{g}}^{\mathbf{C}'}, 0, -g_i)^\top$, then the vector $\mathbf{g}^{\mathbf{C}} := (\bar{\mathbf{g}}^{\mathbf{C}'}, g_i)^\top$ is an elementary vector for a circuit $C \in \mathcal{C}(M(AD))$. Because this vector shares the same nonzero values with $\mathbf{g}^{\mathbf{C}'}$, it holds that

$$\kappa^*_{A'} \leq \kappa_{A'D'} = \kappa_{A'D'}(C') = \kappa_{AD}(C) \leq \kappa_{AD} = \kappa^*_A.$$

2. Otherwise, $\mathbf{g}^{\mathbf{C}'} = (\mathbf{0}, g_i, g_i)^\top$. In that case, $\kappa_{A'D'}(C') = 1$. By Lemma 22,

$$\kappa^*_{A'} \leq \kappa_{A'D'} = \kappa_{A'D'}(C') = 1 \leq \kappa_{AD} = \kappa^*_A.$$

To prove the other inequality, consider the $(n+1) \times (n+1)$ diagonal matrix with positive entries on the diagonal

$$D' = \left(\begin{array}{c|c} \bar{D'} & \begin{matrix}0\\ \vdots \\ 0\end{matrix} \\ \hline 0 \cdots\cdots 0 & d_{n+1} \end{array}\right)$$

such that $\kappa^*_{A'} = \kappa_{A'D'}$.

Define $D := \bar{D}'$ and consider $C \in \mathcal{C}(M(AD))$ such that $\kappa_{AD} = \kappa_{AD}(C)$ together with the elementary vector $\mathbf{g}^{\mathbf{C}}$. Then, the vector $\mathbf{g}^{\mathbf{C}'} := (\mathbf{g}^{\mathbf{C}}, 0)^\top$ is an elementary vector of a circuit in $M(A'D')$ with $\kappa_{A'D'}(C') = \kappa_{AD}(C)$. Therefore it must hold that

$$\kappa^*_A \leq \kappa_{AD} = \kappa_{AD}(C) = \kappa_{A'D'}(C') \leq \kappa_{A'D'} = \kappa^*_{A'}. \qquad \square$$

Lemma 26. *Performing operation (O2) does not change the value of κ.*

Proof. Let A, A' be matrices as before, $C' \in \mathcal{C}(M(A'))$ such that $\kappa_{A'} = \kappa_{A'}(C')$ and its elementary vector $\mathbf{g}^{\mathbf{C}'}$. For the elementary vector it must hold that

$$\left(\begin{array}{c|c} A & \begin{matrix}0\\ \vdots \\ 0\end{matrix} \\ \hline 0 \cdots\cdots 0 \quad \pm 1 & 1 \end{array}\right) \left(\begin{array}{c} \vdots \\ \bar{\mathbf{g}}^{\mathbf{C}'} \\ \vdots \\ \hline g_{n+1} \end{array}\right) = \left(\begin{array}{c} \vdots \\ \mathbf{0} \\ \vdots \\ \hline 0 \end{array}\right).$$

This can be split into two conditions

$$\bar{\mathbf{g}}^{\mathbf{C}'} \in \mathrm{Ker}(A)$$
$$g_{n+1} = \pm g_n.$$

We will show that $\mathbf{g}^{\mathbf{C}} := \bar{\mathbf{g}}^{\mathbf{C}'}$ is an elementary vector of a circuit $C := \mathrm{supp}(\bar{\mathbf{g}}^{\mathbf{C}'}) \in \mathcal{C}(M(A))$. We already know that C is dependent. For contradiction let us now suppose that there exists another dependent set $\hat{C} \subset C$. This circuit can be extended to a circuit $\hat{C}' \in \mathcal{C}(A')$ such that $\hat{C}' \subset C'$, contradicting that C' is a circuit. The set C is therefore maximal dependent.

Vectors $\mathbf{g}^{\mathbf{C}'}$ and $\mathbf{g}^{\mathbf{C}}$ have the same nonzero absolute values, which yields $\kappa_A(C) = \kappa_{A'}(C')$. Consequently,

$$\kappa_{A'} = \kappa_{A'}(C') = \kappa_A(C) \leq \kappa_A.$$

For showing the opposite inequality, it suffices to observe that every elementary vector $\mathbf{g}^{\mathbf{C}}$ for a circuit $C \in \mathcal{C}(A)$ can be uniquely extended to an elementary vector $\mathbf{g}^{\mathbf{C}'}$ for a circuit $C' \in \mathcal{C}(A')$ such that $\kappa_A(C) = \kappa_{A'}(C')$. Specifically for C such that $\kappa_A = \kappa_A(C)$, it then follows that

$$\kappa_A = \kappa_A(C) = \kappa_{A'}(C') \leq \kappa_{A'}. \qquad \square$$

Lemma 27. *Performing operation (O2) does not change the value of κ^*.*

Proof. We again denote the matrices A and A' before and after performing the operation. First let us show that $\kappa^*_{A'} \geq \kappa^*_A$. Let D' be a diagonal matrix with positive numbers on the diagonal such that $\kappa^*_{A'} = \kappa_{A'D'}$. The scaled matrix can be written as

$$A'D' = \left(\begin{array}{ccc|c} & A\bar{D}' & & \begin{matrix}0\\\vdots\\0\end{matrix} \\ \hline 0 & \cdots\cdots & 0 \quad \pm d_n & d_{n+1} \end{array}\right).$$

For any elementary vector $\mathbf{g}^{\mathbf{C}'} = (\bar{\mathbf{g}}^{\mathbf{C}'}, g_{n+1})^\top$ of a circuit in $\mathcal{C}(M(A'D'))$ it must hold that

$$\begin{aligned} \bar{\mathbf{g}}^{\mathbf{C}'} &\in \mathrm{Ker}(A\bar{(}D')) \\ g_{n+1} &= \pm\tfrac{d_n}{d_{n+1}} g_n. \end{aligned}$$

Let $C \in \mathcal{C}(A\bar{D})$ be a circuit such that $\kappa_{A\bar{D}}(C) = \kappa_{A\bar{D}}$ with an elementary vector $\mathbf{g}^{\mathbf{C}}$. Define $\mathbf{g}^{\mathbf{C}'} := (\mathbf{g}^{\mathbf{C}}, \pm\frac{d_n}{d_{n+1}} g_n)^\top$ and observe that $C' := \mathrm{supp}(\mathbf{g}^{\mathbf{C}'}) \in \mathcal{C}(M(A'D'))$.

Because $\mathbf{g}^{\mathbf{C}'}$ contains all the non-zero elements that $\mathbf{g}^{\mathbf{C}}$ contains, it must be the case that $\kappa_{A'D'}(C') \geq \kappa_{A\bar{D}}(C)$, leaving us with

$$\kappa^*_{A'} = \kappa_{A'D'} \geq \kappa_{A'D'}(C') \geq \kappa_{A\bar{D}}(C) = \kappa_{A\bar{D}} \geq \kappa^*_A.$$

For the other direction, consider $D \in \mathbf{D}$ such $\kappa^*_A = \kappa_{AD}$. Choose D' such that

$$D' := \left(\begin{array}{cc|c} & 0 & 0 \\ \bar{D} & \vdots & \vdots \\ & 0 & 0 \\ & d_n & 0 \\ \hline 0 & \cdots\cdots\cdots\ 0 & d_n \end{array}\right).$$

Consider the circuit $C' \in \mathcal{C}(M(A'D'))$ such that $\kappa_{A'D'}(C') = \kappa_{A'D'}$ together with its elementary vector $\mathbf{g}^{\mathbf{C}'} = (\bar{\mathbf{g}}^{\mathbf{C}'}, g_{n+1})^\top$. Plugging into the conditions above, it must hold that

$$\begin{aligned} \bar{\mathbf{g}}^{\mathbf{C}'} &\in \mathrm{Ker}(AD) \\ g_{n+1} &= \pm g_n. \end{aligned}$$

We can observe that $\bar{\mathbf{g}}^{\mathbf{C}'}$ is an elementary vector of a circuit $C \in \mathcal{C}(M(AD))$. The vectors $\mathbf{g}^{\mathbf{C}'}$ and $\bar{\mathbf{g}}^{\mathbf{C}'}$ have the same nonzero absolute values, therefore $\kappa_{A'D'}(C') = \kappa_{AD}(C)$. By putting everything together, we get

$$\kappa^*_A = \kappa_{AD} \geq \kappa_{AD}(C) = \kappa_{A'D'}(C') = \kappa_{A'D'} \geq \kappa^*_{A'}. \qquad \square$$

By combining Lemmas 23, 25, 26 and 27, we finally get the following theorem.

Theorem 28. *Let A be a problem matrix of an LP in semi-standard form and A' a problem matrix for the same LP converted to standard form using the standard construction. Then, $\kappa_A = \kappa_{A'}$ and $\kappa^*_A = \kappa^*_{A'}$.*

3.2 Practical Aspects of Dealing With the Vector Matroid

In Section 2.3 we have introduced the mathematical notions of matroid and vector matroid. In order to successfully perform computations with the vector matroid and to understand their efficiency, it is important to understand how some of the operations with vector matroids can be implemented. These details will have great implications for the practicality and scalability of the experiments.

3.2.1 Exactness of Computation

When performing computations with real numbers on modern computers, a floating point number representation is typically used. While this representation is convenient in many cases, practical calculations using these numbers suffer from an inherent precision loss. We will show that this causes issues when using float computations for working with the vector matroid of a matrix. Not only can small numerical differences in values of the matrix completely change the structure of the vector matroid, but they may also arbitrarily change both the values κ and κ^*.

Example 1 (Floating-point arithmetics and vector matroid)**.** *For $a \in \mathbb{R}$ consider the matrix*

$$A := \begin{pmatrix} a & a & 4a \\ 0 & a & a \end{pmatrix}.$$

Because $3 \cdot A_{*1} + A_{*2} = A_{*3}$, $\{1, 2, 3\}$ is not an independent set of $M(A)$. However, when using floating-point operations $\cdot_F$ and $+_F$, one may get the result

$$3 \cdot_F \begin{pmatrix} a \\ 0 \end{pmatrix} +_F \begin{pmatrix} a \\ a \end{pmatrix} = \begin{pmatrix} 4a + \varepsilon_1 \\ a + \varepsilon_2 \end{pmatrix}$$

for small values of ε_1 and ε_2. This might lead an algorithm relying on floating-point arithmetics to believe that all the three columns are linearly independent and that $\{1, 2, 3\}$ is an independent set of $M(A)$.

We tried to take advantage of the fast calculations in limited precision for finding a unique circuit in a subset of elements by using Lemma 20. The key idea is to use inexact arithmetics with increasing precision to obtain the solution and then check the (potentially) small solution using exact arithmetics. The approach is described in detail in Algorithm 2.

Function `CheckIsCircuit`(C, A)
  if $\dim(\mathrm{Ker}(A_C)) \neq 1$ **then**
    return *false* ;
  end
  $\mathbf{g} \leftarrow$ find kernel element in A_C with exact arithmetics ;
  if $\mathbf{g}$ *contains a zero* **then**
    return *false* ;
  end
  return *true* ;
end

Data: Matrix A, subset of columns S such that $M(A_S)$ has a unique circuit, values $\varepsilon_0 \in \mathbb{R}^+$, $\varepsilon_\delta \in (0,1)$, $p_0 \in \mathbb{N}^+$, $p_\delta \in \mathbb{N}^+$
Result: The unique circuit of $M(A_S)$
$p \leftarrow p_0$, $\varepsilon \leftarrow \varepsilon_0$;
repeat
  $\mathbf{g} \leftarrow$ find unique kernel element in A_S with p-bit precision arithmetics ;
  $C \leftarrow \{i \mid |\mathbf{g}_i| < \varepsilon\}$; `// support with tolerance` ε
  $p \leftarrow p + p_\delta$; `// increase precision`
  $\varepsilon \leftarrow \varepsilon \cdot \varepsilon_\delta$; `// decrease tolerance`
until `CheckIsCircuit`(A, C) = *true*;

Algorithm 2: Find-and-verify approach to finding a unique circuit

This approach turned out to be impractical because all attempted choices of p_δ, ε_δ led to much larger iteration counts than was reasonable in exchange for the time improvement given by running the exact algorithm on a smaller instance. As a matter of fact, the algorithm converged in reasonable time only for very small problem instances.

While it may be possible to work with a vector matroid using inexact arithmetics in a consistent way (i.e. the sets that an algorithm working with inexact arithmetics marked as independent satisfy matroid axioms), I have not discovered such a sensible way. For this reason, we will be forced to use exact arithmetics throughout the algorithm. To do that, we will be using the support for rational numbers that SageMath provides.

SageMath also lacks support for working with vector matroids of matrices over the Real Field with limited precision. It is possible to create a `LinearMatroid` object with such matrix and call its methods, but they may not give consistent results. We have observed this deficiency and reported it, which led to creation of a ticket[3] on SageMath bug tracker. At the time of writing this text, the documentation nor the behavior have been fixed yet.

In addition to the fact that working with the vector matroid without using exact arithmetics is at least tricky, we can also observe that small changes in matrix values can inflict large changes of both κ and κ^*. This can be dangerous if we obtain a nearly optimal rescaling of a matrix and then try to use it in inexact context (e.g., run an LP solver that uses float arithmetics).

Example 2 (Floating-point arithmetics and κ)**.** *For $a, C \in \mathbb{R}^+$ consider the*

[3]Now available at: `https://github.com/sagemath/sage/issues/33657`

matrix

$$A := \begin{pmatrix} a & a & Ca+\varepsilon \\ a & a & Ca \end{pmatrix}.$$

In a mathematical sense, the only circuit of $M(A)$ is $\{1,2\}$ and $\mathbf{g} = (1,-1,0)^\top$ is its elementary vector, implying $\kappa_A = 1$. An algorithm based on floating-point numbers could however falsely believe that $\{2,3\}$ is a circuit with an elementary vector $(0,C,-1)^\top$ and therefore that $\kappa_A = C$.

Example 3 (Floating-point arithmetics and κ^*)**.** *The vector matroid of matrix*

$$A := \begin{pmatrix} \bar{\kappa} & 1 & \bar{\kappa} & 0 \\ \bar{\kappa} & \bar{\kappa}^2 & 0 & \bar{\kappa}^2 \\ \varepsilon & 0 & 0 & 0 \end{pmatrix}$$

for $\bar{\kappa} > 1$ has a single circuit $\{2,3,4\}$. The scaling matrix

$$D := \begin{pmatrix} 1 & & & \\ & \bar{\kappa} & & \\ & & 1 & \\ & & & \bar{\kappa} \end{pmatrix} \qquad \textit{gives} \qquad AD = \begin{pmatrix} \bar{\kappa} & \bar{\kappa} & \bar{\kappa} & 0 \\ \bar{\kappa} & \bar{\kappa}^3 & 0 & \bar{\kappa}^3 \\ \varepsilon & 0 & 0 & 0 \end{pmatrix}.$$

The only circuit then has an elementary vector $\mathbf{g} = (0,1,-1,-1)^\top$, witnessing $\kappa^*_A = \kappa_{AD} = 1$.

An algorithm using floating-point numbers may however give results as if $\varepsilon = 0$ due to loss of precision. The situation is then similar to the one in Lemma 19, which, as observed by the algorithm, effectively makes $\kappa^*_A = \bar{\kappa}$.

If we try to use the exactly computed scaling matrix D in inexact context (assuming ε is treated as 0), we get $\kappa_{AD} = \bar{\kappa}^2$. That shows that not only the value κ^* is sensitive to inaccuracies, but the same is also true for the scaling matrix D.

3.2.2 Needed Matroid Operations

For the purposes of the algorithm, there is no need to worry about implementation aspects of every single vector matroid operation. We will only be interested in the following operations that the implemented algorithm uses:

- Components() splits the vector matroid to matroids for its components,
- Basis() finds any basis of the vector matroid,
- FundamentalCircuits(B) finds all fundamental circuits with respect to basis B,
- ElementaryVector(C) finds an elementary vector for the circuit C and
- UniqueCircuit(B, U, V) finds a circuit contained in $(B \setminus U) \cup V$ that is guaranteed to be unique, where $U \subseteq B$, $V \subseteq [n] \setminus B$.

3.2.3 Utilizing SageMath

SageMath comes with a class `LinearMatroid` that provides most of the operations out of the box and the rest is not hard to implement using the tools it provides. If `M` is an instance of `LinearMatroid`, then Components() can be implemented by using `M.components()`, Basis() can be implemented by using `M.basis()` and FundamentalCircuits(B) can be implemented by using the method `M.fundamental_circuit(B, e)` for every non-basic element e.

The operations ElementaryVector(C) and UniqueCircuit(B, U, V) can be implemented either using the method in Lemma 20 or the one in Proposition 21. In both cases, we will work directly with the matrix A. In the first case this can be done using `A.matrix_from_columns(C).right_kernel().basis()[0]` for ElementaryVector(C) and `A.matrix_from_columns(B.difference(U).union(V)).right_kernel().basis()[0].support()` for implementing the operation UniqueCircuit(B, U, V).

The latter option is a bit more involved. Firstly, the matrix A needs to be converted to RREF using Gauss-Jordan elimination. Then, we construct a "vector finding graph" that will allow for efficient implementation of the procedure.

Definition 17 (Vector finding graph)**.** *Let A be a matrix in RREF such that $M(A)$ is connected, let U, V and B be sets as above and let $S := (B \setminus U) \cup V$. Consider the submatrix $\bar{A}$ obtained by selecting exactly the columns in S and the rows where columns in U have their only nonzero-value (this value is 1). Let $\overline{uv}$ denote the indices such that $\bar{A}_{\overline{uv}}$ corresponds to A_{uv}.*

Then, the vector finding graph *of A is a weighed directed graph G such that*

$$\begin{aligned} V(G) &:= [n], \\ E(G) &:= \left\{ (u,v) \in [n]^2 \;\middle|\; \begin{array}{l} \textit{there is a row } r_{\overline{uv}} \textit{ in } \bar{A} \textit{ whose only nonzero} \\ \textit{values are in columns } \bar{u} \textit{ and } \bar{v} \end{array} \right\}, \\ w_G((u,v)) &:= -\frac{\bar{A}_{r_{\overline{uv}}\bar{u}}}{\bar{A}_{r_{\overline{uv}}\bar{v}}}. \end{aligned}$$

This vector finding graph can be constructed in time $\mathcal{O}(nm)$. The meaning of this graph is that for an elementary vector $\mathbf{g}$ of the unique circuit of $M(A)$ it holds that if $(u,v) \in E(G)$, then $\frac{g_v}{g_u} = w_G((u,v))$.

Practically, the procedure from Proposition 21 can be implemented as follows: we start with an empty vector $\mathbf{w}$ and set $w_{v_1} \leftarrow 1$. Then, we run a depth-first search (DFS) from v_1 in the vector finding graph and for every discovered edge (i,j), we set $w_j \leftarrow w_i \cdot w_G((i,j))$. This order of processing edges of the graph guarantees that we use every edge (i,j) only after the value w_i is known, everything that can be propagated is propagated and no value g_j is computed more that once. Lastly, we need to set values to indices in $B \cap S$ such that $\mathbf{w} \in \mathrm{Ker}(A)$. This can be done by solving a system of linear equations, or specifically by setting

$$\mathbf{w}_{B \cap S} \leftarrow -(A\mathbf{w})_{B \cap S}.$$

Remark. Note that it suffices to convert the matrix A to RREF only once in the runtime of the entire algorithm and then reuse the result for many invocations of ElementaryVector and UniqueCircuit.

Upon inspection of the source code of SageMath, we have discovered that its vector matroid implementation can not leverage matrix sparsity. In particular, the matrix is converted to an internal dense representation when constructing the vector matroid object. Because matrices of problems in Netlib and MIPLIB usually have many zeroes, representing them densely has great impact on both memory and runtime of the operations. It is therefore natural to explore other options that work with the matrix directly in its sparse representation.

3.2.4 Sparsity and RREF

While it might seem sufficient to modify the algorithm internal details to work directly with the sparse representation, there are more fundamental problems that make it impossible to fully avoid the memory blow-up caused by converting to a dense representation. The root of the problem lies in needing the RREF of matrix A. Even when the original matrix has many zeroes and its sparse representation is efficient, after converting it to RREF it is often not the case and the representation becomes inefficient. This is referred to as matrix *fill-in* and has been studied for example by Brayton, Gustavson and Willoughby [BGW]. This means that the memory savings (and consequently the time savings) that motivated the use of sparse matrices are not directly possible.

Note that while we can avoid using RREF explicitly by using the alternative method in Lemma 20, the SageMath implementation of vector matroid echelonizes the matrix A internally.

I am aware that it might be possible to reorder columns of the matrix in a way that minimizes the fill-in. While there are strategies that minimize the fill-in when computing LU and QR decompositions, I do not know of any strategies that do the same for computing RREF. Also, any such algorithm would need to be manually implemented either in Python or Cython and its fast implementation would arguably not be trivial. For these reasons, we do not further explore this approach as we consider it to be out of scope of this thesis.

3.2.5 Implementing Vector Matroid Using Sparse Operations

To implement vector matroid operations in a way that uses sparsity, we use SageMath abilities to work with sparse matrices. Note that we do not examine the internals of SageMath implementations of sparse matrix operations and assume that they are optimal. In particular, we are not concerned whether the operations cause any fill-in internally.

For implementing Components we leverage Lemma 10 and determine matroid components by looking for components of the graph of fundamental circuits. Note that we need this graph later in the algorithm so we can cache the result and compute it only once.

For the implementation of Basis we construct the basis iteratively. We start with the empty set. Then, for every vector, try adding it to the set and if the result is a linearly independent set of vectors, keep it there. By the Steinitz exchange theorem (Theorem 4) the obtained set is a basis.

For computing ElementaryVector and UniqueCircuit we can use the approach from Subsection 3.2.3 using Lemma 20.

Operation FundamentalCircuits can be implemented very similarly using Lemma 20, by following the definition of a fundamental circuit. As in the case of the first two operations, the result of this operation can be cached.

3.2.6 Comparison of the Methods

The choice of a suitable methods depends on the available computational resources. Thanks to its efficient use of sparsity, the second method based on sparse operations is a good fit for scenarios where available memory is limited. The SageMath-based method exhibited higher memory usage during our tests, but this method has the benefit considerable shorter run times than the alternative.

Because of our time constraints, we have opted for the SageMath-based method. It is, however, not clear whether using the slower method can be more practically feasible for some extremely memory-demanding problems, and there is potential for further research in this direction.

3.3 Other Implementation Concerns

There are other operations that the algorithm needs to perform that do not concern the vector matroid. This is most notably the case of solving the maximum geometric mean problem for the pairwise imbalance measure digraph and the single-source shortest path problem for finding the rescaling diagonal values.

3.3.1 Maximum Geometric Mean Problem

The maximum geometric mean problem can be converted to the minimum geometric mean problem in a straightforward way. Then, this problem can be further reduced to the minimum arithmetic mean problem by first computing logarithms of all edge weights. This is a well-known problem that can be solved by using Karp's algorithm [Kar78].

As far as I am aware, there is no implementation of Karp's algorithm inside SageMath. We have therefore decided to implement the algorithm in plain Python. While this dynamic programming algorithm is a textbook example of an algorithm with great potential to be sped up by being implemented in a more low-level language, this procedure runs only for a small portion of the total algorithm runtime. Any optimizations of the implementation of this method would therefore have only a small effect on the total performance.

Remark. At the point in the algorithm where maximum geometric mean is computed there is no longer the need for exact arithmetics. By inaccuracies from computing the logarithms, the maximum geometric mean can change only by a small amount. In the shortest path finding step, the shortest path lengths will also only change by a small amount. In Lemma 29 we will later show that this is not a problem.

3.3.2 Single-source Shortest Path Problem

The graph in which we need to find shortest paths from u to all other vertices v does contain negative edges, so the natural choice for finding single-source shortest paths is the Bellman-Ford algorithm. This is implemented in SageMath as the method `shortest_paths(u, by_weight=True, algorithm="Bellman-Ford_Boost")` of SageMath graph objects. This uses the Bellman-Ford algorithm as implemented in Boost and it does not support weights that are SageMath-specific infinite-precision rational numbers.

By using inexact arithmetic, the shortest paths can only change by a very small amount. Because the shortest path lengths are exactly the diagonal elements of the obtained rescaling matrix, this causes us to have found a slightly different rescaling. The following lemma shows that this rescaling can be at most slightly worse than the one with exact values.

Lemma 29. *Let $D \in \mathbf{D}$ and let $D' \in \mathbf{D}$ such that*

$$D'_{ii} \in \left[\frac{1}{\alpha} D_{ii}, \alpha D_{ii}\right]$$

for every $i \in [n]$ and some $\alpha \geq 1$. Then, $\kappa_{AD'} \leq \alpha^2 \kappa_{AD}$.

Proof. Consider the circuit C and indices i, j such that $\kappa_{AD'} = \kappa_{ij}^{AD'}(C)$. Let $\mathbf{g^{AD}}$ and $\mathbf{g^{AD'}}$ be the elementary vectors corresponding to C such that $\mathbf{g^{AD}} \in \mathrm{Ker}(AD)$ and $\mathbf{g^{AD'}} \in \mathrm{Ker}(AD')$. Then,

$$\kappa_{AD'} = \kappa_{ij}^{AD'}(C) = \left|\frac{g_j^{AD'}}{g_i^{AD'}}\right| \leq \frac{\alpha \left|g_j^{AD}\right|}{\alpha^{-1} \left|g_i^{AD}\right|} \leq \alpha^2 \kappa_{AD}. \qquad \square$$

Even though the original graph is guaranteed to not contain any negative cycles, it might happen that by using inexact arithmetic some cycles with slightly negative weight sum emerge. Shortest paths are well-defined only if no negative cycles exist and the Boost implementation fails if it discovers any such edges. For this reason, we increase all edge weights by a value ε to compensate for the inaccuracies. In this way, all computed shortest path lengths change by at most $n \cdot \varepsilon$.

4. Scaled Instances Performance

As our final contribution, we have evaluated the performance of some practical LP solvers both on the original problem and the problem after applying the approximate optimal rescaling. We have then made an effort to evaluate the impact of the rescaling. We note that while there are some performance guarantees given by Dadush et al. [Dad+20], their relevance in our case is somewhat limited. Firstly, all claims in the article by Dadush et al. [Dad+20] are relevant for the LP solving algorithm by Vavasis and Ye [VY96]. As far as I am aware, none of the tested algorithms directly use this method and to the best of my knowledge, no theoretical claims have been made about the effects of the rescaling for other algorithms.

The second issue is that for problems with large κ^*, the rescaling may be very far from the optimal one and the theoretical complexity bound is therefore weak. However, real-world instances may exhibit nice properties that may cause the rescaling algorithm to be efficient in practice.

Lastly, in Subsection 3.2.1 we have explored the pitfalls of working with the circuit imbalance measure using non-exact arithmetic. In spite of this,we also evaluate the performance of non-exact linear programming algorithms. We do so for completeness and because most LP solvers used in practice do not use exact arithmetic.

4.1 Exact Solvers

Some of the solvers can be configured to find the solution using exact arithmetic. In Subsection 3.2.1 it has been discussed why this will interest us.

However, even the exact solvers read its input from MPS files and at least some of them parse the numeric values in it into floating-point numbers and only then into an infinite precision data type. Because of issues outlined in Subsection 3.2.1, if the exact solver is used naively, problems with precision loss will not be avoided.

Due to properties of the IEEE 754 floating-point number arithmetic used in modern computers, multiplying a number in this arithmetic by a (possibly negative) power of two does not cause precision loss, if the mathematically correct result can be represented as a IEEE 754 floating-point number. Therefore, if we restrict the elements of our rescaling matrix to be such powers of two, the rescaled problem as loaded by the exact solver will be mathematically precisely the original problem with scaled columns of the problem matrix.

As the following lemma shows, if we round down the obtained the values of the rescaling matrix to integral powers of two, we obtain a solution that is not much worse than the original one.

Lemma 30. *Let $D = \mathrm{Diag}(\mathbf{d}) \in \mathbf{D}$. Define $\hat{d}_i$ to be the nearest smaller integral power of two than d_i for every $i \in [n]$ and let $\hat{D} := \mathrm{Diag}(\hat{\mathbf{d}})$. Then,*

$$\kappa_{A\hat{D}} \leq 2 \cdot \kappa_{AD}.$$

The proof of this lemma is almost identical to the proof of Lemma 29, so we do not repeat it.

4.2 Experimental Methodology

We have measured solving times for both original and rescaled instances using the open-source solvers GLPK and SCIP [Bes+21] and using the proprietary solver Gurobi [Gur23]. We have used Python libraries `pyglpk`, `gurobipy` and `pysciopt`. In the case of the exact version of SCIP, we have interfaced with its terminal client using the `pexpect` Python package. In all cases, garbage collection was disabled during the measuring process to keep it from interfering with the measurement. The measured time is the time that the solving method needed to finish, not including problem parsing.

We tried to ensure that any built-in scaling for the solvers is disabled, so that it does not interfere with our scaling, but we did not manage to verify whether the flags for disabling scaling actually worked as expected. We repeat every measurement 10 times and then consider the arithmetic mean to mitigate faulty measurements.

We are mainly interested in the relative effect the rescaling has on the solver run time. Formally speaking, let τ_o be the time in seconds that a solver needs to solve the original instance of the problem and τ_r the time in seconds the solver needs to solve the rescaled instance. Then, define the *time change ratio* of the rescaling as

$$\rho := \left(\frac{\tau_r}{\tau_o} \cdot 100\right) \%.$$

For solving methods with non-exact arithmetic, we measure τ_r considering the precise rescaling, while for methods with exact arithmetic we consider the rescaling rounded to powers of two as described in Section 4.1.

The software versions used to perform the measurements were GLPK v4.65, Gurobi v10.0.2, SCIP v8.0.3 for measurements using non-exact arithmetic and an exact version[1] of SCIP v8.0.0.1 for measurements using exact arithmetic.

For GLPK, we make separate measurements for its simplex solving method, interior point solving method and exact solving method. For Gurobi we only test the default solving method. For SCIP, we measure both with exact arithmetic enabled and disabled. All solver settings not regarding scaling or exact arithmetic are set to their default values.

For measuring original instances, an MPS file created from the problem converted to semi-standard form is used in contrast to using the original MPS files. This is done so that the difference between the two tested problems is solely in column scaling and structural differences of the matrices can not affect performance.

For the computation we used a cluster of computers with identical hardware specifications. The cluster machines are equipped with AMD EPYC 7532 CPU's and they have a SPECfp2017 score of 6.9 per core. The solvers were limited to use a single CPU core.

4.3 Measurement Results

We managed to obtain the approximate optimal rescaling for a total of 142 problem instances of MIPLIB and Netlib. We ran the measurements for all

[1] Available from `https://github.com/scipopt/scip/tree/exact-rational`

solution methods and all problem instances. Several of the measurements did not finish in the alloted time slot, so we exclude these measurements from the results.

Best mean time change ratio was attained for GLPK using an interior-point method with the value 1.32 %. The highest number of problem instances with time change ratio under 100 % was attained for Gurobi with 51 said problem instances. Both these values suggest that the overall impact of the rescaling in the measured cases is neutral at best.

On the other hand, let us point out that the lowest achieved time change ratio is 7.6 %, meaning that applying the rescaling led to 13× shorter runtime of the solver. On a similar note, for 112 of the problem instances the rescaling had time change ratio smaller than 100 % for at least one of the solution methods, causing solving speedup.

We have not observed strong correlations between which instances are sped up for which solvers.

Detailed results of the experiment can be found in Figure 1.1. Columns of the scatter plot correspond to individual problem instances. In addition to time change ratios for all solving methods, we have included the best and the mean time change ratio for every problem instance.

For raw results of measurements for individual problems, see Attachments A.1, A.2 and A.3.

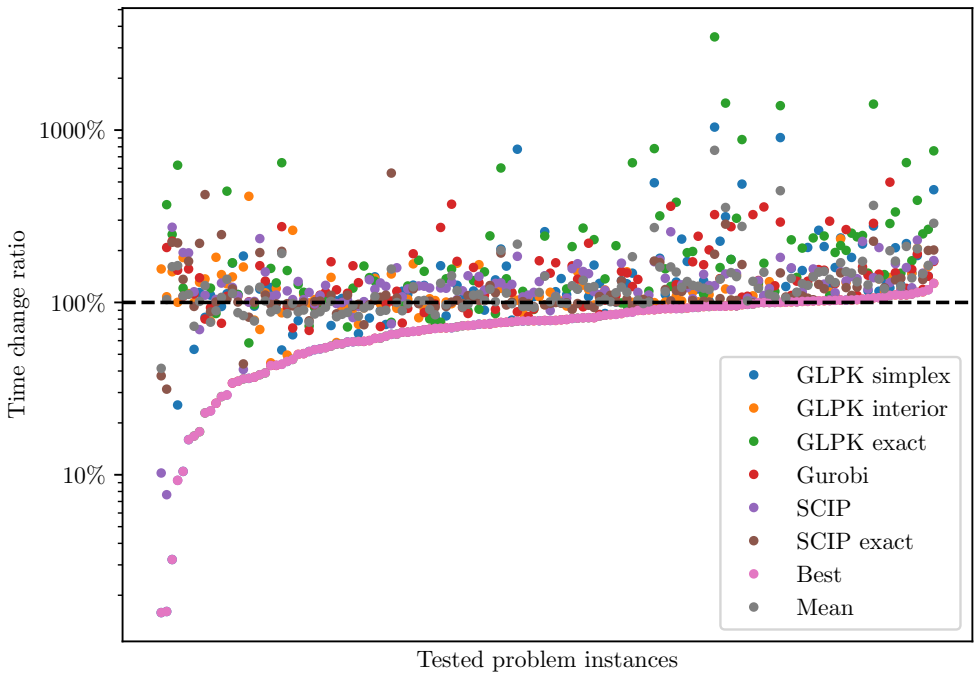


Figure 1.1: A plot of time change ratios for all rescaled instances and all tested LP solving methods. (repeated from page 4)

Conclusion

In this work, we have explored the implementation possibilities for an algorithm by Dadush et al. [Dad+20] for computing an approximate optimal rescaling of an LP optimizing a condition value κ of the constraint matrix. After tackling challenges regarding sparsity of the LP representation, numeric errors and implementing matroid operations, we devised and implemented two possible approaches the algorithm: one offering smaller memory usage, and a second one with higher memory requirements, compensating them by shorter runtime.

These implementations allowed us to compute approximate optimal rescalings of problem instances in the benchmark sets Netlib [Gay85] and MIPLIB [Gle+21]. The size of some of the instances prevented successfully finishing running the demanding algorithm on a big part of the dataset. We have, however, managed to obtain a sufficient amount of rescalings to allow us to run experiments to explore the impact the rescalings have on the performance of real-world LP solvers.

In the experiments, we used solving methods using the LP solver GLPK, methods using SCIP and one method using Gurobi. The evaluated methods are of three kinds: simplex methods based on non-exact arithmetic, interior-point methods based on non-exact arithmetic and simplex methods based on exact arithmetic. The exact solvers were included in the experiment suspecting that non-exact solvers will suffer from sensitivity of κ to numeric errors.

We have measured the ratio in which time to solve an instance changes by applying a rescaling for all the successfully rescaled instances and solving methods. The experiment results are not too encouraging, as for all of the solution methods applying the rescaling has negative impact on runtime for more instances than it has positive effect. Furthermore, average time to solve an instance also rises by applying the rescaling for every method.

While this may be in contrast with the fact that the rescaling was chosen to approximate a rescaling that should reduce the runtime, I believe this result may be caused by a combination of several factors. Firstly, it may be the case that the computed approximations were simply not close enough and instead of reducing the real value of κ, further numerical issues were introduced. This is perfectly possible, as the approximation error is given by a cube of the value κ^*, which can by itself be arbitrarily large. This possible issue might be mitigated by discovering a method for finding a closer approximation of the optimal rescaling.

A second possible factor is that the LP solving method for which a complexity bound on κ is proven is an interior-point method based on exact arithmetic. None of the solving methods used in our experiments is of this kind and using such method might arguably have a better chance of yielding better results.

Lastly, we have only managed to compute the approximate rescaling for problems of fairly small sizes that are mostly easy for the solvers to solve. It might be the case that benefits of applying the rescaling start emerging only when dealing with sufficiently-sized problems.

4.3.1 Further Work

There are several directions in which it is possible to follow up on this work. One such direction may be further looking into implementations of the approximating

algorithm. This thesis does not provide any rigorous observations or measurements regarding the implementations and further research in this direction may be beneficial. Research oriented towards finding heuristics that in practice obtain better approximations than the theory guarantees is certainly also possible.

One more direction to explore is looking for correlations between properties of constraint matrices and benefits of applying the approximate rescaling. If any such correlations were to be discovered, it would be possible to choose candidate problems with greatest potential to benefit from applying the rescaling.

Acknowledgements

Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic. I also gladly acknowledge the support I have received from the project 22-22997S of GA ČR.

Bibliography

[Bes+21] Ksenia Bestuzheva et al. *The SCIP Optimization Suite 8.0*. Technical Report. Optimization Online, Dec. 2021. URL: http://www.optimization-online.org/DB_HTML/2021/12/8728.html.

[Bix12] Robert E. Bixby. *A brief history of linear and mixed-integer programming computation*. 2012. DOI: 10.4171/dms/6/16.

[BGW] Robert K. Brayton, Fred G. Gustavson, and Ralph A. Willoughby. "Some results on sparse matrices". In: 24 (), pp. 937–954. ISSN: 0025-5718. DOI: 10.1090/s0025-5718-1970-0275643-8.

[Dad+20] Daniel Dadush et al. "A scaling-invariant algorithm for linear programming whose running time depends only on the constraint matrix". In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2020. Chicago, IL, USA: Association for Computing Machinery, June 2020, pp. 761–774. ISBN: 9781450369794. DOI: 10.1145/3357713.3384326. URL: https://doi.org/10.1145/3357713.3384326.

[GM07] Bernd Gärtner and Jiří Matoušek. *Understanding and using linear programming*. Universitext. Springer, 2007. ISBN: 978-3-540-30697-9.

[Gay85] Gay, D. M. "Electronic Mail Distribution of Linear Programming Test Problems". In: *Mathematical Programming Society Committee on Algorithms Newsletter* 13 (1985). Data available from https://netlib.org/lp/data/.

[Gle+21] Ambros M. Gleixner et al. "MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library". In: *Math. Program. Comput.* 13.3 (2021), pp. 443–490. DOI: 10.1007/s12532-020-00194-3.

[Gur23] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: https://www.gurobi.com.

[Jän94] Klaus Jänich. *Linear Algebra*. 1994. DOI: 10.1007/978-1-4612-4298-7.

[Kar78] Richard M. Karp. "A characterization of the minimum cycle mean in a digraph". In: 23 (1978), pp. 309–311. ISSN: 0012-365X. DOI: 10.1016/0012-365x(78)90011-0.

[Kha79] Leonid G. Khachiyan. "A polynomial time algorithm in linear programming". In: *Soviet Math. Dokl.* 20 (1979), pp. 191–194.

[Oxl92] J. G. Oxley. *Matroid Theory*. New York: Oxford University Press, 1992.

[The22] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.7)*. https://www.sagemath.org. 2022.

[Sch99] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999. ISBN: 978-0-471-98232-6.

[Tom75] J. A. Tomlin. “On scaling linear programming problems”. In: *Computational Practice in Mathematical Programming*. Ed. by M. L. Balinski and Eli Hellerman. Berlin, Heidelberg: Springer Berlin Heidelberg, 1975, pp. 146–166. ISBN: 978-3-642-00766-8. DOI: 10.1007/BFb0120718. URL: https://doi.org/10.1007/BFb0120718.

[Tun99] Levent Tunçel. “Approximating the complexity measure of Vavasis-Ye algorithm is NP-hard”. In: *Math. Program.* 86.1 (1999), pp. 219–223. DOI: 10.1007/s101070050087.

[VY96] Stephen A. Vavasis and Yinyu Ye. “A primal-dual interior point method whose running time depends only on the constraint matrix”. In: 74 (1996), pp. 79–120. ISSN: 0025-5610. DOI: 10.1007/bf02592148.

List of Abbreviations

DFS	Depth-first search
LP	Linear programming
MIP	Mixed integer programming
RREF	Reduced row echelon form

A. Attachments

A.1 Raw GLPK Timings

Problem	Original/ simplex	Rescaled/ simplex	Original/ interior	Rescaled/ interior	Original/ exact	Rescaled/ exact	Rescaled by powers of 2/exact
MIPLIB problems							
22433	0.0215	0.0186	0.0405	0.0478	2.1793	7.0406	2.9850
23588	0.0096	0.0077	0.0457	0.0420	0.5510	4.7355	1.1617
assign1-5-8	0.0203	0.0120	0.0180	0.0166	3.0127	133.9219	3.6466
b-ball	0.0010	0.0008	0.0008	0.0008	0.0079	0.0103	0.0060
beavma	0.0058	0.0132	0.0055	0.0075	0.1601	0.5865	0.2551
blend2	0.0057	0.0067	0.0080	0.0086	0.4917	1.7490	0.6580
bppc8-02	0.0036	0.0027	0.0163	0.0157	0.1699	0.1847	0.1081
control30-3-2-3	0.0294	0.0351	0.0070	0.0049	10.5879	4.3302	4.0303
dcmulti	0.0084	0.0093	0.0035	0.0034	0.2391	0.5333	0.2864
ej	0.0002	0.0001	0.0001	0.0006	0.0002	0.0001	0.0001
enlight11	0.0003	0.0002	0.0025	0.0024	0.0013	0.0014	0.0010
enlight4	0.0002	0.0002	0.0004	0.0003	0.0002	0.0003	0.0004
enlight8	0.0003	0.0001	0.0011	0.0011	0.0006	0.0009	0.0006
enlight9	0.0003	0.0001	0.0014	0.0014	0.0008	0.0010	0.0007
enlight_hard	0.0004	0.0003	0.0017	0.0019	0.0038	0.0037	0.0037
f2gap40400	0.0035	0.0034	0.0041	0.0059	0.0394	0.6135	0.1213
fhnw-sq2	0.0060	0.0039	0.0070	0.0183	0.8986	6.1749	0.4197
flugpl	0.0004	0.0002	0.0003	0.0003	0.0014	0.0020	0.0010
flugplinf	0.0003	0.0002	0.0004	0.0003	0.0023	0.0029	0.0013
g503inf	0.0005	0.0005	0.0006	0.0011	0.0092	0.0379	0.0079
gen-ip002	0.0007	0.0005	0.0009	0.0008	5.1582	6.6361	5.9827
gen-ip016	0.0007	0.0005	0.0007	0.0007	2.3639	3.4358	3.6892
gen-ip021	0.0008	0.0008	0.0016	0.0009	11.9920	5.7461	6.8744
gen-ip054	0.0007	0.0007	0.0007	0.0007	3.3315	2.0978	1.9132
glass4	0.0027	0.0017	0.0714	0.1044	0.8203	1.1188	0.6289
gr4x6	0.0003	0.0003	0.0004	0.0005	0.0028	0.0025	0.0028
graphdraw-gemcutter	0.0131	0.0185	0.0556	0.0767	0.4780	1.7407	0.6347
gt2	0.0003	0.0007	0.0018	0.0018	0.0053	0.0097	0.0068
ic97_tension	0.0062	0.0076	0.0112	0.0050	0.1778	0.2352	0.2802
k16x240b	0.0024	0.0024	0.0043	0.0045	0.0227	0.0637	0.0229
mad	0.0014	0.0015	0.0112	0.0116	0.0880	3.0784	0.2112
markshare1	0.0002	0.0002	0.0007	0.0008	0.0019	0.0030	0.0021
markshare2	0.0002	0.0002	0.0009	0.0010	0.0029	0.0079	0.0033
markshare_4_0	0.0001	0.0001	0.0004	0.0004	0.0005	0.0016	0.0006
markshare_5_0	0.0001	0.0002	0.0005	0.0006	0.0008	0.0019	0.0008
mas74	0.0013	0.0014	0.0112	0.0088	0.7670	1.0044	0.6986
mas76	0.0012	0.0011	0.0145	0.0108	0.2204	0.4706	0.3316
misc05inf	0.0112	0.0159	0.0637	0.1030	0.3915	20.9772	1.0397
mod008inf	0.0005	0.0006	0.0109	0.0112	0.0140	0.1554	0.0236
neos-1425699	0.0011	0.0010	0.0017	0.0014	0.0123	0.0126	0.0119
neos-2624317-amur	0.0118	0.0087	0.0077	0.0108	7.9862	21.9949	4.4570
neos-2626858-aoos	0.0161	0.0145	0.0137	0.0135	11.6869	32.4151	7.2545
neos-2652786-brda	0.0059	0.0069	0.0069	0.0102	0.4562	1.7868	0.6744
neos-2656603-coxs	0.0048	0.0060	0.0068	0.0115	0.3198	0.5251	0.2307
neos-2657525-crna	0.0063	0.0077	0.0073	0.0102	0.2076	1.8050	0.5954
neos-3046601-motu	0.0147	0.0240	0.0379	0.0461	0.3659	0.7095	0.5018
neos-3046615-murg	0.0149	0.0187	0.0283	0.0338	0.2668	0.4022	0.3943
neos-3072252-nete	0.0085	0.0085	0.0046	0.0041	0.2120	0.2483	0.2116
neos-3610040-iskar	0.0135	0.0142	0.0094	0.0092	0.5592	2.3035	0.4095
neos-3611447-jijia	0.0197	0.0201	0.0124	0.0137	0.7839	0.9845	0.7427
neos-3611689-kaihu	0.0165	0.0160	0.0115	0.0104	0.5268	7.5782	0.6125
neos-3754480-nidda	0.0243	0.0110	0.1583	0.0784	20.0167	51.9247	30.6436
neos-5140963-mincio	0.0106	0.0056	0.0186	0.0185	1.1822	0.2487	0.1983
neos-5192052-neckar	0.0011	0.0012	0.0014	0.0011	0.0210	0.0292	0.0218
neos5	0.0028	0.0023	0.0065	0.0063	0.3012	0.1817	0.4197
neos859080	0.0010	0.0008	0.0044	0.0050	0.0214	0.0727	0.0165
nexp-50-20-1-1	0.0103	0.0265	0.0047	0.0046	0.2927	0.7512	0.7101
noswot	0.0030	0.0035	0.0034	0.0045	0.3692	0.1497	0.0656
opt1217	0.0069	0.0071	0.0061	0.0079	0.1062	1.5610	0.2129
p0201	0.0037	0.0047	0.0095	0.0144	0.2072	0.9206	0.1990
p2m2p1m1p0n100	0.0001	0.0001	0.0005	0.0007	0.0004	0.0004	0.0004
pb-market-split8-70-4	0.0003	0.0003	0.0024	0.0024	0.0047	0.4218	0.0100
pk1	0.0006	0.0008	0.0017	0.0025	0.0241	0.5869	0.0570
ran12x21	0.0054	0.0065	0.0026	0.0026	0.1985	0.1853	0.1858
ran13x13	0.0028	0.0031	0.0018	0.0019	0.0907	0.0850	0.0709
rlp1	0.0045	0.0042	0.0030	0.0036	0.0539	0.1915	0.0700
rout	0.0070	0.0112	0.0113	0.0123	0.3643	9.7253	0.7100
sp150x300d	0.0060	0.0066	0.0071	0.0055	0.1394	0.1388	0.1688
stein15inf	0.0004	0.0004	0.0006	0.0007	0.0063	0.0063	0.0065
stein45inf	0.0119	0.0099	0.0598	0.0601	0.5071	1.0171	0.6953
stein9inf	0.0002	0.0002	0.0003	0.0002	0.0010	0.0013	0.0006
supportcase14	0.0054	0.0048	0.0054	0.0057	0.2018	1.8468	0.1619
supportcase16	0.0030	0.0032	0.0044	0.0052	0.1122	0.4465	0.0915
timtab1	0.0022	0.0036	0.0077	0.0075	0.0387	0.1009	0.0895
timtab1CUTS	0.0148	0.0178	0.0264	0.0397	0.8703	15.2763	0.7048

Continued on next page

Problem	Original/ simplex	Rescaled/ simplex	Original/ interior	Rescaled/ interior	Original/ exact	Rescaled/ exact	Rescaled by powers of 2/exact
			Netlib problems				
25FV47	0.2681	0.0429	0.0973	0.1096	2295.5904		
ADLITTLE	0.0013	0.0018	0.0010	0.0011	0.0329	0.3993	0.0478
AFIRO	0.0002	0.0003	0.0003	0.0003	0.0007	0.0016	0.0015
AGG	0.0037	0.0166	0.0307	0.0397	0.1276	2.1830	0.9667
AGG2	0.0034	0.0310	0.0393	0.0393	0.1202	12.1306	1.6656
AGG3	0.0036	0.0019	0.0439	0.0382	0.1383	2.3668	0.8945
BANDM	0.0304	0.0284	0.0110	0.0126	7.8485	146.4538	15.5079
BEACONFD	0.0025	0.0054	0.0087	0.0096	0.0447	7.7300	0.2892
BLEND	0.0010	0.0018	0.0014	0.0014	0.0297		0.0749
BNL1	0.0761	0.1018	0.0458	0.0395	15.0224	441.1302	14.9258
BOEING1	0.0144	0.0341	0.0349	0.0816	1.1725	6.6550	2.5074
BOEING2	0.0028	0.0053	0.0082	0.0169	0.1116	0.7452	0.4368
BORE3D	0.0038	0.0042	0.0089	0.0114	0.1435	0.0634	0.6347
BRANDY	0.0126	0.0174	0.0081	0.0083			
CAPRI	0.0068	0.0112	0.0134	0.0095	1.4671	5.8932	1.1274
DEGEN2	0.0320	0.0475	0.0287	0.0423	1.9854	45.2887	3.4645
DEGEN3	0.4826	0.7135	0.5645	0.5651	135.9188	12460.6241	878.2734
E226	0.0098	0.0258	0.0104	0.0106	2.2084	76.4171	
ETAMACRO	0.0140	0.0184	0.0257	0.0279	0.4996	40.1687	1.9062
FFFFF800	0.0243	0.0329	0.0957	0.0727	5.9050	13.0782	5.0861
FINNIS	0.0126	0.0234	0.0149	0.0239	1.3846	0.6259	0.4987
FIT1D	0.0279	0.0336	0.1973	0.2389	10.0379	815.4271	24.4242
FIT1P	0.0760	0.0847	0.8835	0.9226	14.1801	50.1393	17.6639
FORPLAN	0.0063	0.0491	0.0165	0.0171	4.0903		
GANGES	0.0681	0.0836	0.0418	0.0524	2.1474	10.1157	2.7896
GFRD-PNC	0.0209	0.0223	0.0083	0.0071	0.5884	0.7076	0.6395
GROW15	0.0271	0.0257	0.0175	0.0253	251.3215	1712.2708	71.4001
GROW22	0.0809	0.0663	0.0254	0.0348	907.7583	4859.8624	207.3977
GROW7	0.0080	0.0060	0.0076	0.0126	19.9438	204.4021	18.2176
ISRAEL	0.0038	0.0084	0.0233	0.0244	0.4246	25.0247	1.0386
KB2	0.0009	0.0012	0.0012	0.0013	0.1346	0.3389	0.1121
LOTFI	0.0022	0.0050	0.0038	0.0038	0.1144	0.3448	0.1055
MAROS	0.1901	0.0030	0.0582	0.0910	32.0731	140.5513	
MODSZK1	0.0581	0.0455	0.0329	0.0356			
PEROLD	0.2104	0.0068	0.0706	0.1063	10000.6069	102583.0157	24811.1619
PILOT4	0.0984	0.0103	0.0573	0.1044	16720.8471	23104.4407	20236.8544
RECIPE	0.0004	0.0004	0.0013	0.0012	0.0050	0.0054	0.0062
SC105	0.0010	0.0013	0.0008	0.0008	0.0235	0.1192	0.0247
SC205	0.0040	0.0045	0.0014	0.0014	0.1060	3.1341	0.1660
SC50A	0.0004	0.0005	0.0004	0.0004	0.0049	0.0219	0.0055
SC50B	0.0004	0.0004	0.0004	0.0004	0.0050	0.0233	0.0069
SCAGR25	0.0168	0.0354	0.0035	0.0036	0.7996	27.1560	1.9427
SCAGR7	0.0020	0.0035	0.0013	0.0012	0.0463	0.3373	0.0823
SCFXM1	0.0121	0.0251	0.0100	0.0111	0.6151	20.1405	2.0601
SCFXM2	0.0378	0.0096	0.0230	0.0230	2.2007	47.7846	13.7650
SCFXM3	0.0738	0.0012	0.0327	0.0353	5.4403	48.2828	20.0830
SCORPION	0.0074	0.0093	0.0036	0.0061	0.2034	0.1740	0.2478
SCRS8	0.0287	0.0367	0.0113	0.0129	4.6556	20.3965	1.8168
SCSD1	0.0028	0.0290	0.0029	0.0028	0.5358	22.0673	18.6108
SCSD6	0.0096	0.0267	0.0052	0.0055	1.1599	51.5837	16.4151
SCTAP1	0.0095	0.0137	0.0050	0.0058	0.1897	1.4933	0.4739
SEBA	0.0181	0.0286	0.1551	0.1562	0.4987	1.6321	1.1513
SHARE1B	0.0048	0.0065	0.0026	0.0033	0.5948	16.6080	1.6061
SHARE2B	0.0011	0.0019	0.0018	0.0016	0.0530	0.7711	0.1031
SHELL	0.0189	0.0149	0.0135	0.0068	0.3994	1.3161	0.5080
SHIP04L	0.0118	0.0137	0.0141	0.0129	0.5411	0.5969	0.5449
SHIP04S	0.0105	0.0106	0.0086	0.0080	0.2664	0.3638	0.4333
SHIP08S	0.0242	0.0304	0.0145	0.0133	0.7836	1.0948	1.1876
SHIP12S	0.0549	0.0580	0.0182	0.0182	1.7979	2.4589	1.8187
STAIR	0.0240	0.0433	0.0408	0.0385	142.3757	9135.2192	452.4875
STANDATA	0.0017	0.0084	0.0127	0.0127	0.0367	0.3480	0.2864
STANDGUB	0.0017	0.0081	0.0132	0.0128	0.0333	0.3634	0.2937
STANDMPS	0.0071	0.0145	0.0241	0.0291	0.1290	0.7027	0.7777
STOCFOR1	0.0010	0.0017	0.0012	0.0018	0.0152	0.0432	0.0258
STOCFOR2	0.1720	0.5404	0.0384	0.0460	6.5906	2809.1620	94.5011
TUFF	0.0165	0.0192	0.0204	0.0279			
VTP.BASE	0.0042	0.0046	0.0074	0.0055	0.1503	0.4712	0.1339

A.2 Raw Gurobi Timings

Problem	Original	Rescaled
MIPLIB problems		
22433	0.0092	0.0094
23588	0.0044	0.0071
assign1-5-8	0.0137	0.0224
b-ball	0.0009	0.0009
beavma	0.0028	0.0039
blend2	0.0031	0.0050
bppc8-02	0.0023	0.0022
control30-3-2-3	0.0115	0.0188
dcmulti	0.0077	0.0061
ej	0.0004	0.0004
enlight11	0.0005	0.0005
enlight4	0.0005	0.0003
enlight8	0.0017	0.0016
enlight9	0.0004	0.0004
enlight_hard	0.0004	0.0006
f2gap40400	0.0022	0.0021
fhnw-sq2	0.0066	0.0047
flugpl	0.0006	0.0004
flugplinf	0.0004	0.0005
g503inf	0.0030	0.0008
gen-ip002	0.0011	0.0010
gen-ip016	0.0009	0.0032
gen-ip021	0.0010	0.0011
gen-ip054	0.0008	0.0008
glass4	0.0013	0.0014
gr4x6	0.0007	0.0022
graphdraw-gemcutter	0.0111	0.0069
gt2	0.0007	0.0010
ic97_tension	0.0048	0.0021
k16x240b	0.0039	0.0035
mad	0.0011	0.0013
markshare1	0.0014	0.0005
markshare2	0.0011	0.0006
markshare_4_0	0.0004	0.0006
markshare_5_0	0.0005	0.0015
mas74	0.0017	0.0015
mas76	0.0017	0.0019
misc05inf	0.0033	0.0047
mod008inf	0.0006	0.0021
neos-1425699	0.0019	0.0007
neos-2624317-amur	0.0049	0.0084
neos-2626858-aoos	0.0149	0.0108
neos-2652786-brda	0.0041	0.0058
neos-2656603-coxs	0.0018	0.0031
neos-2657525-crna	0.0020	0.0099
neos-3046601-motu	0.0023	0.0018
neos-3046615-murg	0.0016	0.0018
neos-3072252-nete	0.0016	0.0021
neos-3610040-iskar	0.0064	0.0060
neos-3611447-jijia	0.0053	0.0107
neos-3611689-kaihu	0.0068	0.0066
neos-3754480-nidda	0.0127	0.0119
neos-5140963-mincio	0.0011	0.0012
neos-5192052-neckar	0.0008	0.0014
neos5	0.0045	0.0027
neos859080	0.0006	0.0007
nexp-50-20-1-1	0.0057	0.0045
noswot	0.0013	0.0018
opt1217	0.0038	0.0100
p0201	0.0016	0.0023
p2m2p1m1p0n100	0.0005	0.0006
pb-market-split8-70-4	0.0007	0.0008
pk1	0.0011	0.0012
ran12x21	0.0036	0.0086
ran13x13	0.0035	0.0029
rlp1	0.0047	0.0020
rout	0.0074	0.0095
sp150x300d	0.0032	0.0030
stein15inf	0.0005	0.0005
stein45inf	0.0037	0.0026
stein9inf	0.0005	0.0006
supportcase14	0.0039	0.0027
supportcase16	0.0019	0.0041
timtab1	0.0012	0.0010
timtab1CUTS	0.0046	0.0063
Netlib problems		
25FV47	0.1550	0.2424
ADLITTLE	0.0013	0.0014
AFIRO	0.0005	0.0005
AGG	0.0047	0.0082
AGG2	0.0033	0.0095
AGG3	0.0038	0.0104
BANDM	0.0103	0.0177
BEACONFD	0.0039	0.0047
BLEND	0.0014	0.0020
BNL1	0.0363	0.0549

Continued in next column

Problem	Original	Rescaled
BOEING1	0.0154	0.0159
BOEING2	0.0022	0.0042
BORE3D	0.0030	0.0033
BRANDY	0.0106	0.0084
CAPRI	0.0034	0.0093
DEGEN2	0.0164	0.0255
DEGEN3	0.1555	0.1922
E226	0.0091	0.0141
ETAMACRO	0.0127	0.0161
FFFFF800	0.0166	0.0151
FINNIS	0.0073	
FIT1D	0.0066	0.0050
FIT1P	0.0325	0.0300
FORPLAN	0.0068	0.0060
GANGES	0.0138	0.0185
GFRD-PNC	0.0114	0.0119
GROW15	0.0685	0.0518
GROW22	0.1466	0.1174
GROW7	0.0101	0.0131
ISRAEL	0.0032	0.0051
KB2	0.0009	0.0007
LOTFI	0.0118	0.0111
MAROS	0.0651	
MODSZK1	0.0048	0.0083
PEROLD	0.0754	0.1710
PILOT4	0.0422	0.0458
RECIPE	0.0009	0.0034
SC105	0.0019	0.0019
SC205	0.0046	0.0042
SC50A	0.0007	0.0012
SC50B	0.0007	0.0010
SCAGR25	0.0120	0.0184
SCAGR7	0.0026	0.0072
SCFXM1	0.0089	0.0133
SCFXM2	0.0187	0.0287
SCFXM3	0.0306	0.0637
SCORPION	0.0037	0.0071
SCRS8	0.0114	0.0151
SCSD1	0.0020	0.0064
SCSD6	0.0069	0.0199
SCTAP1	0.0044	0.0052
SEBA	0.0021	0.0023
SHARE1B	0.0027	0.0025
SHARE2B	0.0015	0.0019
SHELL	0.0112	
SHIP04L	0.0075	0.0057
SHIP04S	0.0069	0.0041
SHIP08S	0.0094	0.0082
SHIP12S	0.0119	0.0137
STAIR	0.0254	0.0233
STANDATA	0.0036	0.0033
STANDGUB	0.0030	0.0033
STANDMPS	0.0072	0.0055
STOCFOR1	0.0055	0.0019
STOCFOR2	0.0816	0.0777
TUFF	0.0052	0.0076
VTP.BASE	0.0081	0.0070

A.3 Raw SCIP Timings

Problem	Original/ non-exact	Rescaled/ non-exact	Original/ exact	Original by powers of 2/exact
MIPLIB problems				
22433	0.0326	0.0383	0.2692	0.2269
23588	0.0177	0.0279	0.1995	0.1820
assign1-5-8	0.0557	0.0455	0.3081	0.2934
b-ball	0.0214	0.0151	0.0813	0.0764
beavma	0.0154	0.0147	0.1509	0.1588
blend2	0.0281	0.0207	0.1372	0.1473
bppc8-02	0.0214	0.0266	0.2270	0.2501
control30-3-2-3	0.0281	0.0659	0.2500	0.4868
dcmulti	0.0211	0.0224	0.2002	0.1992
ej	0.0006	0.0006	0.0947	0.0778
enlight11	0.0019	0.0020	0.0683	0.0752
enlight4	0.0007	0.0009	0.0573	0.0565
enlight8	0.0014	0.0014	0.0606	0.0613
enlight9	0.0016	0.0018	0.0625	0.0636
enlight_hard	0.0021	0.0019	0.0640	0.0632
f2gap40400	0.0088	0.0130	0.1092	0.1121
fhnw-sq2	0.0270	0.0281	0.1509	0.1440
flugpl	0.0024	0.0028	0.0861	0.0674
flugplinf	0.0026	0.0025	0.0696	0.0679
g503inf	0.0029	0.0030	0.0739	0.0743
gen-ip002	0.0087	0.0094	0.1482	0.1498
gen-ip016	0.0043	0.0056	0.1417	0.1423
gen-ip021	0.0097	0.0095	0.1806	0.1552
gen-ip054	0.0050	0.0050		
glass4	0.0317	0.0238	0.2836	1.5969
gr4x6	0.0026	0.0026	0.0740	0.0734
graphdraw-gemcutter	0.0119	0.0142	0.2594	0.2670
gt2	0.0052	0.0053	0.0843	0.0892
ic97_tension	0.0201	0.0218	0.1601	0.1558
k16x240b	0.0082	0.0085	0.1298	0.1349
mad	0.0301	0.0413	0.1485	0.1821
markshare1	0.0033	0.0037	0.0740	0.0744
markshare2	0.0038	0.0039	0.0784	0.0787
markshare_4_0	0.0026	0.0028	0.0706	0.0706
markshare_5_0	0.0029	0.0030	0.0717	0.0736
mas74	0.0155	0.0178	0.1365	0.1290
mas76	0.0143	0.0169	0.1083	0.1109
misc05inf	0.0363	0.0431	0.1820	0.3646
mod008inf	0.0088	0.0096	0.1212	0.1211
neos-1425699	0.0043	0.0044	0.0880	0.0880
neos-2624317-amur	0.0246	0.0253	0.1675	0.1843
neos-2626858-aoos	0.0252	0.0327	0.1715	0.1839
neos-2652786-brda	0.0183	0.0219	0.1612	0.1696
neos-2656603-coxs	0.0176	0.0183	0.1643	0.1691
neos-2657525-crna	0.0169	0.0221	0.1643	0.1806
neos-3046601-motu	0.0128	0.0155	0.1850	0.1787
neos-3046615-murg	0.0110	0.0128	0.1636	0.2261
neos-3072252-nete	0.0164	0.0187	0.3349	0.3090
neos-3610040-iskar	0.0161	0.0188	0.1664	0.1629
neos-3611447-jijia	0.0147	0.0185	0.1561	0.1571
neos-3611689-kaihu	0.0174	0.0192	0.1494	0.1519
neos-3754480-nidda	0.0211	0.0246	0.3072	0.3023
neos-5140963-mincio	0.0083	0.0096	0.1445	0.1370
neos-5192052-neckar	0.0067	0.0073	0.0936	0.0935
neos5	0.0106	0.0144	0.1327	0.1284
neos859080	0.0120	0.0153	0.1121	0.1125
nexp-50-20-1-1	0.0101	0.0144	0.1397	0.1352
noswot	0.0158	0.0110	0.1265	0.2786
opt1217	0.0176	0.0234	0.1559	0.1901
p0201	0.0191	0.0167	0.1211	0.1268
p2m2p1m1p0n100	0.0072	0.0086	0.0715	0.0714
pb-market-split8-70-4	0.0125	0.0106	0.0927	0.0933
pk1	0.0099	0.0155	0.0994	0.1030
ran12x21	0.0088	0.0109	0.1479	0.1432
ran13x13	0.0062	0.0068	0.1146	0.1129
rlp1	0.0535	0.0590	0.1254	0.1230
rout	0.0635	0.0724	0.3618	0.3723
sp150x300d	0.0513	0.0120	0.1451	0.1280
stein15inf	0.0074	0.0078	0.0735	0.0719
stein45inf	0.0162	0.0196	0.1333	0.1262
stein9inf	0.0067	0.0107	0.0682	0.0724
supportcase14	0.0123	0.0149	0.1154	0.1174
supportcase16	0.0106	0.0156	0.1157	0.1114
timtab1	0.0073	0.0085	0.1161	0.1085
timtab1CUTS	0.0211	0.0355	0.1860	0.1833

Continued on next page

Problem	Original/ non-exact	Rescaled/ non-exact	Original/ exact	Original by powers of 2/exact
		Netlib problems		
25FV47	0.7653	1.4841	1.5760	2.7314
ADLITTLE	0.0074	0.0080	0.0817	0.0993
AFIRO	0.0027	0.0031	0.0680	0.0742
AGG	0.0104	0.0183	0.1275	0.2569
AGG2	0.0185	0.0338	0.1971	0.2923
AGG3	0.0171	0.0075	0.1908	0.3777
BANDM	0.0275	0.0289	0.2778	0.2849
BEACONFD	0.0076	0.0120	0.1128	0.1253
BLEND	0.0069	0.0098	0.0929	0.1020
BNL1	0.0881	0.1489	0.4276	0.5704
BOEING1	0.0320	0.0483	0.1718	0.2549
BOEING2	0.0076	0.0175	0.1129	0.1290
BORE3D	0.0085	0.0025	0.1138	0.1191
BRANDY	0.0296		0.1725	0.1707
CAPRI	0.0178	0.0217	0.1837	
DEGEN2	0.0612	0.0745	0.2351	0.2632
DEGEN3	1.0368	0.9119	1.6931	1.6204
E226	0.0376	0.0567	0.1928	0.2454
ETAMACRO	0.0353	0.0325	0.1989	0.2335
FFFFF800	0.1420	0.1833	0.4280	0.4865
FINNIS	0.0323	0.0132	0.1852	0.0815
FIT1D	0.1212	0.0915	0.3790	0.4752
FIT1P	0.2504	0.3000	1.9414	1.3051
FORPLAN	0.0303	0.0561	0.3448	0.2688
GANGES	0.0646	0.0861	0.4241	0.4229
GFRD-PNC	0.0287	0.0309	0.3315	0.3649
GROW15	0.0822	0.1065	1.0074	2.4927
GROW22	0.2204	0.2427	1.2943	5.4675
GROW7	0.0280	0.0396	0.6239	0.6402
ISRAEL	0.0166	0.0375	0.1356	0.2086
KB2	0.0039	0.0043	0.0756	0.0820
LOTFI	0.0097	0.0123	0.1291	0.1496
MAROS	0.1011	0.0104	0.7263	0.2725
MODSZK1	0.0327	0.0432		
PEROLD	0.3250	0.8875	1.2975	2.8978
PILOT4	0.1260	0.2454	1.0170	1.3846
RECIPE	0.0084	0.0216	0.0802	0.0819
SC105	0.0049	0.0051	0.0868	0.0933
SC205	0.0067	0.0093	0.1138	0.1353
SC50A	0.0028	0.0041	0.0748	0.0760
SC50B	0.0028	0.0032	0.0732	0.0738
SCAGR25	0.0167	0.0214	0.2037	0.2445
SCAGR7	0.0057	0.0067	0.0964	0.0997
SCFXM1	0.0291	0.0488	0.2231	0.3018
SCFXM2	0.0632	0.0059	0.4573	1.0126
SCFXM3	0.1045	0.0080	0.7879	0.2473
SCORPION	0.0097	0.0066	0.1141	0.1367
SCRS8	0.0433	0.0651	0.3802	0.4198
SCSD1	0.0210	0.0263	0.3105	0.5913
SCSD6	0.0463	0.0644	0.6068	1.3746
SCTAP1	0.0179	0.0264	0.1928	0.2617
SEBA	0.0112	0.0178	0.1799	0.1921
SHARE1B	0.0135	0.0204	0.1271	0.1035
SHARE2B	0.0077	0.0094	0.0940	0.1201
SHELL	0.0337	0.0446	0.4133	0.3806
SHIP04L	0.0477	0.0514	0.9875	0.6637
SHIP04S	0.0315	0.0318	0.5239	0.3887
SHIP08S	0.0449	0.0558	0.8788	0.6103
SHIP12S	0.0502	0.0631	1.3144	0.7179
STAIR	0.0565	0.0998	0.9555	1.6282
STANDATA	0.0179	0.0308	0.1655	0.2878
STANDGUB	0.0252	0.0247	0.1912	0.3172
STANDMPS	0.0252	0.0328	0.2319	0.4484
STOCFOR1	0.0058	0.0070	0.0867	0.1039
STOCFOR2	0.1105	0.1617	0.4977	1.4164
TUFF	0.0440	0.0916	0.2675	0.2850
VTP.BASE	0.0052	0.0074	0.0857	0.0938