A Bound on Matrix-Vector Products for $(0,1)$-Matrices via Gray Codes

Andrew A. Anda
Department of Computer Science
St. Cloud State University
aanda@eeyore.stcloudstate.edu

Abstract

We demonstrate a differencing method for reducing the number of arithmetic operations within a $(0,1)$-matrix vector product. We use that method to more efficiently perform Gray code matrix vector products. Using one or more Gray code matrix vector products, we show how to reduce the number of arithmetic operations for a general $(0,1)$-matrix vector product. Finally, we show how $(0,1)$-matrix vector products may be applied to the performance of certain restricted classes of more general matrix vector products.

Introduction

A $(0,1)$-matrix (also identified as zero-one or Boolean) is a rectangular matrix for which each element of the matrix has the value of either one or zero.

$(0,1)$-matrices arise from problems in a variety of application areas. Prominent examples include:

• adjacency matrices for simple graphs, representing the connectivity relationships between vertices; (Rosen, Michaels, Gross, Grossman, & Shier, 2000)

• term-by-document matrices generated by binary vector space model based computational information retrieval (BVIR) algorithms and applications such as search engines; an element w_{ij} in a weight matrix $W \in \{0,1\}^{n \times m}$ is set to 1 if a term t_j appears in document D_i, with 0 otherwise; (Dominich, 2001)

• matrix calculus applications in statistics and econometrics which generate special (0,1)-matrices such as selection, permutation, commutation, elimination, duplication, and shifting matrices; (TURKINGTON, 2002)

In fact, any general matrix may be decomposed into the linear combination of

conformal (0,1)-matrices. For a general real matrix $A \in \mathbb{R}^{m\times n}$,

$$A = \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_{ij} E_{ij}, \left\{ E_{ij} = \mathbf{e}_i \mathbf{e}_j^T, 1 \le i \le m; 1 \le j \le m \right\} \tag{1}$$

The matrix-vector product operation,

$$Ax = y, \tag{2}$$

represents the product of a matrix, $A \in \{0,1\}^{m\times n}$, by the vector, $x \in \mathbb{R}^n$, to yield the vector, $y \in \mathbb{R}^m$. More generally, the vectors may actually be over any algebraic ring for which addition and multiplication is closed and well defined.

The general matrix-vector product is an example of *level 2* operation, and as such, it exhibits a quadratic complexity. (A doubly nested loop is required for its evaluation). There is little that can be done to improve the efficiency of the general matrix-vector product. The operations may be blocked for modest performance gains for large matrices on a hierarchical memory architecture. But, we know of no way to reduce the number of scalar additions and multiplications. For certain classes of structured matrices, however, we can attain $O(n \ln n)$ by applying the FFT to the calculation. A structured matrix is one which can be fully characterized by $O(n)$ parameters. In fact the product of a rank one matrix and a vector can be performed in $O(n)$ if we know the two vectors that formed the rank-one matrix, $Ax = yz^T x$ for some vectors y and z. We will be considering general $(0,1)$-matrices though.

A Gray code is a circular ordering of all 2^n binary strings of length n in which adjacent strings differ in exactly one bit, i.e. all adjacent strings have a *Hamming distance* of unity. One can label the vertices of a *hypercube* with bitstrings such that all *adjacent* neighbors have a bitstring differing in exactly one bit. Each Gray code corresponds to a *Hamiltonian cycle* about the vertices of the hypercube.

A Differencing Method

The general Matrix-vector product $Ax = y$, $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$, is computed with the following equation,

$$y_i = \sum_{j=1}^{n} \alpha_{ij} x_j, 1 \le i \le m. \tag{3}$$

Computing this requires an algorithm having a doubly nested loop, with two possible orderings. We'll consider the ordering which performs an inner product in the inner loop, so that each outer loop completely calculates one element of the resultant vector y. With either ordering, the number of additions and multiplications each is mn. For a $(0,1)$-matrix though, $\alpha_{ij} \in \{0,1\}, \forall i,j$. So, either the product $\alpha_{ij}x_j$ contributes to the sum as x_j, in the case of $\alpha_{ij} = 1$, or it is skipped, in the case of

$\alpha_{ij} = 0$. This implies that the maximum possible amount of work is mn additions which would occur if the matrix were all ones.

Now, lets consider computing the difference between two elements of y, y_i and y_k:

$$\begin{aligned} y_k - y_i &= \sum_{j=1}^{n} \alpha_{kj} x_j - \sum_{j=1}^{n} \alpha_{ij} x_j && (4) \\ &= \sum_{j=1}^{n} (\alpha_{kj} x_j - \alpha_{ij} x_j) && (5) \\ &= \sum_{j=1}^{n} x_j (\alpha_{kj} - \alpha_{ij}) && (6) \end{aligned}$$

Now lets consider computing y_k if y_i has already been computed.

$$\begin{aligned} y_k &= y_i + \sum_{j=1}^{n} x_j (\alpha_{kj} - \alpha_{ij}) && (7) \\ &= y_i + \sum_{j=1}^{n} x_j (d_j), d_j = \alpha_{kj} - \alpha_{ij} && (8) \end{aligned}$$

This implies that apart from the first element of y to be computed, each subsequent element of y can be computed as the sum of a previously computed element with the inner product of x with the difference vector of the two rows of A, d. Let A_k be the kth row of A. Then we have saved $\|A_k\|_1 - \|d\|_1 - 1$ operations in computing y_k. If $\|d\|_1 + 1 < \|A_k\|_1$, the savings is positive. (the offset of 1 is due to the addition of y_i) $\|d\|_1$ is also the *Hamming distance* between two rows of A.

We can deal with any duplicated rows in A by loading its previously computed value at the expense of no additions.

If the remaining rows are unique, what's the lowest number of additions that we need? Because there are no duplicated rows, there must be a Hamming distance of at least one between each row of A and each other row. The optimum in this case would be for there to be, for every row of A, some other row of A of unit Hamming distance. We have already defined a sequence of Boolean vectors which satisfies this condition, a Gray code.

A Gray code q bits wide consists of a sequence of $r = 2^q$ bitstrings. Using the differencing method above, a $r \times q$ Gray code matrix vector product can be computed in a maximum of r additions. If one substitutes a load for an addition when there are at most 1 nonzero bits in the row, the maximum becomes $r - q - 1$ required additions.

If we have a general $(0,1)$-matrix matrix having q columns and any number of rows, we can first precompute a scratch y using the Gray code Matrix vector product with x. Then, for each row of the general matrix A, we load into y the entry in the

scratch y corresponding to the bitpattern for that row. The only additions are from the precomputation of the scratch y from the Gray code matrix. If $m > r$, we have a greater reduction in the maximum number of additions for the Gray code matrix itself.

If $n > q$, one of two conditions can hold:

- $n \bmod q = 0$ in which case we will use n/q Gray code matrix stripes across A with a subsequent sum across the stripes for all m.
- $n \bmod q \neq 0$ in which case we will use $\lfloor n/q \rfloor$ Gray code matrix stripes of width q, and one additional Gray code matrix stripe of width $n \bmod q$ with a subsequent sum across the stripes for all m.

Let us define $w(q, m, n)$ as the number of additions required for a general $(0,1)$-matrix. Then for an arbitrary m and n, ignoring the $q + 1$ savings from above,

$$w(q,m,n) = \begin{cases} 2^n & \text{if } n < q \\ 2^q & \text{if } n = q \\ \frac{n}{q}\left(2^q + m\right) - m & \text{if } n > q \text{ and } 0 = n \bmod q \\ \lfloor \frac{n}{q} \rfloor 2^q + 2^{n \bmod q} + m\left(\lceil \frac{n}{q} \rceil - 1\right) & \text{if } n > q \text{ and } 0 \neq n \bmod q \end{cases} \tag{9}$$

The minimizing value of q may be determined informally by selecting the smallest w for a reasonably small set of q's clustered around $\lg m$. More formally, we can evaluate $\frac{\partial w}{\partial q}$ and find where it has the value of zero. So, using *Maple* on the expression corresponding to the third condition,

$$\frac{\partial w}{\partial q} = \frac{n 2^q \ln 2}{q} - \frac{n(2^q + m)}{q^2} \tag{10}$$

We find $0 = \frac{\partial w}{\partial q}$ at

$$q = \frac{LambertW(me^{-1}) + 1}{\ln 2}, \text{where } LambertW(x)e^{LambertW(x)} = x \tag{11}$$

Exploiting the $(0,1)$-Matrix Vector Product

Equation 1 has little practical value unless one considers the special case where the number of distinct α_{ij} terms is smaller than the smallest of the two indices. Then all of the E_{ij} matrices will be added together forming denser $(0,1)$-matrices for each set of identical α_{ij} terms. We can then refactor equation 1 as:

$$A = \sum_{i=1}^{k} \beta_i B_i, \tag{12}$$

Where there are k distinct entries, β_i in A and each B_i represents a distinct $(0,1)$-matrix.

For a single β, the matrix vector product can be written as:

$$Ax = y = \beta Bx = B(\beta x), \tag{13}$$

In equation 13, we see the scalar product moved from being $O(mn)$ to $O(n)$. Equations 12 and 13 can be combined:

$$Ax = y = \left(\sum_{i=1}^{k} \beta_i B_i\right) x = \sum_{i=1}^{k} B_i(\beta_i x). \tag{14}$$

Lets consider the case where $k = 2$. The elements of the matrix A are either of only two values, $\alpha_{ij} \in \{\beta_1, \beta_2\}$. The cardinality of the set that α_{ij} is drawn from is the same as that of the $(0,1)$-matrix. So, rather than use a pair of $(0,1)$-matrices as shown in equation 14, we can use a single one if we first affinely transform the equation with an appropriate scaling and translation. The elements of A can be scaled simply by multiplying by a scaling factor: γA. Scaling equation 2 we get:

$$\gamma Ax = \gamma y. \tag{15}$$

However to translate the values of A, we need to create a conformal translation matrix, $T = \{1\}^{m\times n}$, which will serve as a basis for the uniform translation of all the entries of A. A unit translation of A towards $+\infty$ is represented by the sum $A + T$. Translating equation 2 we get:

$$(A+T)x = Ax + Tx = y^{(A)} + y^{(T)}, \text{ where } y_j^{(T)} = \sum_{i=1}^{n} x_i \tag{16}$$

We can translate the entries of A an arbitrary distance δ. We then augment equation 16:

$$(A+\delta T)x = Ax + \delta Tx = y^{(A)} + \delta y^{(T)}, \tag{17}$$

We can now combine equations 15 and 16 to represent the full affine transformation:

$$\gamma(A+\delta T)x = \gamma(Ax + \delta Tx) = \gamma y^{(A)} + \gamma\delta y^{(T)} = \gamma(y^{(A)} + \delta y^{(T)})). \tag{18}$$

Equation 18 can be used to transform any two-valued matrix vector product into a $(0,1)$-matrix based vector product. For example, a common two valued A is $\{-1,1\}^{m\times n}$. This can be transformed into a $(0,1)$-matrix based vector product by setting $\gamma = 2$ and $\delta = -\frac{1}{2}$:

$$\begin{aligned}
\{-1,1\}^{m\times n}x &= \gamma\left(\{0,1\}^{m\times n} + \delta\{1\}^{m\times n}\right)x && (19)\\
&= \gamma(A+\delta T)x && (20)\\
&= \gamma(Ax + \delta Tx) && (21)\\
&= \gamma(y^{(A)} + \delta y^{(T)})) && (22)\\
&= 2(y^{(A)} - \frac{1}{2}y^{(T)})) && (23)\\
&= 2y^{(A)} - y^{(T)})). && (24)
\end{aligned}$$

Conclusion

We were able to demonstrate a differencing method for reducing the number of arithmetic operations within a $(0,1)$-matrix vector product. We then used that method to more efficiently perform Gray code matrix vector products. Then using one or more Gray code matrix vector products, we showed how to reduce the number of arithmetic operations for a general $(0,1)$-matrix vector product. We then showed how $(0,1)$-matrix vector products may be applied to the performance of certain restricted classes of more general matrix vector products.

References

Dominich, S. (2001). *Mathematical foundations of information retrieval.* Dordrecht, The Netherlands: Kluwer.

Rosen, K. H., Michaels, J. G., Gross, J. L., Grossman, J. W., & Shier, D. R. (Eds.). (2000). *Handbook of discrete and combinatorial mathematics.* Boca Raton FL: crc.

TURKINGTON, D. A. (2002). *Matrix calculus and zero-one matrices.* New York: Cambridge University Press. (Statistical and econometric applications)