Theoretical Computer Science 19 (1982) 337–341
North-Holland Publishing Company

NOTE

IS BINARY ENCODING APPROPRIATE FOR THE PROBLEM-LANGUAGE RELATIONSHIP?

Nimrod MEGIDDO
Statistics Department, Tel Aviv University, Tel Aviv, Israel

Communicated by M. Nivat
Received January 1982

Abstract. It is proved that there exist encoding schemes which are arbitrarily as efficient as the binary encoding (in terms of compactness and arithmetic operations), with respect to which Khachiyan's algorithm for Linear Programming is exponential. This constitutes an objection to the standard translation of problems into languages via the binary encoding.

When we speak about the complexity of a problem in which numbers are involved, we usually think of a formalization as a language recognition problem where the numbers are encoded in binary. Most of the people believe that the work of Khachiyan [2] has resolved the question of the complexity of linear programming. However, if we wish to be precise, Khachiyan has proven that the language of linear inequalities in binary encoding belongs to the class P. The complexity of linear programming as a problem (rather than a language) still constitutes an interesting open question.

A major open question is the following: Is there an algorithm and is there a polynomial $p(m, n)$ such that every set of m linear inequalities in n variables can be solved by the algorithm in less than $p(m, n)$ arithmetic operations? We shall call such an algorithm genuinely-polynomial. Special linear programming problems for which genuinely-polynomial algorithms are known are the max-flow problem, the shortest-path problem and the assignment problem. One may argue that the distinction between polynomial and genuinely-polynomial is not essential since the amount of time required for the arithmetic operations is at least proportional to the logarithms of the numbers. More specifically, let A denote the maximal absolute value of a coefficient in a given set of m inequalities in n variables with integral coefficients. Khachiyan's algorithm works in $q(m, n, \log A)$ time where q is a certain polynomial, whereas a genuinely-polynomial algorithm requires at least $p(m, n) \cdot \log A$ time. So, in what sense are the two notions distinct? The answer is simple. A genuinely-polynomial algorithm runs in polynomial time whenever the arithmetic operations can be carried out in polynomial time, whereas Khachiyan's

0304-3975/82/0000–0000/$02.75

algorithm may require exponential time even when the arithmetic operations are polynomial.

We will enhance the argument of the preceding paragraph by proving the following:

Theorem. *For every $\varepsilon > 0$ there exists an encoding scheme (i.e., a one-to-one mapping $E: \mathbb{Z} \to \{0, 1\}^*$) such that*
(i) *For every integer $N \neq 0$ the length of $E(N)$, denoted by $l(N)$, satisfies $l(N) \leq (1+\varepsilon)(\lfloor \log_2 |N| \rfloor + 1)$.*
(ii) *Comparisons and arithmetic operations can be carried out in time polynomial in the $l(N)$'s (uniformly in ε).*
(iii) *There are infinitely many numbers N such that $l(N) = O(\text{loglog } N)$.*

The theorem claims that there exist encoding schemes which are almost as efficient as the binary encoding (in the sense of (i)) and are also convenient for manipulating arithmetics. The third property implies that algorithms like Khachiyan's run in exponential time if such encoding schemes are being used. This is because when those numbers which are represented compactly (i.e., in $O(\text{loglog } N)$ bits) appear in the set of inequalities then the factor $\log N$ which appears in the runtime is exponential in terms of loglog N.

Proof. We will first prove the claim of the theorem with respect to $\varepsilon = 1$ and then indicate how to extend it for any $\varepsilon > 0$.

For simplicity let us work with positive numbers. Let N be any positive integer and let $B(N) = b_1 b_2 \ldots b_k$ denote its binary representation, i.e., $b_i \in \{0, 1\}$ $(i = 1, \ldots, k)$, $b_1 = 1$ and $N = \sum_{i=1}^{k} b_i 2^{k-i}$. The binary expansion consists of blocks of consecutive ones and blocks of consecutive zeros. The sequence of *lengths* of these blocks characterizes the number N. In our scheme we will represent these lengths in binary and separate them by commas. Later, the commas will be eliminated.[1] Specifically, the encoding $E(N)$ is recursively defined as follows. If $b_i = 1$ $(i = 1, \ldots, k)$ then we define $E(N) = B(k)$; otherwise, let $j_1 = \min\{i: b_i = 0\}$ and $j_2 = \min\{i: i > j_1, b_i = 1\}$ and define $E(N) = B(j_1 - 1), B(j_2 - j_1), E(b_{j_2} \ldots b_k)$. For example, the number 63631 which is encoded in binary as 1111100011111100 will be encoded in our scheme as 101, 11, 110, 10. An obvious way to eliminate the commas is as follows. We will utilize only the odd-numbered bits for representing the block-lengths. The even-numbered bits will represent the commas according to the convention that a comma exists where an even-numbered bit contains a one. For example our number 63631 will be encoded without the commas as 1000111011101001100. The number of bits required by this encoding scheme is less than twice the number of bits required in binary. The worst-cases are numbers

[1] An efficient method for eliminating the commas is described by Even and Rodeh [1]. Instead of doubling the number of bits, their method adds to an n-bit number $O(\log n)$ more bits to eliminate a delimiter.

like 170, whose binary expansion is 10101010. In our scheme it is 1, 1, 1, 1, 1, 1, 1, 1 or, without the commas, 11111111111111. It is obvious that claim (i) is true in our case. Also, claim (iii) is proved by the numbers of the form $N = 2^K - 1$. This encoded in binary is a string of K ones and in our scheme it requires $2\lfloor \log_2 K \rfloor + 1$ bits which is O(loglog N).

We will now show how to perform the arithmetic operations and comparisons using our encoding scheme. For simplicity of notation we will work with the version that uses commas.

To compare two positive numbers N_1, N_2 we operate as follows. Suppose $E(N_1) = A_1, \ldots, A_r$ and $E(N_2) = B_1, \ldots, B_s$ where the A_i's and B_i's are binary representations of block-lengths in the expansion of N_1 and N_2, respectively. Let $A = \sum A_i$ and $B = \sum B_i$. If $A > B$ $(A < B)$ then $N_1 > N_2$ $(N_1 < N_2)$. Suppose $A = B$. Now, if $A_1 > B_1$ $(A_1 < B_1)$ then $N_1 > N_2$ $(N_1 < N_2)$. If also $A_1 = B_1$ then if $A_2 > B_2$ $(A_2 < B_2)$ then $N_1 < N_2$ $(N_1 > N_2)$ and so on. Obviously, the comparison between two numbers in our encoding scheme reduces to comparisons in binary of block-lengths. We conclude that comparisons can be done in linear time.

Additions can also be carried out in linear time. We first note that a number A_i (or B_i) represents a block of ones if i is odd and a block of zeros if i is even. We start from the blocks represented by A_r and B_s. Consider, for example, the case where r is even and s is odd. The last block in the expansion of the number $N_1 + N_2$ consists of ones. If $A_r \neq B_s$ then the length of that block is equal to $\min(A_r, B_s)$. If $A_r = B_s$ then we need to compare A_{r-1} with B_{s-1} in order to tell the length of the first block; if $A_{r-1} = B_{s-1}$ then we need to proceed to A_{r-2} and B_{s-2} and so on. However, in any case the addition $N_1 + N_2$ amounts to no more than $r + s$ additions and comparisons between A_i's and B_j's.

Multiplication is naturally more complicated but can also be carried out in polynomial time. Consider first the case where we need to multiply two numbers N_1, N_2 encoded as above, however assuming $s = 1$. In other words, $N_2 = 2^{B_1} - 1$. In this case we first find $E(N_1 \cdot (N_2 + 1))$ and then subtract N_1. The multiplication of N_1 by $N_2 + 1$ is simple: If r is odd then $E(N_1 \cdot (N_2 + 1)) = A_1, \ldots, A_r, B_1$ and if r is even then $E(N_1 \cdot (N_2 + 1)) = A_1, \ldots, A_{r-1}, (A_r + B_1)$. Thus, in this case $E(N_1 \cdot N_2)$ is found in linear time. It is easy to see that, in general, $E(N_1 \cdot N_2)$ can be found by multiplying N_1 by the odd-numbered blocks of N_2, shifting and adding, in time which is $O((\sum \log A_i)(\sum \log B_i))$. There are of course faster ways for multiplication in our scheme which resemble the faster methods for multiplication in binary.

To prove the theorem for an arbitrary $\varepsilon > 0$, we modify the encoding as follows. We select an integer M which is sufficiently large, depending on ε. In the encoding scheme $E_M(N)$ which we define below the bits whose locations are at 1 and 2 (mod $(M+2)$) play a distinguished role. They indicate how the contents of the succeeding M bits should be interpreted. This is explained in detail below. An example is given in the Appendix.

Suppose $B(N) = b_1 b_2 \ldots b_k$. Consider the first block of consecutive ones. If its length L is less than or equal to M then the first $M+2$ bits in $E_M(N)$ will be

$01b_1b_2 \ldots b_M$. The prefix 01 indicates that what follows is copied from the binary representation of N. Suppose L is greater than M and let $a_1 \ldots a_{iM}$ denote the binary expansion of L using an integral multiple of M bits (possibly with leading zeros). The first $i(M+2)$ bits in $E_M(N)$ will then be $11a_1 \ldots a_M 10a_{M+1} \ldots a_{2M} 10a_{2M+1} \ldots a_{3M} 10a_{(i-1)M+1} \ldots a_{iM}$. The prefix 11 in bits 1, 2 indicates that at that point we start to describe in binary representation the length of a block of consecutive ones from $B(N)$. Similarly, we will use the prefix 00 to signify the start of a binary representation of a length of a block of zeros. The pair 10 indicates continuation of the same interpretation from the preceding group of M bits. Inductively, suppose we have translated all the bits $b_1, \ldots, b_j$ from $B(N)$ to $E_M(N)$ and we are now at the bit numbered $q(M+2)+1$. We now consider the bits $b_{j+1}, \ldots, b_{j+M+1}$. If they are identical then we write 00 or 11 (depending on the contents of these identical bits) in the bits $q(M+2)+1$ and $q(M+2)+2$ of $E_M(N)$. We then look at the number $L = \min\{i: i > j, b_i \neq b_{j+1}\} - j - 1$. This is the length of the maximal block of identical bits starting at b_{j+1}. Consider the expansion of L in an integral multiple of M bits (possibly with leading zeros). We now copy this expansion into $E_M(N)$ starting at bit $q(M+2)+3$, using the continuation code 10 in $(q+1)(M+2)+1$, $(q+1)(M+2)+2$, $(q+2)(M+2)+1$, $(q+2)(M+2)+2$, etc., if necessary. If, on the other hand, the bits $b_{j+1}, \ldots, b_{j+M+1}$ are not all identical, then we simply copy the bits $b_{j+1}, \ldots, b_{j+M}$ into $E_M(N)$ starting at bit $q(M+2)+3$ while the bits $q(M+2)+1$ and $q(M+2)+2$ contain the prefix 01. We then proceed by induction.

We note that the bits copied directly from $B(N)$ contribute on the average $(M+2)/M$ bits in $E_M(N)$ per bit in $B(N)$. On the other hand, a block of length L $(L > M)$ which is translated into $E_M(N)$ via the expansion of its length occupies $\lceil (\lfloor \log_2 L \rfloor + 1)/M \rceil \cdot (M+2)$ bits. The ratio of the latter to the number L is maximal when $L = M+1$. That maximal ratio is $(M+2)/(M+1)$. Thus, by selecting M large enough we can make the ratio arbitrarily close to one. The elementary operations in E_M are essentially the same as in E. This in fact completes the proof. □

It is easy to see that a stronger theorem can be proved if we are willing to represent lengths of blocks by the lengths of blocks in the binary representation of the lengths of the blocks, etc. This would enable us to strengthen claim (iii) and prove that there exist infinitely many numbers N such that $l(N) = O(\log\log\log N)$ or $l(N) = O(\log\log\log\log N)$, etc. This demonstrates that an algorithm like Khachiyan's looks very poor when we operate with encoding schemes which are arbitrarily as efficient as the binary encoding, while any genuinely-polynomial algorithm would remain polynomial in any such encoding scheme. We hope this will motivate further research in the direction of genuinely-polynomial algorithms. It is conceivable though that a genuinely-polynomial algorithm for linear programming exists only if P = NP. However, for the case of linear inequalities with at most two variables per inequality a genuinely-polynomial algorithm is known [3]. For the transportation problem the question is still open to the best of the author's knowledge, even though the dual has only two variables per inequality.

Appendix

We show as an example the encoding of the number 1033731 under the scheme E_5. First, the binary representation of the number is 11111100011000000011. The first block is of length 6 and hence we find the binary representation of the number 6, i.e., 110. With the prefix 11 the first block is encoded 1100110. The following 5 bits are not all identical so they are copied with the prefix 01 i.e., 0100011. The following 5 bits are identical and in fact the block is of length 7. So, with the prefix 00 it is encoded 0000111. The rest of the bits are encoded with the prefix 01 again, i.e., 0111. In summary, $E_5(1033731) =$ 1 1 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 1 1 1 0 1 1 1.

References

[1] S. Even and M. Rodeh, Efficient encoding of commas between strings, *Comm. ACM* **21** (1978) 315–317.

[2] L.G. Khachiyan, A polynomial algorithm in linear programming, *Soviet Math. Dokl.* **20** (1979) 191–194.

[3] N. Megiddo, Towards a genuinely polynomial algorithm for linear programming, Discussion Paper No. 493, The Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL 60201 (1981).