# Bounded Integer Linear Constraint Solving via Lattice Search

Joe Hendrix and Benjamin F Jones

Galois Inc., Portland,OR
jhendrix@galois.com, bjones@galois.com

**Abstract**

We present a novel algorithm for solving integer linear constraint problems of the form $\boldsymbol{l} \leq \mathbf{A}\boldsymbol{x} \leq \boldsymbol{u}$. Our approach is based on techniques used to solve the closest vector problem for lattices, but adapted to use the $L^\infty$ distance metric. We have implemented this algorithm in a constraint solver called BLT.

The BLT solver was motivated by efforts to apply SMT solvers to signal processing algorithms. In particular, we describe here the problem of *reversing JPEG*, that is, finding compressed data that decompress into an image satisfying given constraints. This problem can be expressed as a bounded integer linear constraint problem, but was intractable to the SMT solvers we tried. In contrast, BLT is able to solve many of the examples in seconds, including both SAT and UNSAT problems.

## 1 Introduction

In this paper, we present a decision procedure for solving systems of integer linear constraints where each expression is subject to both upper and lower bounds. Such systems have the form:

$$\begin{array}{ccccc} l_1 & \leq & a_{11}x_1 + \cdots + a_{1n}x_n & \leq & u_1 \\ l_2 & \leq & a_{21}x_1 + \cdots + a_{2n}x_n & \leq & u_2 \\ \vdots & & \vdots & & \vdots \\ l_m & \leq & a_{m1}x_1 + \cdots + a_{mn}x_n & \leq & u_m \end{array} \tag{1}$$

where $l_i, u_i, a_{ij}$ are rational constants and the $x_i$ are unknown integer variables. As a more compact notation, we use $\boldsymbol{l} \leq \mathbf{A}\boldsymbol{x} \leq \boldsymbol{u}$ to denote systems with this form.

Our decision procedure is based on a variant of the Schnorr-Euchner algorithm [15] for computing the closest lattice element to a target point, and is implemented in a tool which we call *BLT*. The decision procedure reduces the constraint problem to the problem of checking whether there is a common point $\boldsymbol{y}$ in both the lattice $\mathcal{L}_{\mathbf{A}}$ generated by the columns of $\mathbf{A}$ and the hyperrectangle containing the points between $\boldsymbol{l}$ and $\boldsymbol{u}$.

We developed BLT while trying to apply constraint solving to signal processing algorithms. In particular, we were studying the problem of *reversing JPEG decompression*, that is finding JPEGs that decompress into images that satisfy given constraints. This could be used to encode specific values on pixels in the image, perhaps for steganographic purposes. As we will show, this problem can be expressed as an integer linear constraint problem with the form (1) over 64 variables.

Before developing BLT, we had generated various instances of this problem, including both unsatisfiable and satisfiable cases. We applied several SMT solvers, including Yices [7], CVC4 [3], and Z3 [6], as well as an evaluation version of Gurobi[1], an industrial linear programming solver. The solvers we tried were incapable of solving all but the most trivial instances of this problem without giving additional hints. This was true even when after running some of the problems for months on selected solvers. In contrast, BLT is able to solve the majority of the problems in under a second.

## 2 Preliminaries

Throughout this section we use bold capital letter symbols such as $\mathbf{A}$ to denote matrices, bold lower case letters such as $\boldsymbol{x}$ to denote vectors, and undecorated lower case letters to denote scalars. The components of a vector $\boldsymbol{x} \in \mathbb{R}^n$ are denoted by $\boldsymbol{x}_i$ with $i \in \{1, \ldots, n\}$. We use Greek letters such as $\theta$ to denote functions that assign values to only some of the coordinates (e.g., the function $\theta : Y \to \mathbb{Z}$ with $Y \subseteq \{\,1 \ldots, n\}$ only assigns values to indices in $Y$). A vectors $\boldsymbol{x}$ is then just an assignment where all the indices have been assigned.

**Lattices.** A *lattice* is a discrete additive subgroup of a Euclidean space. A subset $\mathcal{L} \subset \mathbb{R}^n$ is a lattice if and only if there is a collection of linearly independent vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{R}^n$ such that $\mathcal{L} = \{\sum_{i=1}^n c_i \boldsymbol{v}_i \mid c_i \in \mathbb{Z}\}$. Such a collection is called a *basis* of $\mathcal{L}$ and the number $n$ is called the *rank*. A lattice generally has many different bases. *Lattice reduction* is a term used to describe algorithms for producing a basis that is "short" and "nearly orthogonal", a property that is extremely useful in practice [12].

Given a fixed basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$ of $\mathcal{L}$ and a partial assignment $\theta : Y \to \mathbb{Z}$ with $Y \subseteq \{\,1 \ldots, n\}$, we define the set of lattice elements $\mathcal{L}_\theta \subseteq \mathcal{L}$ as follows:

$$\mathcal{L}_\theta := \left\{ \sum_{i=1}^{n} c_i \boldsymbol{v}_i \mid c_i \in \mathbb{Z},\, i \in \mathrm{dom}(\theta) \Rightarrow c_i = \theta(i) \right\}.$$

We call $\mathcal{L}_\theta$ a *sublayer* of $\mathcal{L}$. The set $\mathcal{L}_\theta$ should be thought of as the subset of lattice elements which remain after a partial assignment of coefficients is made.

Our algorithm for finding *integer* solutions will rely on an underlying solver for systems of *real* solutions. To model this, we define the real-affine linear space $\mathcal{L}_\theta^{\mathbb{R}}$ containing $\mathcal{L}_\theta$:

$$\mathcal{L}_\theta^{\mathbb{R}} := \left\{ \sum_{i=1}^{n} c_i \boldsymbol{v}_i \mid c_i \in \mathbb{R},\, i \in \mathrm{dom}(\theta) \Rightarrow c_i = \theta(i) \right\}.$$

A fundamental problem in the theory of lattices is the *closest vector problem* (CVP) [12]. We introduce it here because it is very closely related to our approach for solving bounded ILP. CVP has the following form: given a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a point $\boldsymbol{q} \in \mathbb{R}^n$, find a vector $\boldsymbol{z} \in \mathcal{L}$ that is *closest* to $\boldsymbol{q}$; i.e. a vector for which $\|\boldsymbol{z} - \boldsymbol{q}\|$ is minimal. Such a closest vector must exist, but it may be difficult to compute and may not be unique. The problem of deciding whether such a vector exists within a given bound is known to be NP-hard [9], though polynomial-time algorithms are known if the rank of $\mathcal{L}$ is fixed [14].

[1] Available at `http://www.gurobi.com/`.

**$L^\infty$ Metric.** In our decision procedure, we work in $\mathbb{R}^n$ with a different metric than the usual Euclidean metric. The $L^\infty$ norm is defined by $\|\boldsymbol{x}\|_\infty := \max\{|x_i| \mid i = 1, \ldots, n\}$. It determines a metric via $d_\infty(\boldsymbol{x}, \boldsymbol{y}) := \|\boldsymbol{x} - \boldsymbol{y}\|_\infty$.

With $L^\infty$, the set of points whose distance from a given point $\boldsymbol{p}$ is at most $r$ (i.e. the closed ball of radius $r$ around $\boldsymbol{p}$) is a hypercube (the $n$-dimensional analogue of a square) each of whose faces is orthogonal to a coordinate axis. Explicitly,

$$\{\boldsymbol{x} \in \mathbb{R}^n \mid d_\infty(\boldsymbol{x}, \boldsymbol{p}) \leq r\} = \{(x_1, \ldots, x_n) \mid p_i - r \leq x_i \leq p_i + r\}\,.$$

The distance metric can be extended to subsets of $\mathbb{R}^n$ by taking it to be the minimum distance between any two points in the subsets, i.e., for $X, Y \subseteq \mathbb{R}^n$,

$$d_\infty(X, Y) := \min_{(\boldsymbol{x}, \boldsymbol{y}) \in X \times Y} d_\infty(\boldsymbol{x}, \boldsymbol{y})$$

We remark that when the sets $X$ and $Y$ can be defined by systems of linear equality and inequality constraints over rational coefficients, then the $L^\infty$ distance can be calculated using linear programming techniques by observing that:

$$\begin{aligned} d_\infty(X, Y) &= \min\{\ d_\infty(\boldsymbol{x}, \boldsymbol{y}) \mid \boldsymbol{x} \in X,\ \boldsymbol{y} \in Y\ \} \\ &= \min\left\{\ \max_i\{|x_i - y_i|\} \mid \boldsymbol{x} \in X,\ \boldsymbol{y} \in Y\ \right\} \\ &= \min\{\ t \mid |x_i - y_i| \leq t,\ \boldsymbol{x} \in X,\ \boldsymbol{y} \in Y\ \} \\ &= \min\{\ t \mid x_i - y_i \leq t,\ -(x_i - y_i) \leq t,\ \boldsymbol{x} \in X\, \boldsymbol{y} \in Y\ \} \end{aligned} \tag{2}$$

The last line is a real linear optimization problem over the free variables in the systems of equations used to define $X$ and $Y$.

# 3 The BLT Decision Procedure

To describe our decision procedure, we assume that we are attempting to check whether the constraints below are satisfiable:

$$\boldsymbol{l} \leq \mathbf{A}\boldsymbol{x} \leq \boldsymbol{u}. \tag{P1}$$

We let $n$ denote the number of free integer variables in $\boldsymbol{x} \in \mathbb{Z}^n$, and let $m$ denote the number of constrained linear forms. The coefficients are rational, so $\mathbf{A} \in \mathbb{Q}^{m \times n}$ and $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{Q}^m$.

Without loss of generality we assume that $n \leq m$ and that $\mathbf{A}$ has rank $n$ (full rank). In case the problem at hand is such that $n > m$ and/or that $\mathbf{A}$ is less than full rank, one can compute a basis of the column space, say $\mathbf{A}'$, that meets the requirement (cf. [4], § 2.7.1). The new system $\boldsymbol{l} \leq \mathbf{A}'\boldsymbol{y} \leq \boldsymbol{u}$ is equisatisfiable with the original and solutions of the new system determine one or more solutions of the original.

Geometrically, we can think of $\boldsymbol{l}$ and $\boldsymbol{u}$ as opposite corners of an $m$-dimensional hyperrectangle defined by

$$\mathcal{C} := \{\boldsymbol{z} \in \mathbb{R}^m \mid \boldsymbol{l}_i \leq \boldsymbol{z}_i \leq \boldsymbol{u}_i\}.$$

We refer to this as the *constraint set* of the problem. Without loss of generality we may scale the rows of (P1) so that the width of $\mathcal{C}$ is the same along every axis, i.e.

$$\boldsymbol{u}_i - \boldsymbol{l}_i = \boldsymbol{u}_j - \boldsymbol{l}_j \quad \forall\, i, j \in \{1, \ldots, m\}.$$

Note that this transformation makes $\mathcal{C}$ a *hypercube*. We let $d_{\mathcal{C}}$ denote the common width, and let $r_{\mathcal{C}} = d_{\mathcal{C}}/2$ denote the corresponding radius of the hypercube.

The problem given by (P1) can also be characterized as trying to find a common point in both a hypercube and a lattice. The columns of $\mathbf{A}$, regarded as vectors, generate a lattice. Let $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n\}$ denote the column vectors of $\mathbf{A}$ and define:

$$\mathcal{L} := \{a_1\boldsymbol{b}_1 + \cdots + a_n\boldsymbol{b}_n \in \mathbb{R}^m \mid a_i \in \mathbb{Z}\}$$

By our assumption that $\mathbf{A}$ has full rank, $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$ are linearly independent, and there is a one-to-one correspondance between elements in $\mathcal{L} \cap \mathcal{C}$ and satifying assignments to (P1). It follows that checking the satisfiability of (P1) is equivalent to deciding whether $\mathcal{L} \cap \mathcal{C}$ is non-empty.

The set $\mathcal{L} \cap \mathcal{C}$ is guaranteed to be finite as a lattice must have a finite number of elements in any space with a bounded volume. Due to the one-to-one correspondance, the number of solutions to (P1) must be finite as well, and hence a procedure capable of enumerating the elements in $\mathcal{L} \cap \mathcal{C}$ can be used as a decision procedure for checking the satisfiability of (P1).

Before describing such a procedure, we note that a nice feature of the lattice and hypercube formulation is that it provides a simple way to estimate the number of satisfying assignments. The *volume* of a lattice $\mathcal{L}$ can be defined in several ways (see [12], §5), but the simplest computationally is $\mathrm{vol}(\mathcal{L}) = |\det \mathbf{B}|$ where the columns of $\mathbf{B}$ generate $\mathcal{L}$. Then, the number of elements of $\mathcal{L} \cap \mathcal{C}$ is approximately $\mathrm{vol}\,\mathcal{C} \,/\, \mathrm{vol}\,\mathcal{L}$. We will use this to compute the number of expected solutions for the JPEG preimage problems described in the Section 4.

## 3.1 Enumerating Lattice Elements

We now turn our attention to the problem of enumerating the elements $\mathcal{L} \cap \mathcal{C}$. Recall that, without loss of generality, we have taken $\mathcal{C}$ to be a hypercube. Let $\boldsymbol{p}$ denote the geometric center of $\mathcal{C}$:

$$\boldsymbol{p} := (\frac{\boldsymbol{l}_1 + \boldsymbol{u}_1}{2}, \ldots, \frac{\boldsymbol{l}_m + \boldsymbol{u}_m}{2}).$$

With respect to the $L^\infty$ metric, $\mathcal{C}$ is a closed ball of radius $r_{\mathcal{C}}$, centered at $\boldsymbol{p}$, and hence the elements in $\mathcal{L} \cap \mathcal{C}$ are precisely those that are at most a distance $r_{\mathcal{C}}$ from $\boldsymbol{p}$.

Algorithms for finding lattice elements that are close to a given point have been extensively studied and many algorithmic approaches to it exist; see [2] for a good survey. We have developed a complete search procedure by adapting the Schnorr-Euchner algorithm for computing the closest vector point to a lattice [15].

We model our search procedure as a non-deterministic transition rule on partial assignments to the vectors $\boldsymbol{x}$. The procedure begins with the empty assignment $\emptyset$, and incrementally assigns values to variables in $\boldsymbol{x}$. If the transition rule terminates with a complete assignment $\boldsymbol{u} \in \mathbb{Z}^n$, then $\boldsymbol{u}$ is a solution to the constraint problem.

$$(\textbf{Split}) \qquad \theta \;\Rightarrow\; \theta \cup \{\, j \mapsto s \,\} \textbf{ where } \begin{cases} j \in \{\, 1, \ldots, n \,\} \setminus \mathrm{dom}(\theta) \\ s \in \mathbb{Z} \textbf{ s.t. } \mathcal{L}^{\mathbb{R}}_{\theta \cup \{\, j \mapsto s \,\}} \cap \mathcal{C} \neq \emptyset \end{cases}$$

This rule takes a partial assignment $\theta$, and extends it with an additional binding $j \mapsto s$ such that the real-affine linear space $\mathcal{L}^{\mathbb{R}}_{\theta \cup \{\, j \mapsto s \,\}}$ of the resulting assignment intersects with $\mathcal{C}$. This

rule models backtracking implicitly; at each step, we may find that there is no legal value $s$ to assign $j$. If this occurs, our procedure must backtrack to a previous step, and explore an alternative assignment.

We can show that the set of lattice points in $\mathcal{C}$ can be enumerated by applying **Split** transitively starting from $\emptyset$. Before stating the theorem, we first observe that each point $\boldsymbol{x} \in \mathcal{L}$ can be expressed as the weighted sum of the columns in $\mathbf{A}$, (i.e., $\boldsymbol{x} = \mathbf{A}\boldsymbol{u}$ for some unique $\boldsymbol{u} \in \mathbb{Z}^n$).

**Theorem 3.1.** *For each vector $\boldsymbol{u} \in \mathbb{Z}^n$, $\mathbf{A}\boldsymbol{u} \in \mathcal{C}$ iff. there is a derivation $\emptyset \Rightarrow^+ \boldsymbol{u}$.*

*Proof.* To see that $\emptyset \Rightarrow^+ \boldsymbol{u}$ implies $\mathbf{A}\boldsymbol{u} \in \mathcal{C}$, observe that the proceeding step must have shown that $\mathcal{L}^{\mathbb{R}}_{\boldsymbol{u}} \cap \mathcal{C} \neq \emptyset$. Since $\boldsymbol{u}$ is a complete assignment, $\mathcal{L}^{\mathbb{R}}_{\boldsymbol{u}} = \{\, \mathbf{A}\boldsymbol{u} \,\}$, and hence $\mathbf{A}\boldsymbol{u} \in \mathcal{C}$.

To see that $\mathbf{A}\boldsymbol{u} \in \mathcal{C}$ implies $\emptyset \Rightarrow^+ \boldsymbol{u}$, observe that for all partial assignments $\theta$, $\mathcal{L}^{\mathbb{R}}_{\theta} \cap \mathcal{C} \neq \emptyset$ implies that $\emptyset \Rightarrow^+ \theta$ by induction on the number of bindings in $\theta$. For a complete assignment $\boldsymbol{u}$, $\mathcal{L}^{\mathbb{R}}_{\boldsymbol{u}} = \{\, \mathbf{A}\boldsymbol{u} \}$. Hence, if $\mathbf{A}\boldsymbol{u}$ is in $\mathcal{C}$, then $\mathcal{L}^{\mathbb{R}}_{\boldsymbol{u}}$ is a non-empty subset of $\mathcal{C}$ and $\varnothing \Rightarrow^+ \boldsymbol{u}$. □

We note that the above theorem holds regardless of the order in which we choose the values of $j$ in **Split**. We only need to consider all valid assignments to $s$ once we have chosen $j$. An implementation then has a choice in which it can use different heuristics to search for an assignment. We will briefly describe the heuristics used by BLT in Section 3.2.

The previous theorem shows that the transition rule is sound and complete from a logical point of view, to show that it is computable we prove the following:

**Theorem 3.2.** *The set of partial assignments $\theta$ such that $\emptyset \Rightarrow^+ \theta$ is finite and computable.*

*Proof.* As each application of **Split** adds an additional binding to the substitution, the number of applications along any path is bounded by $n$. To show that the set of $\theta$ is finite, we must show that the number of potential values of $s$ used to instantiate **Split** is both finite and computable. More precisely, we must prove that there are at most a finite number of integers $s \in \mathbb{Z}$ such that

$$\mathcal{L}^{\mathbb{R}}_{\theta \cup \{\, j \mapsto s \,\}} \cap \mathcal{C} \neq \emptyset. \tag{3}$$

Observe that for any $u, k \in \mathbb{Z}$ with $k \neq 0$, the affine set $\mathcal{L}^{\mathbb{R}}_{\theta \cup \{\, j \mapsto u+k \,\}}$ can be obtained by shifting the set $\mathcal{L}^{\mathbb{R}}_{\theta \cup \{\, j \mapsto u \,\}}$ by a multiple $k$ of the basis vector $\boldsymbol{b}_j$. As $\boldsymbol{b}_j$ is linearly independent from the other basis vectors, it follows that $\mathcal{L}^{\mathbb{R}}_{\theta \cup \{\, j \mapsto u \,\}}$ and $\mathcal{L}^{\mathbb{R}}_{\theta \cup \{\, j \mapsto u+k \,\}}$ are disjoint and separated by some positive distance $k \times d_{\theta,j}$ where $d_{\theta,j}$ is the distance between the adjacent hyperplanes $\mathcal{L}^{\mathbb{R}}_{\theta \cup \{\, j \mapsto 0 \,\}}$ and $\mathcal{L}^{\mathbb{R}}_{\theta \cup \{\, j \mapsto 1 \,\}}$ As the distance between any two points $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}$ is at most $d_{\mathcal{C}}$, it follows that the number of distinct $s$ satisfying (3) is at most $d_{\mathcal{C}}/d_{\theta,j}$. Moreover, as both $\mathcal{C}$ and $\mathcal{L}^{\mathbb{R}}_{\theta \cup \{\, j \mapsto s \,\}}$ are convex, there must be a bounded interval $s \in \{\, l, l+1, \ldots, u-1, u \,\}$ of values satisfying (3).

Rather than compute the bounds explicitly, we compute the $L^\infty$-distance between $\mathcal{L}^{\mathbb{R}}_{\theta}$ and the point $\boldsymbol{p}$ at the center of $\mathcal{C}$ using the reduction to linear programming described at the end of the Section 2 in equation (2). This reduction to linear programming allows one to find an assignment $\boldsymbol{y} \in \mathbb{R}^n$ so that $\mathbf{A}\boldsymbol{y}$ is one of the points in $\mathcal{L}^{\mathbb{R}}_{\theta}$ with minimal distance to $\boldsymbol{p}$. We can then start by considering for $s$ the points $\{\, \lfloor \boldsymbol{y}_j \rfloor, \lfloor \boldsymbol{y}_j - 1 \rfloor, \ldots \}$ and $\{\, \lceil \boldsymbol{y}_j \rceil, \lceil \boldsymbol{y}_j + 1 \rceil, \ldots \}$ until we have explored all the assignments in the set $\{\, l, l+1, \ldots, u \,\}$. □

### 3.2 Implementation Decisions

Turning the previous section into a working and efficient procedure involves many more choices and details than we have room to describe. We would like to indicate, however a couple choices we have made in implementing BLT.

**Search Strategy.** In implementing the transition system, we have chosen to adopt a strategy similar to Schnorr and Euchner in [15]. We use the LLL algorithm [12] to generate a reduced basis, and fix the basis vectors by sorting in order of decreasing $L^2$ magnitude. We then proceed by applying **Split** in a depth first order with the sequence of $j$'s chosen according to our basis order. The variables with the largest magnitude are typically the most-constrained variables in our problems, as they have the largest distance between adjacent sublayers. Choosing the most constrained variable is a common strategy in constraint satisfication, and we have found the strategy effective in this case as well.

The other choice we have with split is to consider which values of $s$ to explore. To maximize the likelihood of finding a satisfying assignment, we would like to choose a value for $s$ that imposes the least constraints on subsequent assignments. This could be done by choosing an assignment to $s$ that maximizes the volume of the intersection between the hypercube $\mathcal{C}$ and real-affine set $\mathcal{L}^{\mathbb{R}}_{\theta\cup\{\, j\mapsto s\,\}}$.

Unfortunately, we do not know of an efficient way to compute the $s$ with the maximal volume[2], but we have developed a proxy that works well in practice. As alluded to in the proof of Theorem 3.2, we use linear programming to find an initial assignment to $s$ that minimizes the $L^\infty$-distance between the center of the hypercube $\boldsymbol{p}$ and $\mathcal{L}^{\mathbb{R}}_{\theta\cup\{\, j\mapsto s\,\}}$. Since the distance between the sublayer and center point is minimal, we can expect that the volume of the sublayer within the hypercube should be maximal or near maximal. If this assignment is found infeasible, and we backtrack, then we explore adjacent assignments $s+\delta, s-\delta, s+2\delta, \ldots$, where $\delta = \pm 1$ depending on orientation, in order of increasing distance.

**Layer-point distance.** Due to efficiency concerns as well as implementation issues with linking GMP with other Haskell code that BLT is linked against, we compute the distance using a conventional linear programming solver, GLPK [1], which uses IEEE double precision floating point for its calculations. If the distance calculation is inaccurate, there is the potential to prune a sublayer that is mistaken for being slightly too far away, and consequently BLT may incorrectly return UNSAT. In cases where it returns SAT, the model is checked against the problem for certainty. In principle the distance calculations can be done using exact arithmetic[3] or arbitrary-precision floating point arithmetic, but we have not attempted to do so yet.

## 4 JPEG Preimage

To validate our work, we have applied BLT to the problem of computing preimages from JPEG decompression. To simplify exposition, we will restrict our attention to *monochrome* JPEG images that consist of a single $8 \times 8$ block of pixels. In experiments, we have also applied BLT to color images; color problems involve more variables and constraints, but are otherwise similar to the monochrome case. Our restriction to images that are 8 by 8 does not effect scalability

[2] In [8], the authors show that the related problem of computing the volume of the intersection of the unit cube and a rational halfspace is #P-hard.

[3] GLPK supports this directly.

either; JPEG compresses each $8 \times 8$ block of pixels within an image independently, so computing the preimage for a larger image is the same as finding preimages for multiple independent 8 by 8 blocks.

When compressing an image, each pixel ranges from 0 to 255 where 0 corresponds to black and 255 corresponds to white. To compress an image, JPEG performs the following steps [13]:

1. Each pixel value is shifted by $-128$ so that the pixel values range between $-128$ and 127.
2. A 2d discrete cosine transform (DCT) is applied to each block that transforms the coordinate space from the image pixel values to the frequency domain. This has the effect of separating out the image components by frequency, so that course grained qualities such as overall brightness are represented distinctly from more fine-grained fluctuations. For a given input block $I$, we denote the frequency representation by $F = \mathrm{dct}_2(I)$. A 2d DCT is obtained by first applying a 1d DCT to each column in the image, and then applying a 1d DCT to each row in the image.
3. A quantization step is performed in which each coordinate in the frequency representation $F$ is quantized to the nearest multiple of an associated value in a *quantization matrix* $\mathbf{Q}_{lvl} \in \mathbb{Z}^{8\times 8}$. The quantization matrix is constructed so that high-frequency components are rounded to more course grained values than low-frequency components.

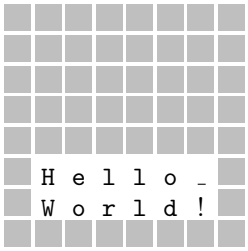
   JPEG allows users some control over the tradeoff between the compression ratio and image quality by providing a parameter $lvl$, called the "quality level" and ranges from 1 to 100. The coefficients in $\mathbf{Q}_{lvl}$ grow as the quality level $lvl$ decreases.

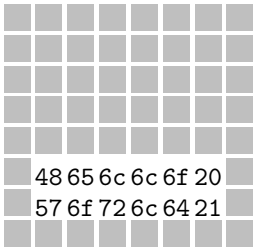
   Given the output of the frequency transform $\mathrm{dct}_2(I)$, the output of the quantization step consists of the rounded quotient $C = \mathrm{round}(\mathrm{dct}_2(I)./\mathbf{Q}_{lvl})$. The division is a pointwise (Hadamard) division, rather than an inverse linear transform.
4. Finally, a variant of Huffman compression is performed that compresses the quantized coefficients into a string of bits. The compression is lossless, and designed to represent the quantized coefficients in a small number of bits.

JPEG decompression just runs these steps in reverse order starting with Huffman decompression. As Huffman compression is lossless and can be directly inverted, for our constraint satisfaction problem we begin with the rounded quantized coefficients $C$, and the resulting image $I$ can be obtained by computing:

$$I = \mathrm{round}(\mathrm{idct}_2(\mathbf{Q}_{lvl}. * C) + 128)$$

As an example preimage problem, suppose that we are looking for any image containing "Hello World!" encoded as ASCII text within a block.




To construct a bounded ILP problem, we take the constraints above and generate the lower and upper bounds needed so that the final rounding function will return an image satisfying

the constraints. This gives us the two matrices $\mathbf{L}$ and $\mathbf{U}$ below:

$$\begin{bmatrix} -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 \\ -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 \\ -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 \\ -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 \\ -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 \\ -0.5 & 71.5 & 100.5 & 107.5 & 107.5 & 110.5 & 31.5 & -0.5 \\ -0.5 & 86.5 & 110.5 & 113.5 & 107.5 & 99.5 & 32.5 & -0.5 \\ -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 & -0.5 \end{bmatrix} \begin{bmatrix} 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 \\ 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 \\ 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 \\ 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 \\ 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 \\ 255.5 & 72.5 & 101.5 & 108.5 & 108.5 & 111.5 & 32.5 & 255.5 \\ 255.5 & 87.5 & 111.5 & 114.5 & 108.5 & 100.5 & 33.5 & 255.5 \\ 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 & 255.5 \end{bmatrix}$$

With these steps, the problem then reduces finding a coefficients $\mathbf{C} \in \mathbb{Z}^{8\times 8}$ such that:

$$\mathbf{L} \leq \text{idct}_2(\mathbf{Q}_{lvl} .* \mathbf{C}) + 128 \leq \mathbf{U}.$$

Both the Hadamard product and inverse DCT are linear transformations, and we evaluate the inverse DCT by evaluating the coefficients to IEEE double floating point precision. This allows us to construct a bounded ILP problem from the equation above.

For this problem, we can compute an estimate of the number of solutions by dividing the size of space bounded by $\mathbf{L}$ and $\mathbf{U}$ by the density of the lattice $\mathbf{A}_{lvl}$ generated by the quantization step and idct function. In Figure 1, we plot the number of solutions on a logarithmic scale. This figure illustrates how dramatically the estimated number of solutions changes with respect to the quality level. At quality levels 98 and higher, the number of expected solutions exceeds $10^{100}$; while less than 1 solution is expected at quality level 25 for the same constraint. In the extreme case at quality level 1, one would only expect to find solutions in roughly 3 out of $10^{90}$ problems with a similarly sized bounds.

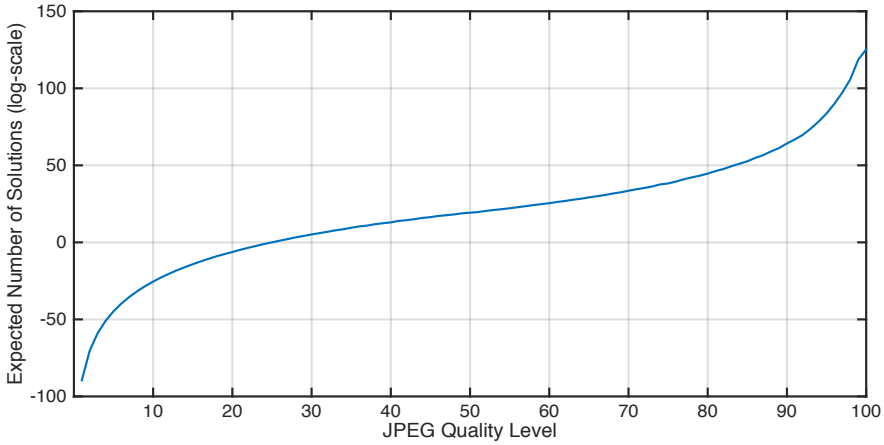



Figure 1: Number of expected solutions

We first tried applying CVC4, Yices, and Z3 to the problem by encoding the problem in a format supported by the solver. We used SMT-LIB for CVC4 and Z3, and Yices' native format for it. Unfortunately, none of these tools were able to solve any of the problems at quality levels from 1 to 100 within a 1 hour cutoff for each problem. We also ran all three solvers without success for over two months on problems at quality levels 99 and 1.

We have had much better success when BLT is applied to these problems. In our testing, BLT has been able to find solutions to all problems at quality level 27 and higher. BLT found that

the problems at quality levels 1 through 18 were *unsatisfiable*. We include a chart of BLT's runtime in Figure 2. In the plot, solid points denote problems for which BLT returns SAT, whereas × points denote problems where BLT returns UNSAT. The large number of problems with roughly constant runtime (mostly on the right side of the plot) have the property that the Babai point (mentioned in section 3.2) is already a solution and thus almost no search is needed. In the filled region between levels 19 and 26, BLT failed to terminate in the 1 hour cutoff.

We should note that BLT is performing floating point arithmetic, and so when BLT returns unsat there is a risk that floating point rounding error lead to BLT detecting a branch was infeasible when it was in fact infeasible. This may also account for some of the performance gap, between BLT and the above solvers, but we suspect it is unlikely that precision alone can account for the more than 6 orders of magnitude runtime difference we have observed above.

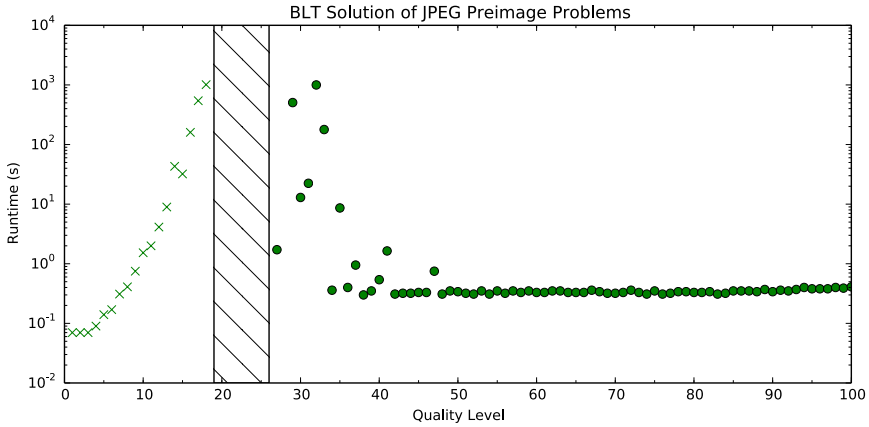



Figure 2: BLT Runtime vs. Problem Level

# 5 Related and Future Work

**Related Work**. Our work builds upon the well-known Schnorr-Euchner algorithm for solving the closest vector problem. Prior to developing BLT, we used the closest-vector solver in `fplll` [10] to solve problems. It uses the $L^2$ norm, and thus is not a decision procedure. However, we found that it could often find satisfying assignments at quality levels greater than 60 despite the lack of completeness.

`LattE` is a program for enumerating lattice points in a rational polytope [5]. This is asking strictly more than the question we've discussed. `LattE` is competitive with commercial branch-and-bound solvers, the same solvers which perform poorly on the DCT problems in section 4. We attempted to evaluate `LattE` on our DCT problems, but found that it would crash given a 16 GB memory limit.

**Future Development**. We are still working on developing BLT, and have plans to continue testing it on a wider variety of challenge problems. We plan to enable the $L^\infty$ distance calculation to use exact arithmetic. We also plan to explore ways to solve problems with one-sided

bounds via heuristics, and ways to infer unsatisfiable subsets of constraints so that BLT can be integrated into an SMT solver. Finally, we would like to integrate techniques from SAT community such as conflict-driven-clause learning into the ILP search performed by BLT. The later should help improve its performance on hard problem instances.

More broadly, it also seems interesting to explore how the calculus and heuristics that BLT uses can be integrated into other ILP solvers, such as those based on cutting planes (e.g. [11]). Those algorithms are able to work on more general problems as they do not require explicit upper and lower bounds for all linear expressions. On the other hand, BLT appears to be more effective at making effective decisions within the search.

*Acknowledgements.* The authors would like to thank Dejan Jovanović, Grant Passmore, and the reviewers for comments that helped improve this paper.

# References

[1] GLPK (GNU linear programming kit). Available at `http://www.gnu.org/software/glpk`.

[2] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *Information Theory, IEEE Transactions on*, 48(8):2201–2214, Aug 2002.

[3] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *CAV 2011*, volume 6639 of *Lecture Notes in Computer Science*. Springer, 2011.

[4] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[5] Jesus de Loera, Raymond Hemmecke, Jeremiah Tauzer, and Ruriko Yoshida. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation*, 38(4):1273–1302, 2004.

[6] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS 2008*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[7] Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Computer-Aided Verification (CAV'2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, July 2014.

[8] M.E. Dyer and A.M. Frieze. The complexity of computing the volume of a polyhedron. *SIAM J. Computation*, 17:967–974, 1988.

[9] P. van Emde-Boas. *Another NP-complete partition problem and the complexity of computing short vectors in a lattice*. Report. Department of Mathematics. University of Amsterdam, 1981.

[10] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In *Coding and Cryptology - Third International Workshop, IWCC 2011*, volume 6639 of *Lecture Notes in Computer Science*, pages 159–190. Springer, 2011.

[11] Dejan Jovanovic and Leonardo de Moura. Cutting to the chase – solving linear integer arithmetic. *J. Autom. Reasoning*, 51(1):79–108, 2013.

[12] H. Lenstra Jr. Lattices. *Algorithmic number theory*, pages 127–181, 2008.

[13] William Pennebaker and Joan Mitchell. *JPEG: Still Image Data Compression Standard*. Springer, 1993.

[14] C. P. Schnorr. A hierarchy of polynomial lattice basis reduction algorithms. *Theoretical Computer Science*, 53:201–224, 1987.

[15] C.P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1–3):181–199, 1994.