Computing a Nonnegative Dyadic Solution to a System of Linear Equations

Gérard Cornuéjols, Anthony Karahalios, Vrishabh Patil*
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213, USA

A rational number is *dyadic* if it is of the form $p/2^k$, where $p \in \mathbb{Z}$, $k \in \mathbb{Z}_+$. In this paper, we present the first implementation of the polynomial-time algorithm of Abdi et al. (2024) for computing a nonnegative dyadic solution to a system of linear equations. Our implementation reveals that this baseline algorithm often produces dense solutions with relatively large denominators. To address this, we introduce algorithmic refinements that incorporate a unimodular matrix reduction, a column generation procedure, and a bounded search over the denominator exponent k, which significantly improve sparsity and reduce k. Computational experiments on random 0–1 systems, integer-infeasible MIPLIB instances, and combinatorial perfect-matching covering instances show that the proposed methods produce exact solutions that are often more compact than floating-point and exact-rational baselines.

Key words: linear programming, integer programming, dyadic rational, floating-point arithmetic, exact computations, Hermite normal form

1. Introduction

Computational approaches to solving optimization problems often require approximating rational numbers with floating-point representations to perform arithmetic operations efficiently. However, such approximations can introduce rounding errors that may accumulate across successive computations. In particular, finite-precision arithmetic introduces numerical errors in linear algebra subroutines such as LU or Cholesky factorizations, operations used by most solvers (Gärtner 1999, Higham 2002, Applegate et al. 2007, Cook et al. 2013). As a result, computed solutions typically satisfy constraints only up to small residuals. In particular, equality constraints may not be satisfied exactly. While this is often acceptable in practice, it becomes problematic when solutions serve as proofs that certain logical statements hold true.

Exact rational solvers provide a natural alternative, producing solutions that satisfy constraints exactly (Espinoza 2006, Applegate et al. 2007, Gleixner et al. 2016, Gleixner and Steffy 2020, Eifler et al. 2025). However, exactness alone does not guarantee practicality. Rational solutions may involve extremely large numerators and denominators, leading to large encoding sizes (*i.e.*,

* Corresponding author at: 4765 Forbes Ave, Pittsburgh, PA 15213, United States of America.
E-mail address: vmpatil@andrew.cmu.edu.

large numerator and denominator bit-lengths) and binary representations that are difficult to store, interpret, or communicate. This creates a fundamental tradeoff between computational efficiency, exact certification, and representational simplicity.

This paper investigates a structured form of exact certification based on *dyadic* rationals, i.e., numbers of the form $p/2^k$ for integers p and $k \geq 0$. Dyadic rationals are of particular interest because they have finite binary representations and can be represented exactly in standard floating-point arithmetic, thereby avoiding approximation errors. Beyond their computational appeal, dyadic solutions arise naturally as controlled relaxations of integer feasibility. In some applications, linear programming relaxations are feasible while integer solutions do not exist. In such cases, one may ask whether solutions with bounded denominator exponent (e.g., half- or quarter-integral solutions) can be obtained. Dyadic solutions thus provide a natural intermediate notion between integral and arbitrary rational feasibility.

Dyadic solutions are also of independent theoretical interest. We highlight the following open question of Seymour (see Schrijver et al. 2003, 79.3e). Let A be a 0–1 matrix such that the set covering polyhedron $Ax \geq \mathbf{1},\ x \geq 0$ has only integral vertices. Then, for any $c \in \mathbb{Z}^n_+$, the linear program $\min\{c^\top x : Ax \geq \mathbf{1},\ x \geq 0\}$ has an optimal 0–1 solution. Seymour conjectured that the dual linear program $\max\{\mathbf{1}^\top y : A^\top y \leq c,\ y \geq 0\}$ always admits a dyadic optimal solution.

We study the problem of computing exact, nonnegative dyadic solutions to linear systems of equations. Given $A \in \mathbb{Z}^{m\times n}$ and $b \in \mathbb{Z}^m$, the *Nonnegative Dyadic Feasibility Problem* consists of producing one of the following outcomes:

(i) a dyadic vector $x \in \mathbb{Q}^n$ with $x \geq 0$ satisfying $Ax = b$ exactly;

(ii) a certificate that the system $Ax = b$, $x \geq 0$ has no real solution;

(iii) a certificate that the system admits a real solution but no dyadic solution.

While dyadic solutions without sign constraints can be obtained via classical techniques such as Hermite normal form, the nonnegativity constraint introduces additional complexity. Nevertheless, Abdi et al. (2024) showed that existence and construction remain polynomial-time solvable, in contrast to the corresponding integer-feasibility problem, which is NP-complete.

Despite this theoretical result, no computational implementation has previously been developed or evaluated. To the best of our knowledge, this work provides the first practical algorithm for solving the Nonnegative Dyadic Feasibility Problem. Our computational study reveals a key limitation of the baseline method of Abdi et al. (2024): the resulting solutions are often dense and may exhibit large denominator exponents. Thus, while the problem inherits the polynomial-time solvability of linear programming, it also exhibits characteristics associated with integer programming, such as dense and computationally cumbersome solutions.

To address this, we develop algorithmic refinements that promote sparsity and reduce denominator size while preserving exact feasibility. Our approach combines column-generation techniques with a bounded search over dyadic denominator exponents, enabling control over both support and fractionality.

The motivation for finding dyadic solutions with small bit length is illustrated by a representative example. Consider a randomly generated 0–1 matrix $A \in \{0,1\}^{100\times 300}$ with density 0.25 and the system $Ax = \mathbf{1},\ x \geq 0$. Solving the system in double precision yields entries such as 0.04722072915708268, which correspond to rationals with denominator 2^{64} and do not satisfy $Ax = \mathbf{1}$ exactly. Exact rational solvers produce feasible solutions with very large entries, such as

$$\frac{1066691814589478587756657784513158546543333828298}{876757114929839476859217243499125727719738719471},$$

whose denominator requires roughly 160 bits to represent. This solution is difficult to interpret, store in memory, and use in arithmetic operations.

In contrast, our implementation of the dyadic algorithm produces exact solutions with compact representations, with entries such as $3007/2^{19}$. The refinements developed in this paper further improve this to yield solutions with entries such as

$$\frac{217}{4096} = \frac{217}{2^{12}}$$

while preserving exact feasibility.

The remainder of the paper is organized as follows. Sections 2 and 3 present the baseline algorithm and implementation details. Section 4 develops algorithmic refinements to the baseline method of Abdi et al. (2024). We include a preliminary experimental study for each proposed improvement. Section 5 reports computational results on random, MIPLIB integer-infeasible, and combinatorial instances. We conclude with a discussion of implications and future directions.

2. A Baseline Algorithm for Nonnegative Dyadic Feasibility

In this section, we present the *baseline algorithm* of Abdi et al. (2024) for the Nonnegative Dyadic Feasibility Problem, together with key implementation choices that are necessary for obtaining exact solutions in practice. We refer the reader to the original work for proofs of correctness and complexity guarantees. Let $P := \{x \in \mathbb{R}^n : Ax = b,\ x \geq 0\}$, where $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^m$.

2.1. Overview

The baseline algorithm can be viewed as consisting of two conceptual phases. First, it constructs an explicit dyadic point in the affine hull of P. Second, if that point is not already nonnegative, it applies a correction along the nullspace of A that preserves the feasibility $Ax = b$ and dyadicness while moving the point into the nonnegative orthant.

The algorithm first determines whether P is nonempty using standard linear feasibility techniques with exact rationals. If $P = \emptyset$, it returns a certificate of infeasibility. Otherwise, the algorithm proceeds with a preprocessing step that identifies the implicit equalities of P other than $Ax = b$. These are of the form $x_j = 0$ for a subset $J^= \subseteq [n]$, yielding the explicit description $\operatorname{aff}(P) = \{x : Ax = b,\ x_j = 0\ \forall j \in J^=\}$. Next, we describe the two phases.

Phase I: Find a dyadic point in $\operatorname{aff}(P)$*.* The procedure either (i) returns a dyadic point $\bar{x} \in \operatorname{aff}(P)$, or (ii) certifies that $\operatorname{aff}(P)$ contains no dyadic point. This step is performed using Hermite normal form computations and an associated unimodular transformation.

If the dyadic point $\bar{x}$ returned by Phase I satisfies $\bar{x} \geq 0$, then $\bar{x} \in P$ and we have found a feasible dyadic solution. Otherwise, Abdi et al. (2024) show that, provided the relative interior $\operatorname{relint}(P)$ is nonempty, P contains a nonnegative dyadic point x^*. Phase II constructs such a point from $\bar{x}$ by moving along directions in the nullspace of A.

Phase II: Move into the nonnegative orthant. If $\bar{x}$ has negative components, the algorithm constructs a dyadic correction $\bar{\rho}$ that satisfies $A\bar{\rho} = 0$ so that $x^* = \bar{x} + \bar{\rho}$ is nonnegative. Such a dyadic correction is constructed as follows.

First, it finds a rational point $x^{\mathrm{rint}} \in \operatorname{relint}(P)$ and the largest radius $\epsilon > 0$ such that the ∞-norm ball B of radius ϵ centered at x^{rint} is contained in P, namely $B := \{x : \|x - x^{\mathrm{rint}}\|_\infty \leq \epsilon\} \subset P$. Let $J^< = [n] \setminus J^=$ and $A^<$ denote the submatrix of A indexed by $J^<$. A point $x^{\mathrm{rint}} = \zeta$ and a radius ϵ can be obtained by solving the following linear program exactly.

$$
\begin{aligned}
\max \quad & \epsilon \\
\text{s.t.} \quad & A^< \zeta = b \\
& \epsilon \leq \zeta_j \qquad j \in J^< \\
& \epsilon \leq 1,
\end{aligned}
\tag{1}
$$

where the last constraint ensures boundedness.

The algorithm then computes an integral basis $d^1, \ldots, d^\ell$ for the constrained nullspace $\{d : Ad = 0,\ d_j = 0\ \forall j \in J^=\}$. The vector $x^{\mathrm{rint}} - \bar{x}$ is then represented in this nullspace basis. More specifically, we compute coefficients $\alpha \in \mathbb{Q}^\ell$ such that

$$\sum_{i=1}^{\ell} \alpha_i d^i = x^{\mathrm{rint}} - \bar{x}.$$

These coefficients are then rounded down to the dyadic grid of width $1/2^{r^*}$, yielding

$$\bar{\rho} = \sum_{i=1}^{\ell} \frac{\lfloor 2^{r^*} \alpha_i \rfloor}{2^{r^*}} d^i,$$

where the exponent r^* is chosen as a function of ϵ and the norms of the basis vectors. In particular, Abdi et al. (2024) show that one may choose

$$r^* = \left\lceil \log_2\left(\frac{\ell \max\{\|d^1\|_\infty, \ldots, \|d^\ell\|_\infty\}}{\epsilon}\right)\right\rceil.$$

The resulting correction $\bar{\rho}$ is dyadic and satisfies $A\bar{\rho} = 0$. Moreover, by the choice of r^*, the point $x^* = \bar{x} + \bar{\rho}$ remains within the ball B centered at x^{rint}, and hence belongs to P.

Algorithm 1 states this procedure in pseudocode. For completeness, the pseudocode for the auxiliary routines is provided in Appendix A.

Algorithm 1 Baseline algorithm as in (Abdi et al. 2024, Section 3.1.2)

Input: $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^m$

Output: a dyadic $x^* \in P$, or a certificate that $P = \emptyset$, or a certificate that $\text{aff}(P)$ contains no dyadic point.

1: (status, $\bar{x}$, U) $\leftarrow$ `findDyadicAffineSolution`(A, b)
2: **if** status = INFEASIBLE **then**
3:     **return** certificate that $P = \emptyset$
4: **else if** status = NO-DYADIC **then**
5:     **return** certificate that $\text{aff}(P)$ contains no dyadic point
6: **if** $\bar{x} \geq 0$ **then**
7:     **return** $\bar{x}$
8: $J^= \leftarrow$ `findImplicitEqualities`(A, b)
9: Compute an integral basis $\{d^1, \ldots, d^\ell\}$ of $\{d : Ad = 0,\ d_j = 0\ \forall j \in J^=\}$
10: $x^{\text{rint}}, \epsilon \leftarrow$ `computeInteriorPoint`$(A, b, J^=)$
11: Choose $r^* = \left\lceil \log_2\left(\frac{\ell \max\{\|d^1\|_\infty, \ldots, \|d^\ell\|_\infty\}}{\epsilon}\right)\right\rceil$
12: Find $\alpha \in \mathbb{Q}^\ell$ with $\sum_{i=1}^{\ell} \alpha_i d^i = x^{\text{rint}} - \bar{x}$
13: $\bar{\rho} \leftarrow \sum_{i=1}^{\ell} \frac{\lfloor 2^{r^*}\alpha_i \rfloor}{2^{r^*}} d^i$
14: $x^* \leftarrow \bar{x} + \bar{\rho}$
15: **return** x^*

2.2. Implementation Details

We describe the implementation of the baseline algorithm, with particular attention to correctness and efficiency. We outline key choices and reference existing software packages used to ensure both.

To preserve exactness throughout the baseline algorithm, all arithmetic is performed using arbitrary-precision integer and rational types. Intermediate quantities arising in the construction

of the unimodular matrix U, the affine solution $\bar{x}$, the nullspace basis vectors, and the correction vector $\bar{\rho}$ may grow far beyond standard fixed-width integer ranges. The use of floating-point arithmetic would compromise feasibility guarantees, particularly in Phase II where small perturbations may violate nonnegativity or equality constraints. We therefore rely on the GNU MP (GMP) library (Granlund 2015) for exact arithmetic. Integers are represented using mpz_int, and rational numbers are implemented via boost::rational<mpz_int>. The GMP library is designed to deliver high performance across a wide range of operand sizes by combining fast algorithms with optimized low-level implementations. In our setting, GMP provides the exact arithmetic backbone required to guarantee correctness of all computations.

We remove redundant rows from the system $[A\, b]$ to ensure that A has full row rank before computing the Hermite normal form. Concretely, we use the FLINT C++ number theory library to perform a fraction-free LU decomposition, which identifies a maximal set of linearly independent rows via pivoting. We also use the FLINT library for the computation of the Hermite normal form of A and the associated unimodular matrix. This choice is motivated by its strong practical performance and robustness on moderately large instances. However, as we will discuss in Section 3, the resulting unimodular matrix may contain entries of large magnitude, which negatively impacts both runtime and the size of the dyadic denominators of the returned solution.

Linear programming (LP) subproblems arising in the algorithm are solved using SoPlex in rational mode whenever exactness is required. In particular, we use the exact solver to determine whether P is nonempty, and to compute a relative interior point and corresponding radius via (1). This either provides a certificate of infeasibility, or ensures that the computed interior point and radius yield a valid dyadic solution.

When identifying implicit equalities of the form $x_j = 0$, however, we solve the corresponding linear programs using Gurobi, declaring $x_j = 0$ whenever $x_j \leq 10^{-6}$, which matches the solver's default feasibility tolerance. Since this subroutine is invoked at least n times, using an exact LP solver would be computationally expensive, whereas Gurobi is a significantly more efficient heuristic for this step. We then verify that P remains nonempty after setting $x_j = 0$ for all $j \in J^=$ using SoPlex as an exact LP solver. If Gurobi fails to detect an implicit equality, *i.e.*, if some variable x_j that should be fixed to zero is not identified, then the radius ϵ of the infinity-norm ball B computed in (1) is equal to zero, indicating that at least one such equality has been missed. We treat this situation as an error condition and revert to solving the corresponding subproblems using SoPlex in exact rational mode to identify the missing equalities. In all of our computational experiments, this fallback was never triggered.

3. Efficient HNF Computation and Unimodular Matrix Reduction

In this section, we focus on the Hermite normal form (HNF) computation which is used to find a dyadic point in Phase I and to produce a nullspace basis for Phase II. The choice of method for HNF computation affects the baseline algorithm's efficiency and the quality of its final solution. We provide experimental evidence for how the HNF computation affects the baseline algorithm and introduce two methods for improving the HNF computation before using it in the baseline algorithm.

First, we define HNF and clarify its use in the baseline algorithm. A matrix $H \in \mathbb{Q}^{m\times n}$ is in *Hermite normal form* if $H = (D\ 0)$, where $D \in \mathbb{Q}^{m\times m}$ is lower triangular with $d_{ii} > 0$ and $0 \leq d_{ij} < d_{ii}$ for $j < i$. In particular, H has full row rank. For any full row rank matrix $A \in \mathbb{Q}^{m\times n}$, there exists a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that $H = AU$, and H is unique. Moreover, such a decomposition can be computed in polynomial time (Kannan and Bachem 1979). We refer the reader to Conforti et al. (2014) and Schrijver (1998) for further details.

Writing $U = (U_1\ U_2)$, we have $AU_1 = D$ and $AU_2 = 0$, so the columns of U_2 form an integral basis of the lattice $\ker_{\mathbb{Z}}(A) = \{d \in \mathbb{Z}^n : Ad = 0\}$. We compute the Hermite normal form of A, set $y = D^{-1}b$, $\bar{x} = U\begin{pmatrix} y \\ \mathbf{0} \end{pmatrix}$, and use the columns of the corresponding matrix U_2 directly in Phase II. Notice that although the matrix H is unique, the matrix U is not unique.

In practice, the algorithm for computing the HNF has a significant impact on performance and solution quality. In our implementation, we use the FLINT library to compute a Hermite normal form factorization $AU = (D\ 0)$ efficiently. While this approach is fast and robust, the associated unimodular matrix U often contains entries of extremely large magnitude. We observed entries as large as 10^{200} in our experiments with random 0–1 matrices $A \in \mathbb{Z}^{100\times 300}$. This affects runtime as these large operands are stored with arbitrary precision, as well as the dyadic denominator exponent as it directly corresponds to large values of r^* computed in Phase II.

We also tested the HNF-LLL approach of Havas et al. (1998), implemented via the CALC system of Matthews. While this method yields unimodular matrices with significantly smaller entries, we found it to be computationally prohibitive for moderate instance sizes (e.g., $m \geq 100$, $n \geq 300$). We therefore rely on FLINT for efficiency and apply a structured postprocessing procedure to reduce the magnitude of the entries of U.

Assume $\text{rank}(A) = m$, let $k = n - m$, and write $U = (U_1\ U_2)$, where $U_2 \in \mathbb{Z}^{n\times k}$ spans the kernel.

Our reduction procedure consists of two steps:

(i) kernel basis reduction, and

(ii) non-kernel column reduction using kernel projections.

First, we reduce the kernel basis U_2 using the LLL algorithm (Lenstra et al. 1982). There exists a unimodular matrix $V \in \mathbb{Z}^{k\times k}$ such that $U_2' := U_2 V$ is LLL-reduced. Since $A(U_2 V) = 0$, this preserves

the factorization $AU = (D\,0)$. Empirically, this step significantly decreases $\|U_2\|_\infty$, defined as the maximum absolute row sum of the matrix U_2, especially when k is large.

Second, we reduce the columns of U_1 using integer combinations of the reduced kernel basis. For each column u of U_1, we compute $u' = u - U_2'c$ for some $c \in \mathbb{Z}^k$. Since $AU_2' = 0$, this operation also preserves $AU = (D\,0)$. The vector c is obtained by approximating the least-squares projection of u onto the span of U_2'.

Let

$$G = {U_2'}^\top U_2' \in \mathbb{Z}^{k\times k}, \qquad Y = {U_2'}^\top U_1 \in \mathbb{Z}^{k\times m},$$

and compute

$$Q = G^{-1} Y \in \mathbb{Q}^{k\times m}.$$

We round componentwise to obtain $C = \lfloor Q \rceil \in \mathbb{Z}^{k\times m}$ and set

$$U_1' = U_1 - U_2' C.$$

The system $GQ = Y$ is solved exactly, and rounding is performed in rational arithmetic. This procedure can be interpreted as a lattice projection that shortens the non-kernel columns relative to the kernel lattice.

We denote the resulting matrix by $U' = (U_1'\, U_2')$. Although heuristic, this procedure consistently produces unimodular matrices with substantially smaller $\|U\|_\infty$, while remaining computationally tractable.

m	n	**Reduction Mode**	$\|U\|_\infty$	**Dyadic Solution Support**	**Dyadic Solution Max k**	**HNF and Reduction Time (s)**	**Total Time (s)**
50	150	No reduction	3.4×10^{34}	150	110.8	0.23	0.68
		U_2 reduced	5.7×10^{32}	150	15.5	0.27	0.46
		U_1, U_2 reduced	2.4×10^{2}	150	15.5	0.32	0.53
100	300	No reduction	5.4×10^{90}	300	305.8	2.0	5.9
		U_2 reduced	9.6×10^{88}	300	19.5	4.8	5.8
		U_1, U_2 reduced	1.7×10^{3}	300	19.5	5.4	6.4
200	600	No reduction	1.4×10^{238}	600	793.7	33	70
		U_2 reduced	1.3×10^{236}	600	24	65	72
		U_1, U_2 reduced	2.1×10^{4}	600	24	73	80

Table 1 Impact of unimodular matrix reduction strategies on solution structure and computational performance. Results are averaged over 10 Bernoulli random 0–1 instances with density 0.25 for each problem size. All runtimes are reported to two significant digits.

Table 1 summarizes the impact of these reduction strategies on randomized Bernoulli 0–1 matrices A with density 0.25. We also evaluate the performance of the baseline algorithm using these strategies on the nonnegative dyadic feasibilty problem $Ax = \mathbf{1}$, $x \geq 0$. We present the support size

and dyadic exponent $k = \log_2(d)$ for the largest denominator d of the dyadic solutions obtained. The experiments compare three configurations:

(i) no postprocessing of U,

(ii) LLL reduction applied only to U_2, and

(iii) LLL reduction of U_2 followed by projection-based reduction of U_1.

All results are averaged over 10 instances for each problem size.

Across all tested dimensions, the reduction procedures decrease $\|U\|_\infty$ by many orders of magnitude. For example, when $m = 100$ and $n = 300$, $\|U\|_\infty$ decreases from 5.4×10^{90} to 1.7×10^3. Correspondingly, the maximum dyadic denominator decreases from approximately 2^{300} to below 2^{20} on average.

Interestingly, reducing only U_2 is already sufficient to control the size of the denominator. However, reducing U_1 further decreases the magnitude of the intermediate affine solution $\bar{x} = U \begin{pmatrix} y \\ \mathbf{0} \end{pmatrix}$, which can otherwise become extremely large even when the final dyadic solution is well-scaled. Moreover, the lattice projection step often introduces many zero rows in U_1, which induces greater sparsity in $\bar{x}$. However, Phase II of the baseline algorithm steers the solution toward the relative interior of P, resulting in a nonnegative dyadic solution x^* with full support.

Overall, the proposed postprocessing pipeline achieves substantial numerical improvements with a moderate overhead in HNF computation time. For the largest tested instances ($m = 200, n = 600$), the total runtime increases marginally while $\|U\|_\infty$ is reduced by over 230 orders of magnitude and max k is reduced by approximately 97%. While the time spent in the HNF and reduction wrapper increases, the increase in total runtime is significantly smaller. This is because downstream Phase II computations, including the evaluation of $\bar{x}$, r^*, α, and $\bar{\rho}$, become substantially cheaper when the unimodular matrix is reduced. In particular, the reduced matrix yields much smaller intermediate rational values, limiting growth in operand size in the GMP arithmetic and thereby reducing computational time and cost.

4. Improving the Baseline Algorithm for Nonnegative Dyadic Feasibility

The computational results of Section 3 reveal two practical shortcomings of the baseline algorithm. First, the dyadic solutions it returns are often fully dense. Second, the resulting dyadic denominators may still be larger than desirable in practice.

In this section, we develop modifications that improve the practical quality of the dyadic solutions produced by the baseline algorithm. For each refinement, we report computational experiments on random 0–1 instances, solving the nonnegative dyadic feasibility problem $Ax = \mathbf{1}$, $x \geq 0$. We measure the quality of a dyadic solution primarily by the support size and the dyadic denominator exponent, k.

4.1. Column Generation

We adopt a column generation framework with the goal of producing dyadic solutions with smaller support. We develop a method for creating an initial set of columns and a pricing strategy for generating columns based on dyadic infeasibility certificates.

Column generation is a general technique to solve linear programs with a large number of variables. In this framework, a restricted master problem is solved over a subset of the columns, and a pricing step identifies new columns that can improve feasibility or optimality. If no such columns exist, the current solution is valid for the full problem; otherwise, the column set is expanded and the process repeats.

In the baseline method, Phase I computes a dyadic affine point by finding the HNF of the full matrix $A \in \mathbb{Z}^{m\times n}$ and Phase II moves to the interior of the polyhedron P in the full n-dimensional space. In contrast to this, our column generation approach considers a restricted column set $C \subset [n]$, and solves the dyadic feasibility problem over the subsystem defined by $A_C \in \mathbb{Z}^{m\times |C|}$, where A_C is the submatrix of A indexed by columns C. The procedure adds a column to C only when a dyadic solution is not found using the columns in C.

To initialize the set of columns C, we first solve the following linear program

$$\min\ \mathbf{1}^\top x \quad \text{s.t.} \quad Ax = b,\ x \geq 0 \tag{2}$$

and let $B \subseteq [n]$ be the basis of the associated basic solution. We set $C := B$. Given an HNF factorization $A_C U_C = H_C = (D_C\ 0)$, we solve $y_C = D_C^{-1} b$. If y_C is dyadic, we recover $\bar{x}_C = U_C \begin{pmatrix} y_C \\ \mathbf{0} \end{pmatrix}$ from the associated unimodular transformation and proceed as in the baseline method: if $\bar{x}_C \geq 0$, we terminate; otherwise, we pass $\bar{x}_C$ to Phase II to obtain a nonnegative dyadic solution x_C^*, which we lift to $\mathbb{R}^n$ by setting $x_i^* = 0$ for $i \notin C$.

If y_C is not dyadic, we enlarge C by adding columns according to the following pricing rule. For a non-dyadic pivot component y_{C_i}, there exists a vector $u \in \mathbb{Q}^m$ satisfying

$$A_C^\top u \in \mathbb{Z}^{|C|}, \qquad b^\top u \notin \mathbb{D},$$

where $\mathbb{D}$ denotes the set of dyadic rationals, which serves as a certificate for dyadic infeasibility. We may pick $u = D_C^{-\top} e_i$ as such a certificate (Abdi et al. 2024, Remark 2.8). Any extension $C' \supset C$ that admits a dyadic solution must therefore include a column a_j with $j \notin C$ that breaks at least one certificate, *i.e.*, a_j that satisfies $a_j^\top u \notin \mathbb{Z}$. In our implementation, each candidate $j \notin C$ is ranked lexicographically by the following criteria:

(i) the number of certificates it breaks,

(ii) whether it breaks the least-broken certificate,

(iii) $\|a_j\|_1$, and

(iv) column index for determinism.

We refer the reader to Appendix A for a condensed pseudocode (Algorithm 7) of the column generation procedure.

We evaluate our column generation procedure on Bernoulli random 0–1 instances, benchmarking it against the Phase I procedure in the baseline algorithm. In both settings, Phase II is fixed to that of the baseline algorithm. Table 2 reports the support size, maximum dyadic exponent k, and total runtime for solutions returned by the algorithms averaged over 10 instances of matrices of varying sizes.

	Baseline			**Column Generation**		
m	**Support**	**Max k**	**Total Time (s)**	**Support**	**Max k**	**Total Time (s)**
50	150	15.5	0.53	51.1	60.2	0.14
100	300	19.5	6.4	101.2	148.3	1.0
200	600	24	70	201.2	368.3	11
300	900	27	500	301.1	655.1	58
400	1200	30	1500	401.1	943.4	160

Table 2 Comparison of the column generation procedure with the baseline algorithm, on Beroulli random 0–1 instances with $n = 3m$. Results are averaged over 10 Bernoulli random 0–1 instances with density 0.25 for each problem size. All runtimes are reported to two significant digits.

We observe that column generation (CG) substantially reduces support size on average, relative to the baseline algorithm. In particular, CG returns solutions with support $\approx m+1$, whereas the baseline algorithm has full support.

A consistent feature of these runs is that the initial restricted set C obtained from LP initialization in CG never produces a dyadic affine solution immediately, so at least one column-generation step is always required. However, the algorithm computes the HNF of significantly smaller matrices A_C, which is drastically faster than computing the HNF of the full sized matrix A. As a result, the CG method returns a dyadic solution with a significantly smaller runtime.

The main drawback of the column generation approach is denominator growth. For example, at $m = 100$, the support size decreases from 300 under the baseline algorithm to 101.2 under CG but the maximum denominator increases from $2^{19.5}$ to $2^{148.3}$, on average. These results therefore indicate a support–denominator trade-off in these algorithms.

4.2. Iterative r-search

The baseline algorithm defines the value $r^* := \left\lceil \log_2 \left(\frac{\ell \max\{\|d^1\|_\infty, \ldots, \|d^\ell\|_\infty\}}{\epsilon} \right) \right\rceil$, which guarantees that the rounded correction remains inside the certified ℓ_∞-ball around the relative interior point and therefore preserves feasibility. In practice, however, this bound can be conservative. We therefore propose replacing the single prescribed choice by an iterative *r-search* over $r = 0, 1, \ldots, r^*$ and accepting the first value that yields a feasible dyadic point.

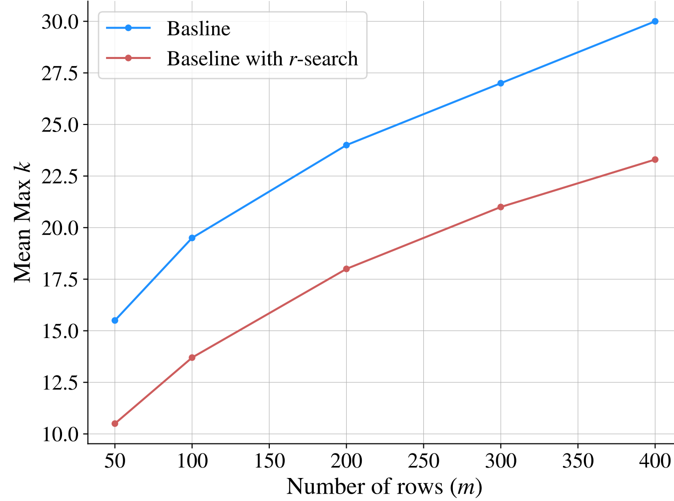

Figure 1 **Effect of the r-search framework compared to the baseline one-shot r^* approach. Results are averaged over 10 Bernoulli random 0–1 instances with density 0.25 for each problem size with $n = 3m$.**

Figure 1 shows that the choice of r^* is indeed conservative in practice, as the maximum denominator of the dyadic solution decreases consistently across all problem sizes when iterating over r. Moreover, this reduction is achieved at only a marginal increase in runtime, which is further reduced by performing a binary search over $[0, r^*]$. However, by the nature of the baseline algorithm, the solutions still have full support.

4.3. Column Generation with a Bounded r-search

We develop a method that integrates column generation with the iterative r-search framework. The aim is to better control the trade-off between sparsity and dyadic denominator size.

Rather than terminating after the first dyadic solution is found, we continue to expand the column set in order to explicitly track how the support and denominator evolve. In particular, we compute the initial dyadic affine solution $\bar{x}_C$, the associated basic solution x^*_C, and the lifted solution x^* using column generation with the certificate pricing rule and the r-search procedure. We then iteratively expand C by one column and recompute $\bar{x}_C$, x^*_C, and x^* at each step. After the first dyadic affine point $\bar{x}_C$ is identified, the certificate pricing rule is no longer applicable, as

no dyadic infeasibility certificate is available. In this regime, we adopt a deterministic fallback rule that augments C with K additional columns having the smallest $\|a_j\|_1$ values, excluding all-zero columns. This choice is motivated by the observation that restricting A_C to columns with small ℓ_1-norm tends to simplify the Hermite normal form computation, leading to smaller entries in the unimodular transformation matrix U and, consequently, more favorable nullspace basis vectors.

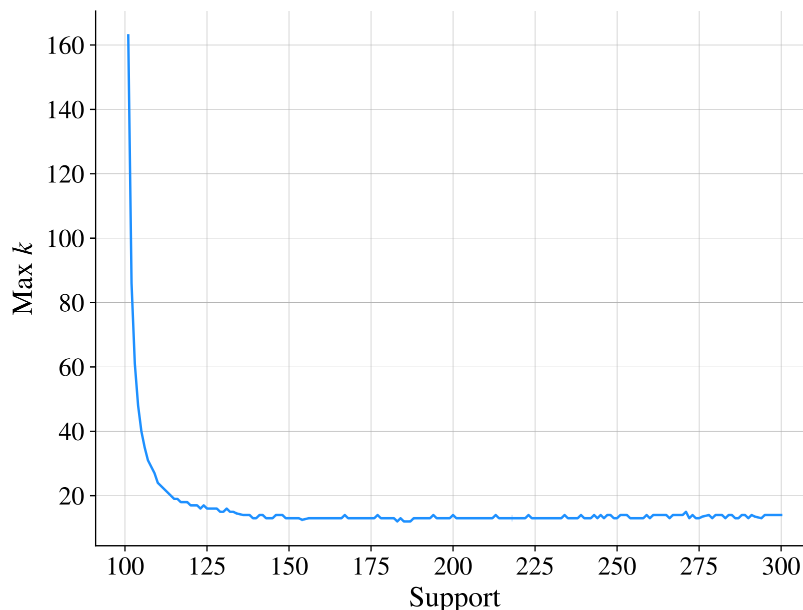

Figure 2 **Support size versus maximum dyadic exponent k of dyadic solutions found using column generation with r-search on one instance of Bernoulli random 100×300, 0–1 matrix with density 0.25. Columns are generated using the certificate pricing rule until a first dyadic $\bar{x}$ is produced, after which columns are generated using the fallback pricing rule, with $K = 1$.**

To investigate the support–denominator trade-off, we solve the nonnegative dyadic feasibility problem $Ax = \mathbf{1},\ x \geq 0$, where $A \in \{0,1\}^{100\times 300}$ is a Bernoulli random matrix with density 0.25. The results of the experiment are given in Figure 2. There is a clear trade-off between support size and the maximum dyadic exponent k. While smaller denominators generally require larger supports, the trade-off is highly favorable: moderate increases in support can yield substantial reductions in k. For the 100×300 instance, increasing the support from 101 to 135 reduces the maximum exponent k from 163 to 14. Notably, most of this improvement occurs within a relatively narrow range of support sizes, indicating diminishing returns beyond moderate support growth. This suggests that significantly smaller denominators can often be achieved at a modest sparsity cost. The smallest maximum exponent observed in this experiment for this instance is $k = 12$. To further probe the minimum achievable value, we also considered the integer feasibility model $\max\{0 : Ax = 2^k\mathbf{1},\ x \in \mathbb{Z}^n_+\}$. Using Gurobi, we verified infeasibility for $k = 0, 1, 2$. For all larger

values up to $k = 12$, the solver reached the 3-hour time limit without resolving feasibility. Thus, the minimum feasible exponent remains unresolved via the direct integer programming approach, whereas our method certifies feasibility at $k = 12$.

Motivated by these observations, we propose a *bounded r-search* over $r = 0, 1, \ldots, \min\{r^*, R\}$, where R is a prescribed upper bound. If $R \geq r^*$, termination at r^* is guaranteed, which suffices for feasibility (Abdi et al. 2024, Lemma 2.5). Once an initial dyadic solution $\bar{x}$ has been obtained, if no feasible nonnegative dyadic point is found within the current r-search bound R during Phase II, we generate additional columns using the fallback pricing rule, recompute $\bar{x}_C$, and repeat the search until no more columns can be generated. The goal is to compute a solution whose entries have denominators at most 2^R while limiting the number of generated columns.

We emphasize that the bottleneck in computational costs of the column generation procedure lies in computing the Hermite normal form of A_C. Since each iteration requires recomputing the HNF for progressively larger submatrices, it is advantageous to use a larger K for larger instances to reduce the total number of HNF computations. Thus, the parameters R and K provide a mechanism to balance denominator size and computational effort.

To study the effect of the bound R, we evaluate the column generation framework with bounded r-search under progressively tighter bounds R. Starting from the conservative estimate r^*, after obtaining a solution with maximum dyadic exponent k, we update the bound to $R = k - 1$ and re-run the algorithm. This yields a sequence of increasingly restrictive problems aimed at reducing the denominator size. In these experiments, we set $K = 1$ for $m = 50, 100$, and $K = 10, 25, 50$ for $m = 200, 300, 400$, respectively. Larger values of K can reduce runtime by decreasing the number of CG iterations, although at the cost of a coarser exploration of the support–denominator trade-off.

	Baseline with r-search			**CG with bounded r-search**		
m	**Support**	**Max k**	**Total Time (s)**	**Support**	**Max k**	**Total Time (s)**
50	149.5	10.5	0.63	87.1	9.2	4.7
100	300	13.7	7.4	181.5	12.5	73
200	600	18.0	85	307.2	17.0	89
300	900	21.0	500	483.6	19.9	420
400	1200	23.3	1500	596.1	23.5	780

Table 3 Performance of the column generation with bounded-r search algorithm benchmarked against the baseline Phase I with r-search, on Beroulli random 0–1 instances with $n = 3m$ and density 0.25. Results are averaged over 10 instances for each problem size. All runtimes are reported to two significant digits.

Table 3 reports the support size, maximum dyadic exponent k, and total runtime for the solutions with the smallest observed k (corresponding to the tightest bound R reached within a time limit of

1800 seconds). We compare these solutions against those returned by the baseline algorithm with r-search. Results are averaged over 10 instances for each problem size. The table demonstrates that combining bounded r-search with column generation can yield solutions with substantially smaller support size and denominators than the baseline approach that uses the full column set for $m = 50, 100, 200, 300$. For $m = 400$, however, the method reaches the time limit before attaining the smaller maximum denominators achieved by r-search with the baseline Phase I. Additionally, despite requiring multiple HNF computations, the column generation approach is faster than the baseline method for $m = 200, 300, 400$. This is because computing the HNF of several smaller submatrices A_C is significantly less expensive than computing the HNF of the complete, large matrix A.

We emphasize that this proposed approach does not guarantee the simultaneous attainment of small support and small denominators. Achieving small denominator exponents requires sufficiently tight bounds R. Simultaneously, overly restrictive choices may lead to excessive column generation or failure to retrieve a solution given such bounds. The reported results are not obtained from a single execution, but from repeated runs with progressively tightened bounds R, selecting the best solution found within the time limit. In practice, comparable performance in one run would require prior knowledge of a suitably tight R, which is generally unavailable. Instead, our experiments demonstrate that, for any prescribed R, the framework can effectively exploit the support–denominator trade-off by systematically augmenting the column set to search for solutions with denominators bounded by 2^R.

5. Experiments

In this section, we report computational experiments comparing the dyadic feasibility algorithms with Gurobi and SoPlex on several classes of instances. All experiments were implemented in C++ and run on an Apple M3 machine with 8 GB of RAM and an 8-core CPU. We used Gurobi 13.0.0 and SoPlex 8.0.0 with default parameter settings, with SoPlex run in rational mode. We consider randomized 0–1 feasibility instances of the form $Ax = \mathbf{1}$, $x \geq 0$, integer-infeasible instances from the MIPLIB Collection Set, and finally combinatorial instances motivated by the Berge–Fulkerson conjecture. Our code and instances are publicly available online at `https://github.com/vmpatil/dyadic-feasibility-solver`.

5.1. Random Feasibility Instances

We begin with Bernoulli randomized 0–1 instances with density 0.25, as introduced in Section 3, where we solve $Ax = \mathbf{1}$, $x \geq 0$. These experiments provide a controlled comparison between the dyadic approach, Gurobi, and SoPlex. For the dyadic algorithms, we report the metrics of the solutions which minimize the product "support $\times$ max k" for each instance. Under this selection

metric, we found that the column generation with bounded r-search scheme achieved the strongest performance. This is consistent with the findings of Section 4, where column generation reduces support while bounded-r search improves denominator quality. In these experiments, R was chosen modestly for each instance and matrix size in order to keep the support size small while bounding the denominator size. We set $K = 1$ for $m = 50, 100$, and $K = 10, 25, 50$ for $m = 200, 300, 400$, respectively.

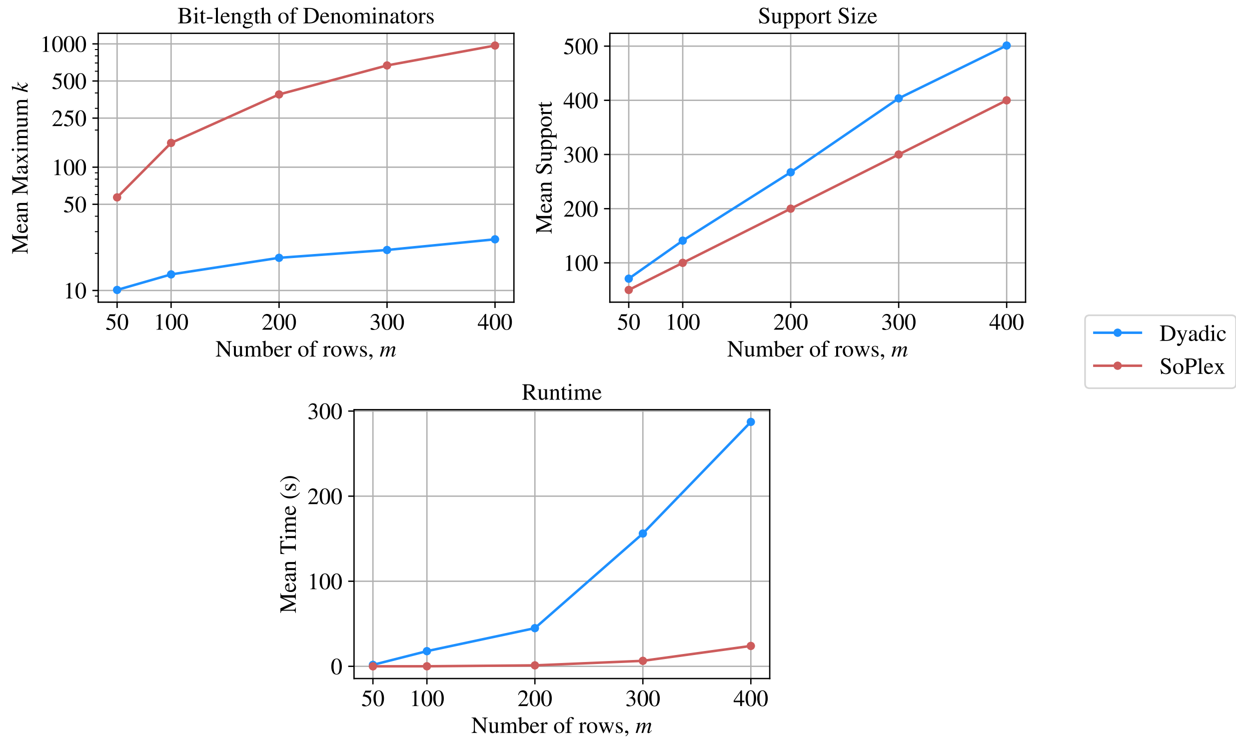

Figure 3 **Results of the dyadic algorithm compared against exact rational solutions obtained by SoPlex in rational mode. The point is an average over 10 Bernoulli random** 0–1 **instances with density** 0.25**. The** y**-axis of the Bit-length of Denominators plot is shown in** $\log$ **scale.**

We present a summary of our results in Figure 3 where each point is an average over 10 Bernoulli random 0–1 instances. We refer the reader to Appendix B for a comprehensive summary of our results. The bit length curve for the dyadic algorithm reports the maximum dyadic exponent k for dyadic solutions, while the curve for SoPlex reports the bit length $k := \lceil \log_2(d+1) \rceil$ of the largest denominator d for non-dyadic solutions. We use the alternate calculation of k for SoPlex as a way to compare its rational solutions with dyadic values. We note that Gurobi returns solutions with support m in under one second for all instances, but these solutions do not satisfy the equations $Ax = b$ exactly: on average, the ℓ_∞-norm of the residuals are on the order of 10^{-15}.

On these instances, both Gurobi and SoPlex return solutions with support equal to m, whereas the dyadic solutions are denser but remain well below n. At the same time, the dyadic method keeps the maximum denominator exponent between 10 and 26 on average, while SoPlex produces rational solutions whose denominators require substantially larger bit lengths. The main limitation of the dyadic algorithm is its runtime. This reflects the additional structure imposed by exact dyadic feasibility. Nevertheless, as discussed in Section 4.3, the parameters R and K provide effective control of the runtime–denominator trade-off. Thus, the dyadic approach trades additional support and runtime for exact feasibility, small bit lengths, and finite binary representation of its solutions.

5.2. MIPLIB Infeasible Instances

We next consider integer-infeasible instances derived from the MIPLIB 2017 Collection Set (Gleixner et al. 2021). As discussed in the introduction, dyadic feasible points can be viewed as a structured alternative to unrestricted linear relaxations. These benchmark instances therefore test whether the proposed methods remain effective beyond synthetic random matrices.

Guided by the scaling behavior observed in Section 5.1, we restrict attention to instances with fewer than 400 constraints. For each instance, we require that the constraint matrix and right-hand side admit an exact rational interpretation; we retain instances with integer or meaningful rational data and exclude those whose coefficients appear to be approximate floating-point values.

Each instance is converted to standard form $Ax = b$, $x \geq 0$ by introducing slack or surplus variables, splitting free variables, shifting variables with finite bounds, expanding ranged constraints, and scaling rows to obtain an integral system. Since we focus only on feasibility, any objective function is discarded.

In addition, we include the `markshare` instances, which are listed as integer feasible in MIPLIB 2017. These models originate from the small, hard 0–1 problems of Cornuéjols and Dawande (1999). In the MIPLIB formulations, slack variables are introduced and the objective minimizes their sum. To obtain infeasible instances, we fix this slack-sum objective to zero, thereby forcing all slack variables out.

This preprocessing yields 18 instances. In all experiments, we use column generation combined with bounded r-search. We set $K = 10$ when $n - m > 100$ and $K = 1$ otherwise. To find solutions with small denominators, we iteratively reduce the threshold R in the bounded-r search. Table 4 reports results for the dyadic method, SoPlex in rational mode, and Gurobi on the resulting feasibility problems $\max\{0 : Ax = b,\ x \geq 0\}$.

Several features stand out in Table 4. First, many of these structured problems admit dyadic solutions with very small denominator exponents. In particular, 6 of the 18 instances yield half- or quarter-integral solutions ($k \in \{1, 2\}$), and in most remaining cases the exponent stays below

			Gurobi			**SoPlex**			**Dyadic**		
Instance	m	n	**Support**	ℓ_∞ **of residuals**	**Total Time (s)**	**Support**	**Max Bit Length**	**Total Time (s)**	**Support**	**Max** k	**Total Time (s)**
stein9inf	23	32	19	0	0.00032	22	3	0.00032	21	1	0.017
flugplinf	30	42	30	5.8×10^{-11}	0.00035	30	20	0.00059	36	6	0.13
enlight4	32	48	17	0	0.00031	17	2	0.00022	17	1	0.0051
markshare_4_0	35	64	34	1.1×10^{-13}	0.00040	34	26	0.00064	56	3	0.94
markshare_5_0	46	85	45	2.3×10^{-13}	0.00042	45	29	0.00081	77	4	1.0
stein15inf	52	67	48	2.2×10^{-16}	0.00039	37	2	0.00062	67	1	0.32
markshare1	63	118	56	0	0.00036	56	39	0.0011	98	5	34
g503inf	64	102	51	4.6×10^{-13}	0.00043	51	27	0.0011	89	12	1.4
markshare2	75	141	67	2.3×10^{-13}	0.00045	67	46	0.0014	100	4	13
ponderthis0517-inf	78	1027	75	2.0×10^{-15}	0.0023	77	23	0.0059	90	9	2.6
p2m2p1m1p0n100	102	202	102	0	0.00028	102	12	0.0010	126	4	1.1
enlight9	162	243	82	0	0.00028	82	2	0.00093	82	1	0.036
fhnw-sq3	216	2503	200	4.6×10^{-13}	0.0082	189	53	0.070	763	12	570
enlight11	242	363	122	0	0.00066	122	2	0.0013	122	1	0.077
neos859080	244	361	163	3.0×10^{-17}	0.00065	124	5	0.0020	125	2	0.19
misc05inf	319	432	226	1.1×10^{-13}	0.0018	227	13	0.0060	200	4	0.74
mod008inf	326	645	326	0	0.00065	326	60	0.0050	364	9	4.1
stein45inf	377	422	320	2.1×10^{-14}	0.0013	312	2	0.0032	372	3	11

Table 4 Comparison of the dyadic algorithm, SoPlex (exact rational mode), and Gurobi on integer infeasible problems derived from MIPLIB. All runtimes are reported to two significant digits.

10. This phenomenon is especially prominent in the `enlight` and `stein` instances, which originate from combinatorial structures. While such fractionality is not unexpected in these settings, it is notable that these models, despite being integer infeasible, consistently admit dyadic solutions with small binary representation. For the `enlight` instances, both Gurobi and SoPlex also recover half-integral solutions, indicating that this structure is intrinsic to the underlying problem rather than an artifact of the dyadic algorithm.

Second, the sparsity patterns differ substantially from those observed in the randomized experiments. In many instances, the dyadic solutions have support strictly smaller than m, reflecting the presence of implicit equalities that reduce the effective dimension of the feasible region. Even when the support exceeds m, the solutions remain relatively sparse. Notably, for instances such as `misc05inf` and `stein9inf`, the dyadic algorithm produces solutions with smaller support than SoPlex. This can be traced to the initialization step, where solving (2) yields a sparse basis, and only a small number of additional columns are required to achieve dyadic feasibility.

In terms of runtime, the dyadic method is slower than the linear feasibility approaches, but the observed performance remains practical. A majority of instances are solved within a few seconds, and even larger models such as `ponderthis0517-inf`, which has over 1000 variables, are solved efficiently. This indicates that the column generation based algorithm is well suited to such problems. Gurobi also performs extremely well on these instances. It finds exact solutions for 7 instances and, in the remaining cases, produces solutions with very small residuals. However, unless the problem is highly structured, neither Gurobi nor SoPlex return solutions with exact finite binary representations.

A small number of instances also illustrate the limits of the dyadic algorithms. The instance `fhnw-sq3` requires substantially more time and produces a solution with both large support and a relatively large denominator exponent. This appears to be driven by the large number of variables, which makes identifying columns leading to small-denominator solutions more difficult. When enforcing a strict threshold R, the algorithm spends considerable time in column generation, leading to growth in both runtime and support. When this restriction is relaxed by setting $R = r^*$, the algorithm produces a solution with support 201 and denominator 46 in 36.52 seconds, indicating that much of the computational effort is devoted to reducing fractionality.

A related phenomenon occurs in `g503inf`, where the algorithm is unable to find a solution with exponent $k \leq 11$, even after all columns are included. Solving the equivalent integer formulation $\max\{0 : Ax = 2^k b,\ x \in \mathbb{Z}^n_+\}$ shows infeasibility for $k = 0, 1, \ldots, 6$, while a feasible solution is found at $k = 7$ with support 71. This example illustrates that, in favorable cases, minimum-denominator solutions can indeed be recovered via integer programming. However, this behavior is not typical. As discussed in Section 4.3, the same formulation becomes computationally intractable even for moderate instance sizes. For the randomized instance with $m = 100$, the solver fails to determine feasibility for any $k \geq 3$ within a 3-hour time limit.

Overall, these results show that structured infeasible instances often admit sparse dyadic solutions with small denominators, and that such certificates can be found efficiently in practice.

5.3. Combinatorial Instances from Perfect-Matching Coverings

We consider graph-theoretic instances inspired by the Berge–Fulkerson conjecture, which posits that certain cubic bridgeless graphs admit half-integral edge coverings by perfect matchings. Let $G = (V, E)$ be such a graph, and let $\mathcal{M}$ denote its set of perfect matchings. We construct the incidence matrix $M \in \{0,1\}^{|E|\times|\mathcal{M}|}$, where each column corresponds to a perfect matching. The system $Mx = \mathbf{1}$ requires each edge to be covered exactly once by the perfect matchings in the graph. The conjecture asserts that a half-integral solution exists for cubic bridgeless graphs, *i.e.*, that there exists a half-integral solution to the problem $Mx = \mathbf{1}, x \geq 0$ with support 6. The Berge–Fulkerson conjecture has been verified for numerous families of snarks (see, Hao et al. 2009, Karam and Campos 2014, Zhu et al. 2014, Manuel and Shanthi 2015, Hao et al. 2018, Liu et al. 2021a,c,b). Moreover, Máčajová and Mazzuoccolo (2020) proved that a possible minimum counterexample must be cyclically 5-edge-connected, and all snarks with order at most 36 have been verified to satisfy the conjecture (see, Brinkmann et al. 2013, Corollary 8.5).

In our experiments, we examine the complete set of 31 snark graphs on 44 vertices, 66 edges, and oddness 4, originally identified by Goedgebeur et al. (2019) as the smallest nontrivial snarks, *i.e.*, cyclically 4-edge-connected graphs of girth at least 5 having oddness at least 4. By the results

of Máčajová and Mazzuoccolo (2020), these 31 snarks are known to satisfy the conjecture. While they are not candidates for counterexamples, their large girth and intricate structure make them interesting test instances for dyadic feasibility.

For each graph, we construct the perfect matching incidence matrix by enumerating all perfect matchings using binary decision diagrams (Akers 1978). Our method differs from previously described methods that enumerate perfect matchings using dynamic programming (Godsil 2017). Most existing methods use a recursive function $\mu(G)$ that returns the number of perfect matchings in a graph $G=(V,E)$ as $\mu(G)=\mu(G-e)+\mu(G-u-v)$ for any edge $e=\{u,v\}\in E$ and compute $\mu(G)$ by solving the subcases in a bottom-up manner. We use a top-down approach that shares some similarities with an enumeration method specifically devised for n-cubes (Östergård and Pettersson 2013).

We enumerate the set of perfect matchings by compiling a binary decision diagram, which is a directed acyclic state-transition graph that represents each perfect matching as a path from a root node to a terminal node. The decision diagram relies on an ordering of the edges $E=(e_0,...,E_{|E|})$ in the input graph. The *top-down* compilation of the decision diagram begins with a root node, and at each node considers two transitions: one that represents adding the next edge in the ordering to the perfect matching, and one that represents not adding the next edge. Each node has an associated state that tracks the set of unsaturated vertices. Next, we give a formal description.

Denote the binary decision diagram as $D=(N,A)$, where N is the set of nodes and A is the set of arcs. The nodes are equivalent to a set of states S and the arcs are equivalent to a transition function $\phi: S\times\{0,1\}\to S$ for each $s\in S$ and $b\in\{0,1\}$ where 0 represents omitting the next edge and 1 represents including the next edge in the perfect matching. Let $r\in S$ and $t\in S$ be the root node's state and terminal node's state, respectively. Let $T\subseteq V$ and $i\in\{0,...,|E|\}$. Each state is represented as a tuple (T,i) where the first component of the tuple is a set of unsaturated vertices and the second component is the index of the next edge to be considered. The root state is $(V,0)$ and the terminal state is $(\emptyset,\frac{|V|}{2})$. To define the transition function, consider two cases. For each state (T,i), let $\phi((T,i),0)=(T,i+1)$. Let $e_i=(u,v)$. For each state (T,i), if $u\in T$ and $v\in T$, let $\phi((T,i),1)=(T-u-v,i+1)$. Otherwise, the transition function is null and the associated arc is not added to the graph. The decision diagram is compiled by starting with the root node and recursively generating the two transitions and associated nodes. After compiling the decision diagram, a breadth-first traversal starting at the root node produces all perfect matchings.

We evaluate our dyadic algorithms on these combinatorial instances. For each instance, we use the column generation procedure with the bounded r-search scheme, for which we keep the conservative bound $R=r^*$. Table 5 reports the solution found by the dyadic algorithm, SoPlex in rational mode, and Gurobi to the linear system $Mx=\mathbf{1},\ x\geq 0$. Since all snarks considered have 44

vertices, each instance has $m = 66$ edges; thus, we report only n, the number of perfect matchings, which varies across instances.

		Gurobi			**SoPlex**			**Dyadic**		
Graph ID	n	**Support**	ℓ_∞ **of residuals**	**Total Time (s)**	**Support**	**Max Bit Length**	**Total Time (s)**	**Support**	**Max** k	**Total Time (s)**
1	560	23	4.4×10^{-16}	0.0025	22	10	0.0090	23	10	0.098
2	568	23	1.3×10^{-15}	0.0028	23	13	0.0079	23	10	0.14
3	576	21	6.7×10^{-16}	0.0025	23	11	0.0079	24	10	0.084
4	555	20	3.3×10^{-16}	0.0025	23	11	0.0078	24	12	0.082
5	540	18	1.1×10^{-15}	0.0026	23	10	0.0075	21	7	1.5
6	541	23	6.7×10^{-16}	0.0023	23	14	0.0078	22	9	0.07
7	553	12	0	0.0022	23	13	0.0078	24	7	0.20
8	578	22	4.4×10^{-16}	0.0026	23	9	0.0079	14	3	0.068
9	522	23	6.7×10^{-16}	0.0023	23	9	0.0075	31	8	0.88
10	576	22	6.7×10^{-16}	0.0027	19	5	0.0086	12	2	0.038
11	576	22	1.2×10^{-15}	0.0028	22	10	0.0086	24	13	0.083
12	558	23	4.4×10^{-16}	0.0025	23	9	0.0080	30	7	0.11
13	570	23	4.4×10^{-16}	0.0029	23	13	0.012	24	12	0.081
14	578	17	8.9×10^{-16}	0.0028	23	13	0.0086	39	8	0.15
15	591	23	3.3×10^{-16}	0.0029	22	10	0.0088	32	7	0.44
16	564	20	1.7×10^{-15}	0.0023	23	12	0.0086	23	8	0.074
17	584	17	1.3×10^{-15}	0.0024	23	10	0.0079	29	9	0.35
18	588	19	3.3×10^{-16}	0.0027	23	8	0.0085	19	4	0.10
19	576	23	4.4×10^{-16}	0.0026	23	13	0.0083	33	8	0.14
20	566	22	8.9×10^{-16}	0.0030	22	10	0.0086	6	1	0.066
21	564	19	5.6×10^{-16}	0.0025	23	12	0.0084	25	9	0.13
22	556	23	6.7×10^{-16}	0.0024	23	10	0.0086	24	6	0.28
23	552	20	1.3×10^{-15}	0.0022	23	11	0.0077	29	6	0.087
24	552	15	2.2×10^{-16}	0.0023	23	9	0.0077	34	10	0.10
25	576	18	1.4×10^{-15}	0.0022	23	12	0.0082	24	10	0.082
26	522	23	3.3×10^{-16}	0.0023	23	12	0.0074	17	4	0.57
27	576	23	5.6×10^{-16}	0.0023	22	10	0.0083	24	10	0.093
28	501	23	4.4×10^{-16}	0.0023	23	12	0.0072	24	13	0.10
29	521	23	6.7×10^{-16}	0.0024	23	11	0.0072	23	12	0.11
30	520	22	4.4×10^{-16}	0.0024	23	11	0.0072	39	9	0.13
31	522	22	4.4×10^{-16}	0.0025	23	10	0.0072	22	5	1.6

Table 5 Comparison of the dyadic algorithm, SoPlex (exact rational mode), and Gurobi on perfect-matching coverings of snark graphs. All instances have $m = 66$ edges; n denotes the number of perfect matchings. All runtimes are reported to two significant digits.

The results show that, despite the large number of variables, all exceeding 500, the proposed dyadic approach solves all instances efficiently. Across all 31 instances, feasible dyadic solutions are obtained within seconds, with most solved in well under one second. In particular, computing the Hermite normal form on the row-reduced matrix M remains tractable for every graph instance considered, and the maximum observed dyadic exponent stays below 10 across all instances.

We also note that the support of all reported solutions is strictly smaller than $m = 66$. This is explained by structural properties of the edge–perfect-matching incidence matrix M, which is not

full row rank for these graphs. Linear dependencies among the rows naturally lead to solutions supported on lower-dimensional faces, and the dyadic solutions exploit this redundancy to achieve sparse representations.

From a combinatorial perspective, the experiments recover coverings consistent with the Berge–Fulkerson conjecture on the tested snarks. We emphasize, however, that these instances were already known to satisfy the conjecture; the purpose of this experiment is not to advance the conjecture itself, but to demonstrate that the dyadic framework can produce exact certificates efficiently in this setting.

Finally, SoPlex and Gurobi do not typically return solutions with the desired dyadic structure when given the linear feasibility formulation of the problem. While SoPlex produces rational solutions with moderate bit length, it does not enforce the combinatorial structure targeted by our approach. The proposed method, by contrast, directly seeks dyadic solutions and therefore yields certificates aligned with the underlying combinatorial interpretation.

6. Conclusion

We presented the first computational implementation of the dyadic feasibility algorithm of Abdi et al. (2024), providing a practical method for computing exact solutions with finite binary representations to linear systems of equations. Our implementation incorporates several refinements, including a unimodular matrix reduction, a column generation procedure, and a bounded r-search, which substantially improve upon the baseline algorithm by producing solutions with smaller support and reduced denominator size.

Our computational results show that the dyadic approach provides a meaningful alternative to both floating-point and exact-rational solvers. While classical LP solvers are faster and often produce sparse solutions, they either rely on floating-point approximations or return exact rational solutions with large bit lengths. In contrast, our method produces exact solutions with finite binary representations and small bit lengths.

Several directions for future work remain. From a computational perspective, further improvements in the efficiency of Hermite normal form computations and column generation could significantly enhance scalability, as these currently constitute the primary computational bottleneck. Avoiding explicit Hermite normal form computations altogether, for example, via pivoting-based methods that preserve dyadicness, may lead to substantial performance gains. Extending the framework to handle inequality constraints directly and to solve dyadic optimization problems with objective functions are also natural next steps. Finally, from a combinatorial standpoint, generating larger snark instances and investigating the existence of half-integral solutions to the perfect-matching covering problem are promising directions that may yield new insights into the longstanding Berge–Fulkerson conjecture.

Acknowledgements

This work is supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE1745016, DGE2140739 and by the U.S. Office of Naval Research under award number N00014-22-1-2528.

Data availability

All instances are available at the GitHub repository `https://github.com/vmpatil/dyadic-feasibility-solver`. We note that the MIPLIB instances used in this paper were available publicly. Specific references are included in the paper.

Code availability

The full code was made available for review and can be accessed at the repository `https://github.com/vmpatil/dyadic-feasibility-solver`.

References

Abdi A, Cornuéjols G, Guenin B, Tunçel L (2024) Dyadic linear programming and extensions. *Mathematical Programming* 213(1-2):473–516.

Akers (1978) Binary decision diagrams. *IEEE Transactions on computers* 100(6):509–516.

Applegate DL, Cook W, Dash S, Espinoza DG (2007) Exact solutions to linear programming problems. *Operations Research Letters* 35(6):693–699.

Brinkmann G, Goedgebeur J, Hägglund J, Markström K (2013) Generation and properties of snarks. *Journal of Combinatorial Theory, Series B* 103(4):468–488.

Conforti M, Cornuéjols G, Zambelli G (2014) *Integer Programming* (Springer Publishing Company, Incorporated), ISBN 3319110071.

Cook W, Koch T, Steffy DE, Wolter K (2013) A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation* 5(3):305–344.

Cornuéjols G, Dawande M (1999) A class of hard small 0-1 programs. *INFORMS Journal on Computing* 11(2):205–210.

Eifler L, Nicolas-Thouvenin J, Gleixner A (2025) Combining precision boosting with lp iterative refinement for exact linear optimization. *INFORMS Journal on Computing* 37(4):933–944.

Espinoza DG (2006) *On linear programming, integer programming and cutting planes* (Georgia Institute of Technology).

Gärtner B (1999) Exact arithmetic at low cost–a case study in linear programming. *Computational Geometry* 13(2):121–139.

Gleixner A, Hendel G, Gamrath G, Achterberg T, Bastubbe M, Berthold T, Christophel PM, Jarck K, Koch T, Linderoth J, Lübbecke M, Mittelmann HD, Ozyurt D, Ralphs TK, Salvagnin D, Shinano Y (2021) MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation* URL `http://dx.doi.org/10.1007/s12532-020-00194-3`.

Gleixner A, Steffy DE (2020) Linear programming using limited-precision oracles. *Mathematical Programming* 183(1):525–554.

Gleixner AM, Steffy DE, Wolter K (2016) Iterative refinement for linear programming. *INFORMS Journal on Computing* 28(3):449–464.

Godsil C (2017) *Algebraic combinatorics* (Routledge).

Goedgebeur J, Máčajová E, Škoviera M (2019) Smallest snarks with oddness 4 and cyclic connectivity 4 have order 44. *ARS Mathematica Contemporanea* 16(2):277–298.

Granlund T (2015) *GNU MP 6.0 Multiple precision arithmetic library* (Samurai Media Limited).

Hao R, Niu J, Wang X, Zhang CQ, Zhang T (2009) A note on Berge–Fulkerson coloring. *Discrete Mathematics* 309(13):4235–4240.

Hao RX, Zhang CQ, Zheng T (2018) Berge–Fulkerson coloring for c (8)-linked graphs. *Journal of Graph Theory* 88(1):46–60.

Havas G, Majewski BS, Matthews KR (1998) Extended gcd and Hermite normal form algorithms via lattice basis reduction. *Experimental Mathematics* 7(2):125–136.

Higham NJ (2002) *Accuracy and stability of numerical algorithms* (SIAM).

Kannan R, Bachem A (1979) Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM Journal on Computing* 8(4):499–507.

Karam K, Campos C (2014) Fulkerson's conjecture and Loupekine snarks. *Discrete Mathematics* 326:20–28.

Lenstra AK, Lenstra HW, Lovász L (1982) Factoring polynomials with rational coefficients. *Mathematische Annalen* 261:515–534.

Liu S, Hao RX, Zhang CQ (2021a) Berge–Fulkerson coloring for some families of superposition snarks. *European Journal of Combinatorics* 96:103344.

Liu S, Hao RX, Zhang CQ (2021b) Rotation snark, Berge-Fulkerson conjecture and Catlin's 4-flow reduction. *Applied Mathematics and Computation* 410:126441.

Liu S, Hao RX, Zhang CQ, Zhang Z (2021c) Berge–Fulkerson coloring for c (12)-linked permutation graphs. *Journal of Graph Theory* 98(4):662–675.

Máčajová E, Mazzuoccolo G (2020) Reduction of the Berge-Fulkerson conjecture to cyclically 5-edge-connected snarks. *Proceedings of the American Mathematical Society* 148(11):4643–4652.

Manuel P, Shanthi A (2015) Berge–Fulkerson conjecture on certain snarks. *Mathematics in Computer Science* 9:209–220.

Östergård PR, Pettersson VH (2013) Enumerating perfect matchings in n-cubes. *Order* 30(3):821–835.

Schrijver A (1998) *Theory of linear and integer programming* (John Wiley & Sons).

Schrijver A, et al. (2003) *Combinatorial optimization: polyhedra and efficiency*, volume 24 (Springer).

Zhu Q, Tang W, Zhang CQ (2014) Perfect matching covering, the Berge–Fulkerson conjecture, and the fan–raspaud conjecture. *Discrete Applied Mathematics* 166:282–286.

Appendix A: Supplementary Algorithms

We provide descriptions of the auxiliary algorithms used throughout the paper. These routines support the algorithms presented in this work by performing key subroutines such as feasibility checks, identification of implicit equalities, dyadic solution construction via Hermite normal form, and computation of interior points. Each algorithm is presented in pseudocode and accompanied by a brief description of its purpose and behavior.

Algorithm 2 LPFeasibility

Input: $A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^m$

Output: A variable status $\in$ {FEASIBLE, INFEASIBLE} for the linear program $\max_{x\in\mathbb{R}^n_+}\{\mathbf{0} : Ax = b\}$.

1: **if** $\{x \in \mathbb{R}^n_+ : Ax = b\} = \emptyset$ **then**
2:     **return** INFEASIBLE
3: **else**
4:     **return** FEASIBLE

Algorithm 2 (`LPFeasibility`) checks the feasibility of the polyhedron $\{x \geq \mathbf{0} : Ax = b\}$ using a standard linear programming routine. It returns a Boolean status flag indicating whether the system is feasible.

Algorithm 3 (`HNFDyadic`) attempts to find a dyadic solution to the system $Ax = b$ by computing the Hermite Normal Form (HNF) of A. If a dyadic solution exists, the algorithm returns the solution, a unimodular transformation matrix U, and a status flag. Otherwise, it returns a certificate that no dyadic solution exists. Given the computed HNF of A, $H = \begin{pmatrix} D & 0 \end{pmatrix}$, a certificate of dyadic infeasibility is a vector $u = D^{-\top} e_i$ such that $A^\top u \in \mathbb{Z}^n$ and $b^\top u$ not dyadic, where e_i is i^{th} the unit column vector and $i \in \{1, \ldots, m\}$ such that y_i not dyadic (Theorem 2.7, Abdi et al. 2024). Our implementation of Algorithm 3 performs the postprocessing procedure described in Section 3 within Line 2.

Algorithm 4 (`findDyadicAffineSolution`) computes a dyadic point in the affine hull defined by $\{Ax = b\}$, after augmenting the system with implicit equalities. It returns a dyadic solution (if one exists), a unimodular matrix U, and a status flag.

Algorithm 5 (`findImplicitEqualities`) identifies indices $j \in [n]$ such that $x_j = 0$ in every solution to $\{x \geq \mathbf{0} : Ax = b\}$. These correspond to implicit equalities in the system.

Algorithm 3 HNFDyadic

Input: $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^m$

Output: A dyadic solution to $\bar{x}$ such that $A\bar{x} = b$, a unimodular matrix $U \in \mathbb{Z}^{n\times n}$, and variable status $\in$ {DYADIC, NO-DYADIC} if a dyadic solution exists.

1: Remove redundant rows in the system $[A, b]$
2: Compute the Hermite normal form H of the matrix A and a unimdoular matrix U such that $AU = \begin{pmatrix} H & \mathbf{0} \end{pmatrix}$
3: Solve $y = H^{-1}b$
4: **if** y is dyadic **then**
5: $\quad x = U\begin{pmatrix} y \\ \mathbf{0} \end{pmatrix}$
6: $\quad$ **return** (DYADIC, x, U)
7: **else**
8: $\quad$ **return** NO-DYADIC

Algorithm 4 findDyadicAffineSolution

Input: $A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^m$, $n \geq m$

Output: Either a certificate that $P = \emptyset$, a dyadic solution $x^* \in \mathbb{R}^n$ such that $Ax^* = b$ along with a unimodular matrix $U \in \mathbb{Z}^{n\times n}$, or a variable status $\in$ {FEASIBLE, INFEASIBLE, DYADIC, NO-DYADIC}

1: status $\leftarrow$ LPFeasiblity(A, b).
2: **if** status = INFEASIBLE **then**
3: $\quad$ **return** status
4: $J^= \leftarrow$ `findImplicitEqualities`(A, b)
5: Let E be a matrix of standard row vectors e^j for $j \in J^=$.
6: Denote by A' the matrix $\begin{pmatrix} A \\ E \end{pmatrix}$ and b' the vector $\begin{pmatrix} b \\ \mathbf{0} \end{pmatrix}$
7: (status, $\bar{x}$, U) $\leftarrow$ HNFDyadic(A', b')
8: **return** status, $\bar{x}$, U

Algorithm 6 (`computeInteriorPoint`): Computes a rational point in the relative interior of the polyhedron $P = \{x \geq \mathbf{0} : Ax = b\}$. It also returns a positive radius ϵ that quantifies distance to the boundary. The LP given by (1) is solved exactly using SoPlex in rational mode.

Algorithm 7: Uses a column generation procedure for Phase I as a refinement over the baseline algorithm for the nonnegative dyadic feasibility problem.

Algorithm 5 findImplicitEqualities

Input: $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^m$

Output: A set of indices J where $x_j = 0$ are implicit equalities in P for all $j \in J$.

1: Initialize $J = \emptyset$
2: **for** $i \in [n]$ **do**
3: $\quad x_i^u = \max\{x_i : Ax = b, x \geq \mathbf{0}\}$
4: $\quad$ **if** $x_i^u = 0$ **then**
5: $\quad\quad J = J \cup \{i\}$
6: **return** J

Algorithm 6 computeInteriorPoint

Input: $A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^m, J \subseteq [n]$

Output: A rational point x^{rint} in the relative interior of the set $\{x \geq \mathbf{0} : Ax = b\}$ and a radius ϵ for ∞-norm ball centered at x^{rint}.

1: Initialize $M \in \mathbb{R}$ large.
2: Let $J^< = [n] \setminus J$ and denote by $A^<$ the column submatrix of A indexed by columns $J^<$
3: $(\bar{x}, \epsilon) = \arg\max\{\epsilon : A^< x = b, \epsilon \leq 1,\ \epsilon \leq x_j,\ j \in J^<\}$
4: Let $x^{\text{rint}} \in \mathbb{Q}^n$ be obtained from $\bar{x}$ by setting entries corresponding to J to zero.
5: **return** $x^{\text{rint}}, \epsilon$

Appendix B: Complete Experimental Results

Tables 6-10 present the complete results of the experiments discussed in Section 5.1. For the dyadic algorithms, we report the metrics of the solutions which minimize the product support $\times$ max k for each instance. The Max k column for the dyadic algorithm reports the maximum dyadic exponent for dyadic solutions, while the Max bit length column for SoPlex reports the bit length $k = \lceil \log_2(d+1) \rceil$ of the largest denominator d for non-dyadic solutions. All runtimes are reported to three significant digits.

Algorithm 7 Condensed pseudocode for column generation algorithm

Input: $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^m$

Output: a dyadic $x^* \in P$, or a certificate that $P = \emptyset$, or a certificate that $\operatorname{aff}(P)$ contains no dyadic point.

1: Solve the linear program (2).
2: **if** infeasible **then**
3:    **return** certificate that $P = \emptyset$
4: Let $C \subseteq [n]$ be the basis of the basic solution
5: $\bar{x}_C \leftarrow$ `findDyadicAffineSolution`(A_C, b)
6: **while** $\bar{x}_C$ is not dyadic **do**
7:    Attempt to price and add a column using the dyadic infeasibility certificate
8:    **if** no such column exists **then**
9:       **return** certificate that $\operatorname{aff}(P)$ contains no dyadic point
10:    $\bar{x}_C \leftarrow$ `findDyadicAffineSolution`(A_C, b)
11: **if** $\bar{x}_C \geq 0$ **then**
12:    Extend $\bar{x}_C$ to $x^* \in \mathbb{R}^n$ by zero outside C
13:    **return** x^*
14: Run Phase II with initial point $\bar{x}_C$ (Lines (9)–(14))
15: Extend x^*_C to $x^* \in \mathbb{R}^n$ by zero outside C
16: **return** x^*

	Dyadic			**SoPlex**		
Inst. ID	**Support**	**Max k**	**Time (s)**	**Support**	**Max bit length**	**Time (s)**
1	66	11	1.37	50	56	0.00626
2	74	9	2.52	50	57	0.00428
3	76	10	2.13	50	59	0.00364
4	71	10	1.58	50	58	0.00404
5	75	9	1.88	50	55	0.00379
6	72	9	1.59	50	57	0.00494
7	62	12	0.84	50	56	0.00358
8	70	10	2.08	50	58	0.00491
9	66	11	1.04	50	54	0.00443
10	76	10	2.10	50	59	0.00406

Table 6 **Results for $m = 50$, $n = 150$. Each row corresponds to one Bernoulli 0–1 instance with density 0.25.**

	Dyadic			SoPlex		
Inst. ID	**Support**	**Max k**	**Time (s)**	**Support**	**Max bit length**	**Time (s)**
1	145	13	19.2	100	160	0.0602
2	144	13	18.4	100	155	0.0586
3	137	14	14.3	100	157	0.0596
4	138	14	14.5	100	157	0.0602
5	137	14	14.3	100	156	0.0644
6	141	14	18.6	100	157	0.0571
7	143	13	17.9	100	157	0.0595
8	145	13	20.7	100	160	0.0748
9	152	13	28.3	100	157	0.0683
10	128	14	12.0	100	157	0.0710

Table 7 Results for $m = 100$, $n = 300$. Each row corresponds to one Bernoulli 0–1 instance with density 0.25.

	Dyadic			SoPlex		
Inst. ID	**Support**	**Max k**	**Time (s)**	**Support**	**Max bit length**	**Time (s)**
1	271	18	48.7	200	393	1.11
2	261	19	39.2	200	386	1.18
3	252	20	33.6	200	391	1.12
4	271	18	48.2	200	388	1.13
5	261	19	39.3	200	384	1.10
6	272	18	48.4	200	390	1.09
7	271	17	47.0	200	386	1.13
8	261	19	40.1	200	388	1.14
9	271	18	47.4	200	390	1.15
10	281	18	56.7	200	394	1.12

Table 8 Results for $m = 200$, $n = 600$. Each row corresponds to one Bernoulli 0–1 instance with density 0.25.

	Dyadic			SoPlex		
Inst. ID	**Support**	**Max k**	**Time (s)**	**Support**	**Max bit length**	**Time (s)**
1	401	21	159	300	671	6.85
2	401	22	157	300	668	6.80
3	376	23	111	300	670	6.73
4	426	21	194	300	672	6.75
5	426	19	210	300	669	6.78
6	401	21	157	300	667	5.77
7	426	22	187	300	661	6.04
8	401	21	142	300	668	5.94
9	402	20	149	300	672	5.91
10	376	23	95.3	300	666	6.83

Table 9 Results for $m = 300$, $n = 900$. Each row corresponds to one Bernoulli 0–1 instance with density 0.25.

	Dyadic			**SoPlex**		
Inst. ID	**Support**	**Max k**	**Time (s)**	**Support**	**Max bit length**	**Time (s)**
1	501	28	307	400	971	30.3
2	501	26	273	400	973	22.4
3	501	26	263	400	971	22.9
4	501	26	315	400	968	22.0
5	501	27	258	400	963	22.5
6	501	26	279	400	962	22.3
7	502	24	289	400	974	24.1
8	501	25	307	400	970	22.4
9	501	27	270	400	969	25.1
10	501	25	312	400	974	25.7

Table 10 **Results for $m = 400$, $n = 1200$. Each row corresponds to one Bernoulli 0–1 instance with density 0.25.**