

ABSTRACT

In this paper we explore a new number system which uses a double base. The representation of the numbers has a very simple geometric interpretation, allowing potentially fast implementation of the basic arithmetic operations. The transformation of the integers into minimal form, however, leads to some problems associated with transcendental number theory, and we identify and open the discussion on these problems. An intriguing implementation vehicle, which has some of the properties associated with symbolic substitution in optical computing, is the use of Cellular Neural Networks (CNNs) to perform digit reduction. Brief details are presented on CNN implementation, and a system-level example is shown in order to justify the applicability of the proposed theory in digital signal processing.

Keywords: Number Systems, Computer Arithmetic, Number Theory, Diophantine Equations, Symbolic Substitution, Cellular Neural Networks, Digital Filtering

1. INTRODUCTION

The representation of numbers has seen intensive investigations from both a purely mathematical as well as from an applied point of view. The conventional classification of number systems is¹:

1. Weighted, where, for example every integer x can be represented in the form of eqn. (1), where α is a corresponding base and $g_i \in \{0, 1, \dots, \alpha - 1\}$:

$$x = \sum_{i=0}^N g_i \alpha^i \quad (1)$$

2. Non-weighted number systems, such as the residue number system², where digits are not assigned specific weights, and the ordering of the digits does not affect the number representation.

In considering a number system, three major questions arise:

1. Does there exist a possibility for every integer to be represented (completeness)?
2. Is the representation of numbers ambiguous or unambiguous (ambiguity)?
3. Does the representation provide advantage in implementing arithmetic operations on the represented numbers?

In some applications the computational complexity of the algorithm crucially depends upon the number of zeros of the input data in the corresponding number system. Some well-known examples are: the basic arithmetic operations, CORDIC algorithms, modular exponentiations. Number systems are often chosen to enable a reduction of the complexity of the arithmetic operations. The most popular are, perhaps, signed-digit number systems³, where, if b is the base, the set of allowable digits is $\{-(b-1), \dots, 0, (b-1)\}$. An analysis of the expected number of the zeros in the representation of arbitrarily integers in the signed-digit number systems is given in⁴. It has been shown by Garner⁵ that using the binary signed-digit number representation for long wordlengths requires, on average, 33% fewer adders to perform multiplication than binary. If $b=2$ we have on average $\frac{1}{3} \lg_2 N$ nonzero digits to present the integer number N . It is less well-known that in the Zeckendorf representation (a representation of a number as a sum of generalized Fibonacci numbers) one needs on average $0.398 \lg_2 N$ ones⁷

to present N . The last fact was used to improve some numerical algorithms^{7,8}.

Summarizing, we need on average the following number of nonzero digits to present N :

1. $0.5 \cdot \lg_2 N$ in the binary number system (nonzero digit: +1)
2. $0.398 \cdot \lg_2 N$ in the Zeckendorf representation (nonzero digit: +1)
3. $0.333 \cdot \lg_2 N$ in the binary signed-digit number system (nonzero digits: +1 and -1).

In all of the above mentioned number systems we need on average $O(\lg N)$ nonzero digits to write the integer N .

A very interesting question arises: does there exist a number system, allowing only digits 0,1 (and perhaps -1), requiring $o(\lg N)$ nonzero digits? In this paper we will provide a positive answer to the question.

We will deal with a representation of numbers using bases 2 and 3, that is, we consider the following form⁶:

$$x = \sum_{i,j} d_{i,j} 2^i 3^j \quad (2)$$

where $d_{i,j} \in \{0, 1\}$. We will refer to this as a *double-based number system (DBNS)* and the representation as a *double-based number representation (DBNR)*. Clearly the binary and ternary number systems are special cases of the above representation. A recently published representation⁹ has some similarities with this scheme. The Gaussian integers are encoded as $z = \sum \sum \sum d_{i,j,k} T^i X^j Y^k$ with $d_{i,j,k} \in \{0, 1, -1\}$. The interdeterminates X and Y are mapped from selected powers of 2 and the interdeterminate T maps $\sqrt{-1}$.

In this paper we will deal with a canonic, or minimal form for the representation as a sum of canonic numbers of the form $2^k 3^l$, that is, using the minimal number of ones; we will analyze the expected value of these numbers.

This representation strategy has an unusually simple geometric interpretation, which turns out to be suitable for optical computing techniques^{10,11}, or via 2-D cellular automata. Arithmetic operations in this number system unfortunately do not guarantee that the results are obtained in the minimal, or canonic form. The problem of conversion from non-canonic form leads to interesting problems in transcendental number theory. The new number system, however, appears to provide very fast carry-free addition and is also suitable for multiplication. We illustrate the ideas with an applications to computing finite impulse response filter inner products and modular exponentiation algorithm.

We finally introduce some open questions, where the proof of some conjectures may lead to important theoretical and practical results.

2. BASIC DEFINITIONS

Following from de-Weger¹², we use the following definitions:

Definition 1: An integer x , is called s -integer if all of its prime divisors are among the first s primes.

Definition 2: Let $G_{2,3}(x)$ be the set of 2-integers, smaller than or equal to x .

The asymptotic behavior of the cardinality of $G_{2,3}(x)$ can be easily estimated from the inequality $2^k 3^m \leq x$, that is:

$$k \cdot \ln 2 + m \cdot \ln 3 \leq \ln x \quad (3)$$

Hence the cardinality of $G_{2,3}(x)$ is equal to the number of nonnegative integer co-ordinates, satisfying (3). This number is approximately $\left\lceil \frac{\lg^2 x}{2 \cdot \lg 3} \right\rceil \approx \frac{1}{3.17} \lg x$. A more accurate estimation can be obtained if one uses the following theorem, proved by Hardy:

$$\text{card}(G_{2,3}(x)) = \frac{(\ln x)^2}{2 \cdot \ln 2 \cdot \ln 3} + \frac{1}{2} \cdot \left(\frac{1}{\ln 2} + \frac{1}{\ln 3} \right) \cdot \ln x + o\left(\frac{\ln x}{\ln \ln x} \right) \quad (4)$$

For our purposes the estimation of the cardinality of $G_{2,3}(x)$ as $O((\lg x)^2)$ will be good enough.

Obviously every integer x has several different representations as a sum of 2-integers. Let us consider a table with $\lceil \log_3 x \rceil + 1$ columns and $\lceil \log_2 x \rceil + 1$ rows, so that in every cell (i,j) the number $2^i 3^j$ is written (this is shown in Table 1):

	3^0	3^1	3^2	...	3^{k-1}	3^k	...
2^0	1	3	9	...	3^{k-1}	3^k	...
2^1	2	6	18	...	$2 \cdot 3^{k-1}$	$2 \cdot 3^k$...
2^2	4	12	36	...	$4 \cdot 3^{k-1}$	$4 \cdot 3^k$...
.
.
2^{m-1}	2^{m-1}	$3 \cdot 2^{m-1}$	$9 \cdot 2^{m-1}$...	$2^{m-1} \cdot 3^{k-1}$	$2^{m-1} \cdot 3^k$...
2^m	2^m	$3 \cdot 2^m$	$9 \cdot 2^m$...	$2^m \cdot 3^{k-1}$	$2^m \cdot 3^k$...
.

Table 1: DBNS Table

An arbitrary integer smaller than or equal to $2^m 3^k$ can be represented as a sum of numbers, which appears in the first $k + \lceil \log_2 3^m \rceil + 1$ rows and $m + \lceil \log_3 2^k \rceil + 1$ columns. We will refer the image obtained in this way as a **DBNS-map**.

Definition 3: The representation of an arbitrary integer in the DBNS-map using a minimal number of ones is called the minimal double-based number representation (MDBNR).

Definition 4: We will call a cell (i,j) active if it takes part in the corresponding DBNS-map.

Thus the *MBDNR* is a *DBNS* - map with the minimal number of active cells.

3. MINIMAL DOUBLE-BASED NUMBER REPRESENTATION

An interesting question is how many ones contains an arbitrary integer in its MDBNR. It is easy to check, for example, that 23 is the smallest nonnegative integer requiring 3 ones. The smallest integer, requiring 4 ones in its MDBNR is 431, which is a considerable distance from 23. The smallest integer, requiring 5 ones, is 18431 (it has, however, 219 different representations as a sum of

five 2-integers). It is not very easy to check that the smallest integer, requiring 6 ones is 3 448 733!

From these numerical results it is clear that the DBNS-map allows extremely sparse representation of the integers, and its sparsity is a good measure for potential implementation of many algorithms. Table 2 shows the MDBNR for two randomly selected integers.

	1	2	4	8	16
1					
3					
9					
27					

$x = 79$

	1	2	4	8	16	32
1						
3						
9						
27						

$x = 107$

Table 2: The MDBNR of the numbers 79 and 107

We shall state, without proving, the following theorem:

Theorem 1: The average number of ones, needed to represent an integer number n of the form (3) is $k = \left(\frac{\log n}{\log \log n}\right) + o\left(\frac{\log n}{\log \log n}\right)$.

It is interesting to search for an algorithm which guarantees the minimality of the presentation.

A natural technique for finding the *MBDNR* would appear to be the greedy algorithm shown below. Unfortunately, this algorithm does not ensure the minimality of the representation. The smallest x , when the greedy algorithm fails is 41. It is intuitively clear, however, that the algorithm provides close solutions to the *MBDNR*, moreover it is very easy to implement, while the problem of finding the *MBDNR* would seem to be *NP*-complete.

The Greedy Algorithm

Input: positive integer x ;

Output: 2-integers a_i , such that $\sum a_i = x$.

procedure greedy(x)

```
{
  while ( $x > 0$ ) then do
  begin
    find the largest 2-integer  $w$ , smaller than or equal to  $x$ ;
    write( $w$ );
     $x := x - w$ ;
    greedy( $x$ )
  end
}
```

Theorem 2: The greedy algorithm terminates on average after $O\left(\frac{\log x}{\log \log x}\right)$ steps.

The proof of this theorem can be found in¹³.

From a practical point of view it is interesting to know how many times the largest 2-integer, smaller than x , does not belong to any

of the *MDBNR* s of x . Empirical observations show us that in about 80% of the cases the largest 2-integer, smaller than x , occurs in at least one of the *MDBNR*s of x . These observations allow us to estimate that the greedy algorithm returns a *MDBNR* with probability $0.8 \frac{\log x}{\log \log x}$, which tends to zero very slowly. The greedy algorithm, however, allows us to find a representation requiring on average $O\left(\frac{\log x}{\log \log x}\right)$ 2-integers. It really does not matter that this representation is most likely not a *MDBNR*, it can still be used as a sufficiently sparse representation of x .

4. ANALYSIS OF THE CANONIC FORM TRANSFORMATION

The mechanism of finding an *MDBNR* (or close to it) plays a crucial role in performing basic arithmetic operations. We shall consider this process in more details.

If we once again consider the geometrical interpretation of the numbers in the DBNR, we find that the application of simple identities allows the decrement of special combinations of active cells. For example, see Table 3, the identity

$$2^i 3^j + 2^{i+1} 3^j = 2^i 3^{j+1}$$

demonstrates that there are no consecutive active cells lying in one column. Table 4 demonstrates another reduction based on $2^i 3^j + 2^i 3^{j+1} = 2^{i+2} 3^j$.

	...	3^j	3^{j+1}	...
...				
2^i				
2^{i+1}				
...				

 \Rightarrow

	...	3^j	3^{j+1}	...
...				
2^i				
2^{i+1}				
...				

Table 3: Reduction of active Cells Based on $2^i 3^j + 2^{i+1} 3^j = 2^i 3^{j+1}$

	...	3^j	3^{j+1}	...
...				
2^i				
2^{i+1}				
2^{i+2}				

 \Rightarrow

	...	3^j	3^{j+1}	...
...				
2^i				
2^{i+1}				
2^{i+2}				

Table 4: Reduction of active Cells Based on $2^i 3^j + 2^i 3^{j+1} = 2^{i+2} 3^j$

We can generalize the problem of decreasing the number of active cells using a purely exponential Diophantine equation:

$$2^{i_1} 3^{j_1} + 2^{i_2} 3^{j_2} + \dots + 2^{i_k} 3^{j_k} = 2^{m_1} 3^{n_1} + 2^{m_2} 3^{n_2} + \dots + 2^{m_l} 3^{n_l} \tag{5}$$

where $l < k$. The problem of solving Diophantine equations such as eqn. (5) has been a subject of investigation over the last two decades¹², although some interesting results were obtained in the 30's and 40's.

We shall consider some special cases for k and l .

4.1. $k=2, l=1$

Theorem 4¹²: The Diophantine equation $x + y = z$ where $GCD(x,y,z) = 1$ and x,y and z are 6-integers (that is x,y,z have the form $2^{x_1} 3^{x_2} 5^{x_3} 7^{x_4} 11^{x_5} 13^{x_6}$, with $x_i \geq 0, i = 1,2,3,4,5,6$) has exactly 545 solutions. Only three of them satisfy the condition $x_3 = x_4 = x_5 = x_6 = 0$ and the corresponding solutions for x,y and z are (1,2,3), (1,3,4) and (1,8,9)

4.2. $k=3, l=1$

The total number of solutions of the equation $x + y + z = t$, $GCD(x,y,z,t) = 1$, in 2-integers is 27 as tabulated in Table 5:

1,2,3,6	1,2,6,9	1,2,9,12	1,2,24,27	1,3,4,8	1,3,8,12
1,3,12,16	1,3,32,36	1,4,27,32	1,6,9,16	1,8,9,18	1,8,18,27
1,8,27,36	1,8,72,81	1,9,54,64	1,12,243,256	1,16,64,81	1,27,36,64
1,32,48,81	1,216,512,729	2,3,4,9	2,3,27,32	2,9,16,27	3,4,9,16
3,8,16,27	8,9,64,81				

Table 5: The Solutions Of the Equation $x + y + z = t$ in 2-integers

The subfield of number theory that studies equations of the form of eqn. (5) is called transcendental number theory and a powerful technique, described by Baker¹⁴, allows one to conclude that eqn. (5) has only a finite number of solutions. Existing methods for bounding the upper limits, however, give very large upper bounds. Therefore in searching for an *MDBNR* we are forced to use methods that do not guarantee exact *minimality* but rather near minimality, with the advantage that they can be realized algorithmically.

5. ADDITION AND MULTIPLICATION USING THE DBNS

5.1. Addition

Let x and y be two integer numbers, represented in *MDBNR*. The additive procedure follows by overlying the corresponding *DBNS* -maps for x and y and translating the result into minimal form. Note that if x and y contain the element $2^i 3^j$ in their *MDBNR*, then the element $2^{i+1} 3^j$ does not exist in their *DBNS* -maps, and therefore the addition can be done in a carry-free manner; the result obtained, however is not in minimal form. The approach described in the previous section allows us to find the *MDBNR* by replacing k active cells with l , where $k > l$. This is referred to as symbolic substitution, and widely used in the field of optical computing¹¹.

Let $I_x(i,j)$ and $I_y(i,j)$ be the *DBNS* -maps of the integers x and y , represented in the *MDBNR*. The image $I_z(i,j)$ of the *DBNS* -map of the number $z = x + y$ can be obtained by the following rules:

$$I_z(i+1,j) = I_x(i,j) \text{ AND } I_y(i,j) \quad \text{Rule (1)}$$

$$I_z(i,j) = I_x(i,j) \text{ XOR } I_y(i,j) \quad \text{Rule (2)}$$

Note, using *MDBNR*, if $I_x(i,j) = I_y(i,j) = 1$, then $I_x(i+1,j) = I_y(i+1,j) = 0$, and therefore addition can be accomplished using the symbolic substitution techniques. Once the image, I_z , is obtained, it has to be transformed into the canonic form. This can be done if one knows all solutions to eqn. (5) (their number is finite, if k and l are fixed), then reduce the active cells as much as possible. It is not clear at this juncture how practical this may be. To ensure the implementation of Rule (1) and Rule (2) it is sufficient to use Rule (3).

$$I_z(i+1,j) = I_z(i,j) \text{ AND } I_z(i+1,j) \quad \text{Rule (3)}$$

If a given *DBNS* map does not contain consecutive active cells lying on one row, that is if $I_z(i,j) \text{ AND } I_z(i+1,j) = \mathbf{0}$ for all i,j , then such a representation can be used for carry-free addition. It is interesting to analyze how many times Rule (3) has to be applied in order to guarantee obtaining a representation ready for carry-free addition. To show this, we introduce.

Theorem 3: The total number of the applications of Rule (3) to a given *DBNS*-map, necessary to obtain a representation, satisfying the condition $I_z(i,j) \text{ AND } I_z(i+1,j) = 0$ for all i,j is at most $\frac{n}{\lg_2 3} \approx 0.630925n$, assuming that z is n -bit binary integer.

Proof: The application of Rule (3) can be applied in parallel to a given *DBNS*-map. Once applied, it guarantees the removing of the active consecutive cells, lying in the last row, the next application will remove the active consecutive cell, lying in the previous row (if such cells exist) and so on. Therefore, in the worst case we require a number of applications equal to the number of rows in the *DBNS* table necessary to represent the n -bit binary integer; this number is $\frac{n}{\lg_2 3}$.

q

Comparing the last result to the worst case analysis of the binary addition, when one needs n carries to reach the final result of summation of two n -bit integers, in *DBNS*, thanks to the second dimension in the number representation, we need in the worst case about 37% less carries.

5.2. Multiplication

We expect to develop a similar procedure to that used for addition.

Once again we shall deal with *DBNS* -maps. Let x and y are integer numbers, presented by *DBNS* -maps in *MDBNR*. Their product is a number whose *MDBNR* is composed by the elements $z^i 2^j 3^k = 2^{i_x + i_y} 3^{j_x + j_y}$. We will also require the transformation back into canonic form. It is clear that the multiplication of a given two *DBNS* -presented integers with a 2-integer, namely, $2^i 3^j$, is equivalent to 2-D shift in the *DBNS* -map of the corresponding integer. Therefore the multiplication can be done as in the conventional binary number system as a sequence of shifts and adds.

6. APPLICATION OF CELLULAR NEURAL NETWORKS IN DBNS ARITHMETIC

As we have shown before the image $\mathbf{I}_z(i,j)$ of the *DBNS*-map of the number $z=x+y$ can be obtained by Rule (1) and Rule (2). Therefore, addition in *DBNS* can be performed by evaluating two simple Boolean functions between the corresponding cells of two images \mathbf{I}_x and \mathbf{I}_y . In this section we describe a method for implementing these functions using cellular neural networks. This method is a generalization of the method proposed by Galias¹⁵, who describes a method for designing cellular neural networks to implement arbitrary Boolean functions defined on the r -neighborhood.

6.1. DBNS addition using cellular neural networks

Using the well known DeMorgan rule, Rule (1) can be rewritten as Rule (4):

$$\overline{\mathbf{I}_z(i, j+1)} = \overline{\mathbf{I}_x(i, j)} \vee \overline{\mathbf{I}_y(i, j)} \quad \text{Rule (4)}$$

This Boolean function can be implemented as follows. Let us consider a 1-neighborhood CNN with the following templates:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{I} = 1$$

Using Galias' method by applying $\mathbf{I}_x(i,j)$ as inputs to the cells and setting the initial values of the state voltage of the cells equal to '-1', the output of the cells, after the transient time (the time needed to implement the simplest Boolean function 'a product with just one term'; for future reference let us call it T°), $\mathbf{y}_{ij}(T^\circ)$ will be $\overline{I_x(i, j-1)}$. Now, by just replacing the old inputs of the cells $\mathbf{I}_x(i,j)$ by new inputs $\mathbf{I}_y(i,j)$ the final output values of the cells $\mathbf{y}_{ij}(2T^\circ)$ will be $\overline{I_x(i, j-1)} \vee \overline{I_y(i, j-1)}$, because by successive applications of these two inputs, the final output will be '1' if either $\overline{I_x(i, j-1)}$ or $\overline{I_y(i, j-1)}$ or both of them are '1'. This approach can be used for implementing any Boolean function between corresponding cells of two images. Rule (2) can be realized in a similar way. The entire addition using CNNs as described above can be done in $9T^\circ$.

6.2. Designing CNNs for applying the reduction rules

We have already discussed reduction rules and their applications in converting non-canonic representation of *DBNS* numbers to canonic form. In the following we explain the application of CNNs in this regard. Reduction rule implementation using CNNs is performed in three steps:

- Step 1: generating a new target image which has active cells resulting from a permissible reduction;
- Step 2: deleting the cells which participated in reducing the original image (whitening);
- Step 3: adding the two images obtained in Step 1 and Step 2 using *DBNS* addition rules.

Step 1:

We explain the design method using the simplest rule $2^i 3^j + 2^{i+1} 3^j = 2^i 3^{j+1}$. For carry-free addition, we can not allow the *DBNR* of the numbers to have two adjacent active cells in any column. It is clear that we require a 3×3 neighborhood around the target cell in order to apply this rule. The value of the target cell at an arbitrary position (i,j) will be '1' if we have active cells at positions $(i,j-1)$ and $(i+1,j+1)$. By implementing the Boolean function using Galias' method, this reduction rule can be performed using a 3×3 neighborhood CNN with the following specifications:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \mathbf{I} = 1$$

By applying $\mathbf{I}_z(i,j)$ to the inputs of the cells, and setting the initial values of the state voltage of the cells equal to '-1', as discussed previously, we generate the Boolean function, given by eqn. (6), in the output image $\mathbf{I}_{zn}(i,j)$, after transient time $3T^\circ$.

$$I_{zn}(i, j) = \overline{I_z(i-1, j-1)} \wedge I_z(i, j-1) \wedge I_z(i+1, j-1) \quad (6)$$

Step 2:

In this step we delete (whiten) active cells in the original image $\mathbf{I}_z(i,j)$ which participated in reductions. In this case we have to consider the Boolean function given by eqn. (7).

$$I_{du}(i, j) = \overline{I_z(i-1, j)} \wedge I_z(i, j) \wedge I_z(i+1, j) \quad (7)$$

This Boolean function can be implemented using a 3x3 neighborhood CNN with the following specifications:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \mathbf{I} = 1$$

This will generate the function given by eqn. (7) after transient time $3T^\circ$.

Step 2 is terminated by implementing the Boolean function of eqn. (8).

$$I_r(i, j) = I_z(i, j) \wedge \overline{I_d(i, j)} \quad (8)$$

Implementing eqn. (8), using simple CNNs with a 1-neighborhood, causes the active cells, participating in reductions, to be deleted from the original image. Step 2 is completed in time $10T^\circ$.

Step 3:

By adding the images obtained in Step 1 and Step 2 ($I_{zn}(i, j)$ and $I_{du}(i, j)$ respectively) using DBNS addition rules we obtain the desired image. The entire procedure for applying the reduction rules takes $22T^\circ$.

7. A SYSTEM EXAMPLE - DIGITAL FILTERING USING DBNS

The basic operation used in FIR digital filtering is linear convolution between the signal $g(k)$ and the filter coefficients $h(k)$. The linear convolution is defined as follows: $a(i) = \sum_{k=\max(0, i-N+1)}^{\min(N-1, i)} g(k) \cdot h(i-k)$ where $i = 0, 1, 2, \dots, 2N-2$. A well established technique to reduce the complexity of implementing the convolution is to choose the filter coefficients to contain only a small number of powers of two. This selection allows the multiplications to be replaced by adds and shifts (or only shifts). Using CSD arithmetic allows the choice of canonic forms of the coefficients, where the digits are -1, 0 and 1. There is usually a trade-off between the average number of non-zero terms in the representation and the length of the filter to meet required specifications.

The *DBNS* can also be used to compute linear convolutions, without general multiplication; to show this we introduce Theorem .

The *DBNS* can also be used to compute linear convolutions, without general multiplication; to show this we introduce Theorem .

Theorem 4: Let n and m be integers. The set $A_{n,m} = \{2^n 3^m\}$ is compact over the nonnegative reals, that is, in every interval $[\delta_1 \div \delta_2]$ ($\delta_1 \geq 0, \delta_2 > \delta_1$) at least one number of the form $2^n 3^m$ appears.

Proof: The proof follows from the well-known fact that the set of the numbers $n + m \cdot \omega$ ($n, m \in \mathbb{Z}, \omega - \text{irrational}$) is compact over the reals. In our case we have $\omega = \log 3$.

Based on our geometrical interpretation, if we extend the *DBNS* -map in both directions, then every nonnegative filter coefficient can be approximated with arbitrary small error using only one active cell. Now multiplication with the filter coefficients can be replaced via 2-D shifts.

However, we can do better! We can represent every active cell as a set of indices. Because the nonnegative filter coefficients are approximated with only one active cell, the multiplications are completely avoided and replaced by simple addition. The negative filter coefficients (if they exist) can not be approximated with arbitrary small error of the form $2^a 3^b$, a, b - integers. However, the impulse response $h(n)$ can be 'raised', so that the filter coefficients became positive. More precisely, $h(n)$ is replaced by $h_1(n) +$

q

$h_2(n)$, where $h_2(n)$ is a negative constant and $h_1(n)$ is positive for all n .

7.1. An Example

The example is based on the following specifications:

1. passband and stopband edge frequencies 0.021 and 0.07;
2. passband ripple and stopband attenuation requirements 0.2 and 60 dB respectively.

The ideal 60-tap filter has a passband ripple 0.20 dB and stopband attenuation 61.7 dB. The coefficients of the ideal filter are given in Table 6.

$h(0)=h(59)=0.003784$	$h(1)=h(58)=0.003417$	$h(2)=h(57)=0.003540$
$h(3)=h(56)=0.002686$	$h(4)=h(55)=0.000977$	$h(5)=h(54)=-0.001831$
$h(6)=h(53)=-0.008423$	$h(7)=h(52)=-0.017095$	$h(8)=h(51)=-0.008301$
$h(9)=h(50)=-0.029785$	$h(10)=h(49)=-0.082031$	$h(11)=h(48)=-0.072266$
$h(12)=h(47)=-0.063965$	$h(13)=h(46)=-0.060059$	$h(14)=h(45)=-0.060425$
$h(15)=h(44)=-0.068359$	$h(16)=h(43)=-0.021484$	$h(17)=h(42)=-0.015503$
$h(18)=h(41)=0.019287$	$h(19)=h(40)=0.064209$	$h(20)=h(39)=0.117676$
$h(21)=h(38)=0.179688$	$h(22)=h(37)=0.245117$	$h(23)=h(36)=0.314453$
$h(24)=h(35)=0.382813$	$h(25)=h(34)=0.445313$	$h(26)=h(33)=0.501709$
$h(27)=h(32)=0.546631$	$h(28)=h(31)=0.579825$	$h(29)=h(30)=0.594238$

Table 6: The coefficients of the ideal 60-tap filter

Because the coefficients $\{h(i)\}$ for $5 \leq i \leq 17$ and $42 \leq i \leq 54$ are negative and greater than -2^{-3} select $h_2(n) = -2^{-3}$ and now the coefficients $h_1(n)$ became positive. The approximation of the filter coefficients, such that $h_1(n)$ consists of 8,9 and 10-bit (including the sign bit) pairs (a, b) , are given in Table 7, Table 8 and Table 9. The pair (a,b) is such that $2^{a3^b}-0.125$ is the closest number to the corresponding coefficient.

$ha(0)=0.003703; (-127,77)$	$ha(1)=0.003434; (-41,24)$	$ha(2)=0.003434; (-41,24)$
$ha(3)=0.002632; (211,-135)$	$ha(4)=0.000914; (230,-147)$	$ha(5)=-0.001940; (100,-65)$
$ha(6)=-0.00843; (176,-133)$	$ha(7)=-0.017085; (122,-79)$	$ha(8)=-0.008189; (92,-60)$
$ha(9)=-0.029772; (-108,66)$	$ha(10)=-0.082009; (-209,109)$	$ha(11)=-0.072266; (-9,3)$
$ha(12)=-0.063917; (-134,82)$	$ha(13)=-0.060043; (23,-17)$	$ha(14)=-0.060514; (-210,130)$
$ha(15)=-0.068275; (213,-137)$	$ha(16)=-0.021384; (179,-115)$	$ha(17)=-0.015613; (103,-67)$
$ha(18)=0.019287; (40,-27)$	$ha(19)=0.064266; (145,-93)$	$ha(20)=0.117808; (120,-77)$
$ha(21)=0.179436; (-35,21)$	$ha(22)=0.244952; (16,-11)$	$ha(23)=0.314379; (-63,39)$
$ha(24)=0.382881; (-104,65)$	$ha(25)=0.445174; (-23,14)$	$ha(26)=0.501908; (-156,98)$
$ha(27)=0.546542; (234,-148)$	$ha(28)=0.578800; (-75,47)$	$ha(29)=0.593619; (139,-88)$

Table 7: Approximation of the filter coefficients as a pairs of 8-bit integers

ha(0)=0.003834; (360,-229)	ha(1)=0.003434; (-41,24)	ha(2)=0.003565; (444,-282)
ha(3)=0.002632; (211,-135)	ha(4)=0.000914; (230,-147)	ha(5)=-0.001808; (-469,294)
ha(6)=-0.00843; (176,-133)	ha(7)=-0.017085; (122,-79)	ha(8)=-0.008308; (-393,246)
ha(9)=-0.029772; (-108,66)	ha(10)=-0.082009; (-209,109)	ha(11)=-0.072266; (-9,3)
ha(12)=-0.063982; (435,-277)	ha(13)=-0.060043; (23,-17)	ha(14)=-0.060448; (275,-176)
ha(15)=-0.068333; (-272,169)	ha(16)=-0.021490; (-306,191)	ha(17)=-0.015496; (-466,292)
ha(18)=0.019287; (40,-27)	ha(19)=0.064266; (145,-93)	ha(20)=0.117560; (-365,229)
ha(21)=0.179748; (450,-285)	ha(22)=0.244952; (16,-11)	ha(23)=0.314379; (-63,39)
ha(24)=0.382881; (-104,65)	ha(25)=0.445174; (-23,14)	ha(26)=0.501908; (-156,98)
ha(27)=0.546542; (234,-148)	ha(28)=0.578050; (494,-312)	ha(29)=0.594386; (-430,271)

Table 8: Approximation of the filter coefficients as a pairs of 9-bit integers

ha(0)=0.003834; (360,-229)	ha(1)=0.003428; (1013,-641)	ha(2)=0.003565; (444,-282)
ha(3)=0.002638; (-843,530)	ha(4)=0.000920; (-824,518)	ha(5)=-0.001814; (585,-371)
ha(6)=-0.00843; (-878,552)	ha(7)=-0.017085; (122,-79)	ha(8)=-0.008308; (-393,246)
ha(9)=-0.029776; (946,-599)	ha(10)=-0.082011; (845,-536)	ha(11)=-0.072266; (-9,3)
ha(12)=-0.063979; (-619,388)	ha(13)=-0.060043; (23,-17)	ha(14)=-0.060445; (-779,489)
ha(15)=-0.068335; (782,-496)	ha(16)=-0.021490; (-306,191)	ha(17)=-0.015501; (588,-373)
ha(18)=0.019334; (525,-333)	ha(19)=0.064266; (145,-93)	ha(20)=0.117560; (-365,229)
ha(21)=0.179748; (450,-285)	ha(22)=0.244952; (16,-11)	ha(23)=0.314379; (-63,39)
ha(24)=0.382859; (950,-600)	ha(25)=0.445174; (-23,14)	ha(26)=0.501880; (898,-567)
ha(27)=0.546571; (-820,517)	ha(28)=0.578081; (-560,353)	ha(29)=0.594354; (624,-394)

Table 9: Approximation of the filter coefficients as a pairs of 10-bit integers

The resulting stopband attenuations are 57.8, 59.3 and 60.1 dB respectively.

8. CONCLUSIONS

In this paper we have opened the discussion on the use of double-based number systems as a technique for representing numbers that allows potentially low complexity arithmetic operations using a variety of implementation media. Our chosen implementation procedure for arithmetic operation is cellular neural networks simulations. For addition in the DBNS we simply have to ensure that there are no consecutive active cells lying in any row on the DBNS map. For multiplication, using 2-D shifts and additions, however, we seek representations with minimal active cells located large '2-D Hamming distances' from each other. Some final open questions are: the ability to express a function predicting the number of ones in an MDBNR in closed form; an estimate of worst case behavior of the greedy algorithm; a possible implementation of large integer modular multiplication with application in cryptography.

9. ACKNOWLEDGMENTS

This work is supported by the Natural Sciences and Engineering Research Council of Canada and the Micronet Network of Centres of Excellence.

10. REFERENCES

1. M.Schroeder, Number theory in science and communications, Springer-Verlag, 1986
2. Residue number system arithmetic: modern application in digital signal processing, ed. M.Soderstrand, W.K.Jenkins, G.A.Jullien and F.J.Taylor, IEEE Press, 1986
3. A.Avizienis, Signed digit number representation for fast parallel arithmetic, IRE Trans.Electro.Comup., vol.10, pp.389-400, 1961
4. T.Wheeler and S.Arno, Signed-digit number representation with minimal Hamming weight, IEEE Trans.Comp., vol.41, pp.1007-1010, 1993
5. H.Garner, Number systems and arithmetic, Advances in Computers, vol.6, pp.131-194, 1965
6. D.J. Mintz, 2,3 Sequence as a Binary Mixture, The Fibonacci Quarterly, vol. 19, pp. 351-360, 1981.
7. V.Dimitrov and B.Donevsky, Faster multiplication of medium-sized integers via the Zeckendorf representation, The Fibonacci Quarterly, vol.33, pp.74-77, 1995
8. V.Dimitrov and T.Cooklev, Two algorithms for modular exponentiations using nonstandard arithmetic, IEICE Trans.Fundam., vol.E78-A, pp.82-87, 1995
9. N.M.Wigley, G.A.Jullien and W.C.Miller, Super pipelined architectures for the polynomial Gauss machine, IEEE Int.Conf APCCAS, pp.525-527, 1992
10. E.Swartzlander, Digital optical computing, Applied Optics, vol.25, pp.3021-3032, 1986
11. K.-H.Brener, A.Huang and N.Streibl, Digital optical computing with symbolic substitution, Applied Optics, vol.25, pp.3054-3060, 1986
12. B.M.M.de-Weger, Algorithms for Diophantine equations, CWI Tracts-Amsterdam, vol.65, 1989
13. V.Dimitrov, G.A.Jullien and W.C.Miller, An improved hybrid algorithm for modular exponentiation', submitted to Electr.Lett.
14. A.Baker, The theory of linear forms in logarithms, in Transcendental Theory-Advances and Applications, A.Baker(ed.), Academic Press, pp.1-27, 1989
15. Z.Galias, Designing cellular neural networks for the evaluation of local Boolean functions, IEEE Trans. on Circuits and Systems-II, vol.40, pp.219-223, 1993