HUGE UNIMODULAR n-FOLD PROGRAMS*

SHMUEL ONN† AND PAULINE SARRABEZOLLES‡

Abstract. Optimization over \(l \times m \times n\) integer 3-way tables with given line-sums is NP-hard already for fixed \(l = 3\), but is polynomial time solvable with both \(l, m\) fixed. In the huge version of the problem, the variable dimension \(n\) is encoded in binary, with \(t\) layer types. It was recently shown that the huge problem can be solved in polynomial time for fixed \(t\), and the complexity of the problem for variable \(t\) was raised as an open problem. Here we solve this problem and show that the huge table problem can be solved in polynomial time even when the number \(t\) of types is variable.

The complexity of the problem over 4-way tables with variable \(t\) remains open, where all we know is that the associated decision problem is in NP intersect coNP. Our treatment goes through the more general class of huge \(n\)-fold integer programming problems. We show that huge integer programs over \(n\)-fold products of totally unimodular matrices can be solved in polynomial time even when the number \(t\) of brick types is variable.

Key words. integer programming, multiway table, Graver basis, bin packing, NP intersect coNP, unimodular


DOI. 10.1137/151004227

1. Introduction. Consider the following optimization problem over 3-way tables with given line-sums:

\[
\min \left\{ wx : x \in \mathbb{Z}_{+}^{l \times m \times n}, \sum_{i} x_{i,j,k} = e_{j,k}, \sum_{j} x_{i,j,k} = f_{i,k}, \sum_{k} x_{i,j,k} = g_{i,j} \right\}.
\]

It is NP-hard already for \(l = 3\); see [3]. Moreover, every bounded integer program can be isomorphically represented in polynomial time for some \(m\) and \(n\) as some \(3 \times m \times n\) table problem; see [4]. However, when both \(l, m\) are fixed, it is solvable in polynomial time [2, 9, 12] and, in fact, in time which is cubic in \(n\) and linear in the binary encoding of \(w, e, f, g\); see [8]. Assume throughout then that \(l, m\) are fixed, and regard each table as a tuple \(x = (x^1, \ldots, x^n)\) consisting of \(n\) many \(l \times m\) layers. Following [13], call the problem huge if the variable number \(n\) of layers is encoded in binary. As in [13], we are then given \(t\) types of layers, where each type \(k\) has its cost matrix \(w^k \in \mathbb{Z}^{l \times m}\), column-sums vector \(e^k \in \mathbb{Z}^m_+\), and row-sums vector \(f^k \in \mathbb{Z}^l_+\). In addition, we are given positive integers \(n_1, \ldots, n_t, n\) with \(n_1 + \cdots + n_t = n\), all encoded in binary. A feasible table \(x = (x^1, \ldots, x^n)\) must then have first \(n_1\) layers of type 1, next \(n_2\) layers of type 2, and so on, with last \(n_t\) layers of type \(t\). The special case of \(t = 1\) is the case of symmetric tables, where all layers have the same cost, row sums, and column sums, and the classical (nonhuge) table problem occurs as the...
special case of \( t = n \) and \( n_1 = \cdots = n_t = 1 \). Note that for each \( k \), the set of possible layers of type \( k \) is
\[
\left\{ z \in \mathbb{Z}^{l \times m} : \sum_i z_{i,j} = e^k_j, \sum_j z_{i,j} = f^k_i \right\}
\]
and may have cardinality which is exponential in the binary encoding of \( e^k, f^k \). So it is not off hand clear how to even write down a single table, let alone optimize.

We note that huge versions of classical problems have been studied in the literature in other situations; see, for example, [6] and the references therein for such studies in the context of bin packing, where items come in types and multiplicity per type. The problem on \( l \times m \times n \) tables is sometimes called the \textit{three index transportation problem}, introduced by Motzkin [11]. Then the small fixed sides \( l, m \) may represent suppliers and commodities, whereas the huge variable side \( n \) may represent a huge number of consumers which can be partitioned into a relatively small number \( t \) of consumer types, where consumers of the same type share the same profile.

The huge table problem was recently considered in [13], where it was shown, combining results of [2, 9, 12] on Graver bases and results of [5, 6] on integer cones, that it can be solved in polynomial time for fixed \( t \). The complexity of the problem for variable \( t \) was raised as an open problem. Here we solve this problem and show that the huge table problem can be solved in polynomial time even when \( t \) is variable.

\textbf{Theorem 1.1.} The huge \( 3 \)-way table problem with a variable number \( t \) of types can be solved in time which is polynomial in \( t \) and in the binary encoding of \( w^k, e^k, f^k, g, n_k \).

It was further shown in [13] that the huge \( d \)-table problem over \( m_1 \times \cdots \times m_{d-1} \times n \) tables with \( m_1, \ldots, m_{d-1} \) fixed and \( n \) variable can also be solved in polynomial time for any fixed number \( t \) of types. Interestingly, we do not know whether Theorem 1.1 could be extended to this more general situation, and the complexity of the huge \( d \)-way table problem with variable \( t \) remains open, already for \( 3 \times 3 \times 3 \times n \) tables.

Theorem 1.1 follows from broader results which we proceed to describe. The class of \textit{n-fold integer programming} problems is defined as follows. The \textit{n-fold product} of an \( s \times d \) matrix \( A \) is the following \((dn) \times (dn)\) matrix, with \( I \) the \( d \times d \) identity:
\[
A^{[n]} := \begin{pmatrix}
I & I & \cdots & I \\
I & A & \cdots & 0 \\
0 & A & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & A
\end{pmatrix}.
\]

The classical \textit{n-fold integer programming} problem is then the following:
\[
(1.1) \quad \min \left\{ wx : x \in \mathbb{Z}^{dn}, A^{[n]}x = b, \ l \leq x \leq u \right\},
\]
where \( w \in \mathbb{Z}^{dn} \), \( b \in \mathbb{Z}^{d+sn} \), and \( l, u \in \mathbb{Z}^{dn} \) with \( \mathbb{Z}_\infty := \mathbb{Z} \cup \{\pm \infty\} \). For instance, optimization over multiway tables is an \textit{n-fold program}, as explained later.

Our starting point is the following result on classical \textit{n-fold integer programming}, established in [2, 9], building on results of [1, 10, 14]. See the monograph [12] for a detailed treatment of the theory and applications of \textit{n-fold integer programming}.

\textbf{Proposition 1.2.} For fixed matrix \( A \), the classical \textit{n-fold integer programming} problem (1.1) can be solved in time polynomial in \( n \) and the binary encoding of \( w, l, u, b \).
This result holds more generally if the identity $I$ in the definition of $A^{[n]}$ is replaced by another fixed matrix $B$. Moreover, recently, in [8], it was shown that the problem can be solved in time which is cubic in $n$ and linear in the binary encoding of $w, b, l, u$.

The vector ingredients of an $n$-fold integer program are naturally arranged in bricks, where $w = (w^1, \ldots, w^n)$ with $w^i \in \mathbb{Z}^d$ for $i = 1,\ldots, n$, and likewise for $l, u$, and where $b = (b^0, b^1, \ldots, b^n)$ with $b^0 \in \mathbb{Z}^d$ and $b^i \in \mathbb{Z}^s$ for $i = 1,\ldots, n$. Following [13] again, call an $n$-fold integer program huge if $n$ is encoded in binary. More precisely, we are now given $t$ types of bricks, where each type $k = 1,\ldots, t$ has its cost $w^k \in \mathbb{Z}^d$, lower and upper bounds $l^k, u^k \in \mathbb{Z}^d$, and right-hand side $b^k \in \mathbb{Z}^s$. Also given are $b^0 \in \mathbb{Z}^d$ and positive integers $n_1,\ldots, n_t, n$ with $n_1 + \cdots + n_t = n$, all encoded in binary. A feasible point $x = (x^1,\ldots, x^n)$ now must have first $n_1$ bricks of type 1, next $n_2$ bricks of type 2, and so on, with last $n_t$ bricks of type $t$. Classical $n$-fold integer programming occurs as the special case of $t = n$ and $n_1 = \cdots = n_t = 1$, and symmetric $n$-fold integer programming occurs as the special case of $t = 1$. In [13] it was shown that for fixed $t$ the problem is polynomial time solvable. Here we show that for totally unimodular $A$ it is efficiently solvable even for variable $t$.

**Theorem 1.3.** Let $A$ be a fixed totally unimodular matrix, and consider the huge $n$-fold program over $A$ with a variable number $t$ of types. Then the optimization problem can be solved in time polynomial in $t$ and the binary encoding of $w^t, l^t, u^t, b^t, n$.

We remark that the binary encoding of the data makes the problem solved here particularly difficult. If the $n_k$ are encoded in unary, then the problem can be solved by classical $n$-fold integer programming theory. If all the data is encoded in unary, then, like the classical knapsack problem, it can be solved by dynamic programming.

We also remark that $n$-fold integer programming was extended in [7] to four-block $n$-fold integer programming. In [13] the huge version of this extension was also discussed, and it was shown that such problems admit compact presentations of optimal solutions. However, whether or not huge four-block $n$-fold programs can be solved in polynomial time, even if the blocks are totally unimodular, remains open.

The rest of the article is organized as follows. In section 2 we discuss the feasibility problem, which is easier than the optimization problem and admits a more efficient algorithm. In section 3 we discuss the optimization problem, using the results on feasibility. We conclude in section 4 with further discussion of tables.

### 2. Feasibility

In this section we consider the feasibility problem for huge $n$-fold integer programs:

$$\{ x \in \mathbb{Z}^{dn} : A^{[n]}x = b, \ l \leq x \leq u \} \text{ nonempty?}$$

We begin with the case of symmetric programs, with one type, so that $t = 1$, over an $s \times d$ totally unimodular matrix $A$. So the data here consists of the top right-hand side $b^0 \in \mathbb{Z}^d$, and for all bricks the same lower and upper bounds $l^1, u^1 \in \mathbb{Z}_\infty^d$ and same right-hand side $b^1 \in \mathbb{Z}^s$. Then the set in question can be written as

$$\left\{ x \in \mathbb{Z}^{dn} : \sum_{i=1}^{n} x^i = b^0, \ Ax^i = b^1, \ l^1 \leq x^i \leq u^1, \ i = 1,\ldots, n \right\}.$$  \hfill (2.1)

We have the following lemma.

**Lemma 2.1.** Let $A$ be totally unimodular. Then the set in (2.1) is nonempty if and only if $Ab^0 = nb^1$ and $nl^1 \leq b^0 \leq nu^1$, and this can be decided in time polynomial in the binary encoding of $n, l, u, b$, even when $A$ is a variable part of the input.
Proof. Suppose first that the set in (2.1) contains a feasible point \( x = (x^1, \ldots, x^n) \).
Then \( Ab^0 = A \sum_{i=1}^n x^i = \sum_{i=1}^n Ax^i = nb^1 \), and \( nu^1 \leq b^0 = \sum_{i=1}^n x^i \leq nu^1 \). For the converse we use induction on \( n \). Suppose \( b^0 \) satisfies the conditions. If \( n = 1 \), then \( x^1 := b^0 \) is a feasible point in (2.1). Suppose now \( n \geq 2 \). Consider the system in \( y \)

\[
l^1 \leq y \leq u^1, \quad Ay = b^1, \quad (n-1)l^1 \leq b^0 - y \leq (n-1)u^1.
\]

Then \( y = \frac{1}{n}b^0 \) is a real solution to this system, and therefore, since \( A \) is totally unimodular, there is also an integer solution \( x^n \) to this system. In particular, \( Ax^n = b^1 \) and \( l^1 \leq x^n \leq u^1 \). Let \( \tilde{b}^0 := b^0 - x^n \). Then \( A\tilde{b}^0 = A(b^0 - x^n) = (n-1)b^1 \), and \( (n-1)l^1 \leq \tilde{b}^0 - x^n \leq (n-1)u^1 \). It therefore now follows by induction that there is an integer solution \((x^1, \ldots, x^{n-1})\) to the \((n-1)\)-fold program

\[
\left\{ x \in \mathbb{Z}^{d(n-1)} : \sum_{i=1}^{n-1} x^i = \tilde{b}^0, \ Ax^i = b^1, \ l^1 \leq x^i \leq u^1, \ i = 1, \ldots, n-1 \right\}.
\]

Then \( \sum_{i=1}^n x^i = \tilde{b}^0 + x^n = b^0 \), and therefore \( x := (x^1, \ldots, x^{n-1}, x^n) \) is a feasible point in (2.1). The statement about the computational complexity is obvious. \( \square \)

We proceed with the general case of \( t \) types. So the data now consists of \( b^k \in \mathbb{Z}^d \)
and, for \( k = 1, \ldots, t \), lower and upper bounds \( l^k, u^k \in \mathbb{Z}^d \), right-hand side \( b^k \in \mathbb{Z}^d \), and positive integer \( n_k \), with \( n_1 + \cdots + n_t = n \). We denote by \( I_1 \cup \cdots \cup I_t = \{1, \ldots, n\} \) the natural partition with \( |I_k| = n_k \). So the set in question can be now written as

\[
(2.2) \quad \left\{ x \in \mathbb{Z}^{dn} : \sum_{i=1}^{n} x^i = b^0, \ Ax^i = b^k, \ l^k \leq x^i \leq u^k, \ k = 1, \ldots, t, \ i \in I_k \right\}.
\]

We have the following theorem asserting that when \( A \) is totally unimodular, the feasibility problem is decidable in polynomial time even if the number \( t \) of types is variable. The algorithm underlying the proof uses only classical \( n \)-fold integer programming and avoids the heavy results of [6] on integer cones used in [13]. The general idea is that, using Lemma 2.1, testing feasibility of the huge \( n \)-fold program can be reduced to testing feasibility of a suitable compressed classical \( t \)-fold program.

**Theorem 2.2.** Let \( A \) be a fixed totally unimodular matrix, and consider the huge \( n \)-fold program over \( A \) with variable number \( t \) of types. Then it is decidable in time polynomial in \( t \) and the binary encoding of \( l^k, u^k, b^k, n_k \) whether the set in (2.2) is nonempty.

**Proof.** Consider the following set of points of a classical \( t \)-fold integer program:

\[
(2.3) \quad \left\{ y \in \mathbb{Z}^{dt} : \sum_{k=1}^{t} y^k = b^0, \ Ay^k = n_kb^k, \ n_k l^k \leq y^k \leq n_k u^k, \ k = 1, \ldots, t \right\}.
\]

We claim that (2.2) is nonempty if and only if (2.3) is nonempty, which can be decided within the claimed time complexity by Proposition 1.2 on classical \( n \)-fold theory.

Thus it remains to prove the claim. First, suppose \( x \) is in (2.2). Define \( y \) by setting \( y^k := \sum_{i \in I_k} x^i \) for \( k = 1, \ldots, t \). Then we have \( \sum_{k=1}^{t} y^k = \sum_{i=1}^{n} x^i = b^0 \), \( Ay^k = A \sum_{i \in I_k} x^i = n_kb^k \), and \( n_k l^k \leq y^k = \sum_{i \in I_k} x^i \leq n_k u^k \), so \( y \) is in (2.3). Conversely, suppose \( y \) is in (2.3). For \( k = 1, \ldots, t \) consider the symmetric \( n_k \)-fold program

\[
\left\{ (x^i : i \in I_k) \in \mathbb{Z}^{d_{n_k}} : \sum_{i \in I_k} x^i = y^k, \ Ax^i = b^k, \ l^k \leq x^i \leq u^k, \ i \in I_k \right\}.
\]
Since $y$ is in (2.3) we have that $Ay^k = n_kb^k$ and $n_kb^k \leq y^k \leq n_ku^k$. Therefore, by Lemma 2.1, this program is feasible and has a solution $(x^i : i \in I_k)$. Let $x = (x^1, \ldots, x^n)$ be obtained by combining the solutions of these $t$ programs. Then we have $\sum_{i=1}^n x^i = \sum_{k=1}^t y^k = b^0$, and $Ax^i = b^k, l^k \leq x^i \leq u^k$ for $k = 1, \ldots, t$ and $i \in I_k$, so $x$ is in (2.2). This completes the proof of the claim and the theorem. ⌦

3. Optimization. In this section we consider the optimization problem for huge $n$-fold programs:

$$\min \left\{ \sum_{k=1}^t \sum_{i \in I_k} w^k x^i : x \in \mathbb{Z}^{dn}, \sum_{i=1}^n x^i = b^0, Ax^i = b^k, l^k \leq x^i \leq u^k, k = 1, \ldots, t, i \in I_k \right\}.$$ 

The optimization problem is harder than the feasibility problem in that we need to actually produce an optimal solution if one exists. Since the problem is huge, meaning that $n$ is encoded in binary, we cannot explicitly even write down a single point $x \in \mathbb{Z}^{dn}$ in polynomial time. But it turns out that we can present $x$ compactly as follows. For $k = 1, \ldots, t$ the set of all possible bricks of type $k$ is the following:

$$S^k := \{ z \in \mathbb{Z}^d : Az = b^k, l^k \leq z \leq u^k \}.$$ 

We assume for simplicity that $S^k$ is finite for all $k$, which is the case in most applications, such as in multiway table problems. Let $\lambda^k := (\lambda^k_z : z \in S^k)$ be a nonnegative integer tuple with entries indexed by points of $S^k$. Each feasible point $x = (x^1, \ldots, x^n)$ gives rise to $\lambda^1, \ldots, \lambda^t$ satisfying $\sum(\lambda^k_z : z \in S^k) = n_k$, where $\lambda^k_z$ is the number of bricks of $x$ of type $k$ which are equal to $z$. Let the support of $\lambda^k$ be $\text{supp}(\lambda^k) := \{ z \in S^k : \lambda^k_z \neq 0 \}$. Then a compact presentation of $x$ consists of the restrictions of $\lambda^k$ to $\text{supp}(\lambda^k)$ for all $k$. While the cardinality of $S^k$ may be exponential in the binary encoding of the data $b^k, l^k, u^k$, it turns out that a compact presentation of polynomial size always exists. The following theorem was shown in [13] using the recent computationally heavy algorithm of [6], which builds on [5]. The proof is quite involved, and the reader is referred to [13, Theorem 1.3] for details.

**Proposition 3.1.** For fixed $d$ and $t$, the huge $n$-fold integer optimization problem with $t$ types, over an $s \times d$ matrix $A$ which is part of the input, can be solved in polynomial time. That is, in time polynomial in the binary encoding of $A, l^k, u^k, b^k, n_k$, either it can be asserted that the problem is infeasible, or a compact presentation $\lambda^1, \ldots, \lambda^t$ of an optimal solution with $|\text{supp}(\lambda^k)| \leq 2^d$ for $k = 1, \ldots, t$ can be computed.

We now show that for a totally unimodular matrix, we can solve the huge problem even for variable $t$, extending both the above result and classical $n$-fold theory. The general idea is the following. First, we use classical $n$-fold integer programming theory and find an optimal solution $y = (y^1, \ldots, y^t)$ of a suitable compressed classical $t$-fold program. Then, for $k = 1, \ldots, t$, we use the brick $y^k$ of that optimal solution to construct a huge symmetric $n_k$-fold program, and using Proposition 3.1, we find a compact presentation of an optimal solution $(x^i : i \in I_k)$ of that program. The combination of these compact presentations for $k = 1, \ldots, t$ provides the desired compact presentation of the optimal solution of the original huge $n$-fold program.

**Theorem 1.3 (restated).** Let $A$ be a fixed totally unimodular matrix, and consider the huge $n$-fold program over $A$ with a variable number $t$ of types. Then the optimization problem can be solved in time polynomial in $t$ and the binary encoding of $u^k, l^k, u^k, b^k, n_k$. 

Proof. Consider the following classical \( t \)-fold integer optimization problem:

\[
\min \left\{ \sum_{k=1}^{t} w^k y^k : y \in \mathbb{Z}^{d_t} \mid \sum_{k=1}^{t} y^k = b^0, A y^k = n_k b^k, n_k t^k \leq y^k \leq n_k u^k, k = 1, \ldots, t \right\}.
\]

By Proposition 1.2 on classical \( n \)-fold theory we can either assert that the problem is infeasible, or obtain an optimal solution \( y \), within the claimed time complexity. As shown in the proof of Theorem 2.2, if this problem is infeasible, then so is the original program, and we are done. So assume we obtained an optimal solution \( y \).

For \( k = 1, \ldots, t \), consider the symmetric \( n_k \)-fold program

\[
\min \left\{ \sum_{i \in I_k} w^k x^i : \left( x^i : i \in I_k \right) \in \mathbb{Z}^{d_{n_k}^t} \mid \sum_{i \in I_k} x^i = y^k, A x^i = b^k, t^k \leq x^i \leq u^k, i \in I_k \right\}.
\]

As shown in the proof of Theorem 2.2, this program is feasible. Since this is a huge symmetric program, that is, with a single type, by Proposition 3.1 we can compute in polynomial time a compact presentation \( \lambda^k \) with \(|\text{supp}(\lambda^k)| \leq 2^d\) of an optimal solution \( (x^i : i \in I_k) \in \mathbb{Z}^{d_{n_k}^t} \). (In fact, any point in that program has the same objective function value \( \sum_{i \in I_k} w^k x^i = w^k y^k \) and is optimal for that program.) Then \( \lambda^1, \ldots, \lambda^t \) obtained from all of these programs provide a compact presentation of a point \( x = (x^1, \ldots, x^n) \) feasible in the original program. We claim this \( x \) is optimal. Suppose indirectly there is a better point \( \bar{x} \), and define \( \bar{y}^k = \sum_{i \in I_k} \bar{x}^i \) for all \( k \).

Then we have

\[
\sum_{k=1}^{t} w^k \bar{y}^k = \sum_{k=1}^{t} \sum_{i \in I_k} w^k \bar{x}^i < \sum_{k=1}^{t} \sum_{i \in I_k} w^k x^i = \sum_{k=1}^{t} w^k y^k,
\]

contradicting the optimality of \( y \) in the \( t \)-fold program. So \( \lambda^1, \ldots, \lambda^t \) indeed provide a compact presentation of an optimal solution of the given huge \( n \)-fold program. \( \square \)

We make the following remark. The algorithm of Theorem 2.2 for the feasibility problem involves only one application of the classical \( n \)-fold integer programming algorithm of Proposition 1.2. In contrast, the algorithm of Theorem 1.3 is much heavier and, in addition to one application of classical \( n \)-fold integer programming, uses \( t \) times the algorithm of Proposition 3.1 for huge \( n \)-fold integer programming with one type, which in turn uses the heavy algorithm for integer cones of \([6]\).

4. Tables. We now return to tables. Consider first \( 3 \)-way \( l \times m \times n \) tables. Index each table as \( x = (x^1, \ldots, x^n) \) with \( x^i = (x^i_1, \ldots, x^i_{l,m}) \). Then the table problem is the \( n \)-fold program with matrix \( A^{[n]}_{l,m} \) with \( A_{l,m} \) the vertex-edge incidence matrix of the bipartite graph \( K_{l,m} \). Indeed, then \( \sum_{i=1}^{n} x^i = g \) provides the vertical line-sum equations, and \( A_{l,m} x^i = b^i \) with \( b^i = (e^i, f^i) \) provides the column- and row-sum equations for \( i \in I_k \). Since \( A_{l,m} \) is totally unimodular, Theorem 1.3 implies our following claimed result.

**Theorem 1.1** (restated). The huge \( 3 \)-way table problem with a variable number \( t \) of types can be solved in time which is polynomial in \( t \) and in the binary encoding of \( w^k, e^k, f^k, g, n_k \). In particular, deciding whether there is a huge table with variable number of types is in \( P \).

Let us continue with \( 4 \)-way \( k \times l \times m \times n \) tables. Index each table as \( x = (x^1, \ldots, x^n) \) with each \( x^k \) a \( k \times l \times m \) layer. Then the table problem is the \( n \)-fold
program with matrix $A^{[n]}$, where $A = A^{[m]}_{k,l}$. Now, unfortunately, for $k, l, m \geq 3$, the matrix $A$ is not totally unimodular. Therefore, the results of the previous sections do not apply, and we remain with the results of [13], which are as follows.

**Proposition 4.1.** The huge 4-way table problem with fixed number $t$ of types is solvable in time polynomial in the binary encoding of $w^k, n_k$ and the line-sums. Moreover, deciding whether there is a huge table with $t$ variable is in $NP$ intersect coNP.

The contrast between Theorem 1.1 and Proposition 4.1 motivates the following.

**Open problem.** What is the complexity of deciding feasibility of huge 4-way tables with a variable number of types? In particular, is it in P for $3 \times 3 \times 3 \times n$ tables?

**Acknowledgment.** The authors thank the referees for their helpful suggestions.

**REFERENCES**


