Integer Representations towards Efficient Counting in the Bit Probe Model

Gerth Stølting Brodal[1], Mark Greve[1], Vineet Pandey[2], and S. Srinivasa Rao[3]

[1] MADALGO*, Department of Computer Science, Aarhus University,
IT Parken, Åbogade 34, DK-8200 Århus N, Denmark.
E-mail: gerth@cs.au.dk, mgreve@cs.au.dk
[2] Computer Science & Information Systems, BITS Pilani, 333031, India.
E-mail: vineetp13@gmail.com
[3] School of Computer Science and Engineering, Seoul National University,
Republic of Korea. E-mail: ssrao@cse.snu.ac.kr

Abstract. We consider the problem of representing numbers in close to optimal space and supporting increment, decrement, addition and subtraction operations efficiently. We study the problem in the bit probe model and analyse the number of bits read and written to perform the operations, both in the worst-case and in the average-case. A counter is *space-optimal* if it represents any number in the range $[0, \ldots, 2^n - 1]$ using exactly n bits. We provide a *space-optimal counter* which supports increment and decrement operations by reading at most $n - 1$ bits and writing at most 3 bits in the worst-case. To the best of our knowledge, this is the first such representation which supports these operations by always reading strictly less than n bits. For *redundant counters* where we only need to represent numbers in the range $[0, \ldots, L]$ for some integer $L < 2^n - 1$ using n bits, we define the efficiency of the counter as the ratio between $L + 1$ and 2^n. We present various representations that achieve different trade-offs between the read and write complexities and the efficiency. We also give another representation of integers that uses $n + O(\log n)$ bits to represent integers in the range $[0, \ldots, 2^n - 1]$ that supports efficient addition and subtraction operations, improving the space complexity of an earlier representation by Munro and Rahman [Algorithmica, 2010].

Keywords: Data structure. Gray code. Bit probe model. Binary counter. Integer representation.

1 Introduction

We propose data structures for integer representation which can perform increment, decrement, addition and subtraction with varying trade-offs between the number of bits read or written and the space needed to represent the number.

* Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

We study the problem in the bit probe model of computation where the complexity measure includes only the bitwise accesses to the data structure and not the resulting computations.

We define a code of dimension n as any cyclic sequence of 2^n distinct binary vectors. For a code of dimension n, we define the operation increment (decrement) as moving the code to its next (previous) code in the cycle. We define a function Val that maps bit sequences to integers, which is used in describing our algorithms. We use B_R and B_W to denote the number of bits read and written respectively. The average number of bits read (written) is computed by summing the number of bits read (written) to perform the operations for each code, and dividing this by the number of different codes. Throughout the paper, $\log n$ denotes $\lceil \log_2 n \rceil$, $\log^{(0)} n = n$ and $\log^{(c)}(n) = \log^{(c-1)}(\log n)$ for $c > 0$.

Previous work. The Standard Binary Code (SBC) uses n bits to represent an integer in the range $[0, \ldots, 2^n - 1]$ where $b_{n-1}b_{n-2}\ldots b_0$ represents the value $\sum_{i=0}^{n-1} b_i 2^i$. An increment or decrement operation using SBC requires n bits to be read and written in the worst-case but the amortized time per operation is constant. A Gray code is any code in which successive binary vectors in the sequence differ in exactly one component. The Binary Reflected Gray Code (BRGC) [3] requires n bits to be read for each increment operation but only 1 bit to write. Bose et al. [1] have developed a different Gray Code called Recursive Partition Gray Code (RPGC) which requires on an average $O(\log n)$ reads for increment operations. The previous results are summarized in Table 1. For the Gray codes BRGC and RPGC, we define $\mathrm{Val}(X)$ as the number of times one needs to increment the code $0 \ldots 0$ to obtain X. The dimension d of a counter refers to the total number of bits used to represent a number and *space-efficiency* refers to the ratio of number of numbers represented out of all possible bit strings generated (2^d) given the dimension d. Space-efficiency equal to one implies that all possible strings are generated and the counter is *space-optimal*. There could be more than one representation for a given number when efficiency is less than one and such counters are called *redundant counters*.

Our results. For space-optimal counters, we introduce the notion of an (n, r, w)-*counter* which is a representation of numbers of dimension n where increment and decrement operations can be performed by reading r bits and writing w bits in the worst-case. We obtain a *(4,3,2)-counter* by exhaustive search and use it to construct an $(n, n-1, 3)$-*counter* which performs an increment or decrement operation by reading at most $n-1$ bits whereas all known results for space-optimal counters read n bits in the worst-case. The codes BRGC and RPGC are examples of $(n, n, 1)$-*counters*. Fredman has conjectured that for Gray codes of dimension n, $B_R = n$ [1, 2]. If this conjecture is true, this would imply that if there exists a code with the property that all increments can be made by reading less than n bits, then it would need to write at least 2 bits in the worst-case.

Space (d)	Space efficiency	Bits read (B_R)		Bits written (B_W)	Inc. & Dec.	Ref.
		Average-case	Worst-case	Worst-case		
n	1	$2 - 2^{1-n}$	n	n	Y	Binary
		n		1	Y	[3]
		$6 \log n$		1	Y	[1]
		$O(\log^{(2c-1)} n)$		c	N	[1]
$n+1$	1/2	$O(1)$	$\log n + 4$	4	Y	[4]
$n + O(t \log n)$	$1 - O(n^{-t})$	$O(\log^{(2c)} n)$	$O(t \log n)$	$2c+1$	N	[1]

Table 1: Summary of previous results

For non-space-optimal counters, the read complexity has been shown to be $\Theta(\log n)$ for a space-efficiency of 1/2 [1, 4]. The best known result so far [1] describes a counter with a space-efficiency of $1 - O(n^{-t})$ to increment a value by reading $O(t \log n)$ bits and writing 3 bits for $t > 0$. Our results shown in Table 2 show that we can reduce the number of bits written to 2 using a representation with space-efficiency $1 - O(2^{-t})$ by reading $\log n + t + 2$ bits where $t \in \mathbb{Z}^+$. By choosing $t = t' \log n$, we can achieve a space-efficiency of $1 - O(n^{-t'})$ by reading $O(t' \log n)$ bits and writing 2 bits. The question that remains open is if redundant counters efficiently allow a representation with 1 write but less than n reads.

For redundant counters with efficiency 1/2, the best known results were $\log n + 4$ bit reads and 4 bit writes [4]. We reduce the number of bits read and written to $\log n + 3$ and 3. Using the one bit read-write trade-off, we can further reduce the number of bits written to 2 by reading $\log n + 4$ bits.

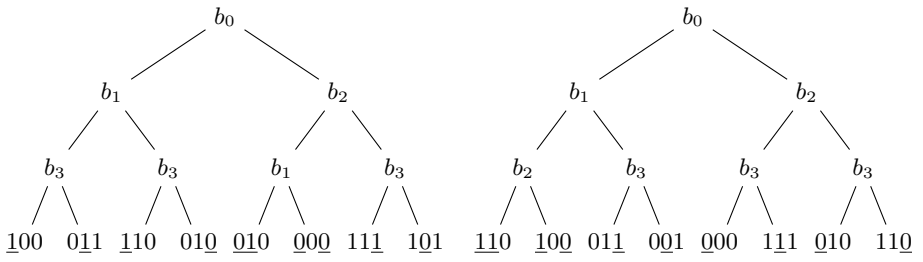
Space (d)	Space efficiency	Average-case		Worst-case		Inc. & Dec.	Ref.
		B_R	B_W	B_R	B_W		
4	1	3	1.25	3	2	Y	Th. 1
n		$6 \log(n-4) + O(2^{-n})$	$1 + O(2^{-n})$	$n-1$	3		Th. 2
$n+1$	1/2	$O(\log \log n)$	$1 + O(n^{-1})$	$\log n + 2$	3	N	Th. 3
				$\log n + 3$	2		Th. 4
		$O(\log n)$		$\log n + 3$	3	Y	Th. 6
				$\log n + 4$	2		
n	$1 - \frac{1}{2^{t-1}}$	$O(\log \log n)$	$1 + O(n^{-1})$	$\log n + t + 1$	3	N	Th. 5
				$\log n + t + 2$	2		
		$O(\log n)$		$\log n + t + 2$	3	Y	Cor. 1
				$\log n + t + 3$	2		

Table 2: Summary of our results

2 Space-optimal counters with increment and decrement

In this section, we describe space-optimal counters which are constructed using a *(4,3,2)-counter* where X denotes the number to be incremented.

(4,3,2)-counter. Fig. 1 shows our *(4,3,2)-counter* obtained through brute force search which represents numbers from $0 \ldots 15$. Assuming the number is of the



Increment tree Decrement tree

0000(0) $\xrightarrow{1}$ 0001(1) $\xrightarrow{2}$ 0100(2) $\xrightarrow{1}$ 0101(3) $\xrightarrow{1}$ 1101(4) $\xrightarrow{1}$ 1001(5) $\xrightarrow{2}$
1100(6) $\xrightarrow{1}$ 1110(7) $\xrightarrow{1}$ 0110(8) $\xrightarrow{1}$ 0111(9) $\xrightarrow{1}$ 1111(10) $\xrightarrow{1}$ 1011(11) $\xrightarrow{2}$
1000(12) $\xrightarrow{1}$ 1010(13) $\xrightarrow{1}$ 0010(14) $\xrightarrow{1}$ 0011(15) $\xrightarrow{2}$ 0000(0)

Fig. 1: Sequence generated by the *(4,3,2)-counter* and increment and decrement trees

form $b_3b_2b_1b_0$, the corresponding increment and decrement trees for the *(4,3,2)-counter* are shown in Fig. 1. For any internal node corresponding to bit b_t, the left edge corresponds to $b_t = 0$ and the right edge corresponds to $b_t = 1$. The leaf nodes contain information about the new values for the bits read and the modified bits are shown underlined in the tree and the text.

As an example, for the fifth leaf node from the left in the increment tree, old $b_0b_2b_1 = 100$ and new $b_0b_2b_1 = \underline{01}0$. To increment 9, for example, we take its representation 0111 and go through the path $b_0b_2b_3 = 110$ in the increment tree to reach the seventh leaf node; so the new values are $b_0b_2b_3 = 11\underline{1}$ and the new number is $\underline{1}111$ which represents 10 (ten). To decrement 9, we go through the path $b_0b_2b_3 = 110$ in the decrement tree to reach the seventh leaf node; so the new values are $b_0b_2b_3 = \underline{0}10$ and the number is $011\underline{0}$ which represents 8.

Theorem 1. *There exists a representation of integers of dimension* 4 *with efficiency* 1 *that supports increment and decrement operations with* $B_R = 3$ *and* $B_W = 2$ *in the worst-case. On average, an increment/decrement requires* $B_R = 3$ *and* $B_W = 1.25$.

2.1 Constructing $(n, n-1, 3)$-*counter* using *(4,3,2)-counter*

We can now construct an n-bit space-optimal counter for $n \geq 4$ by dividing the code for a number X into two sections $X_{(4,3,2)}$ and X_{G} of length 4 and $n-4$ respectively where $X_{(4,3,2)}$ uses the above-mentioned *(4,3,2)-counter* representation and X_{G} uses the Gray code [3], that is a $(n-4, n-4, 1)$-*counter*. To increment X, we first increment X_{G} and then check if it represents 0 (which is possible since we read all bits of X_{G}). If X_{G} is 0, then we increment $X_{(4,3,2)}$. In the worst-case, this requires $n-4$ reads and 1 write to increment X_{G} and then

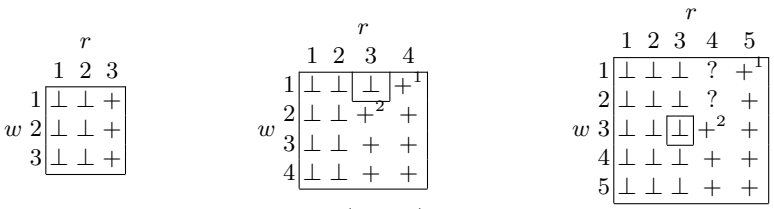


Fig. 2: Exhaustive search results for (n, r, w)-*counter* for $n = 3$, 4 and 5 respectively

3 reads and 2 writes to increment $X_{(4,3,2)}$, providing us with $n - 1$ reads and 3 writes overall.

X_{G} is represented using RPGC where incrementing or decrementing a code of dimension n requires $6 \log n$ average number of reads (although [1, Theorem 2] considers only generating the next code, i.e., increment operation, one can verify that the same analysis holds for the decrement operations as well). The worst-case and hence the average number of writes to increment or decrement a number using RPGC is 1. Since the average number of reads and writes for $X_{(4,3,2)}$ are 3 and 1.25 respectively, and we increment/decrement $X_{(4,3,2)}$ only in one out of every 2^{n-4} codes, the average number of reads and writes are $6 \log(n-4) + 3/2^{n-4}$ and $1 + 1.25/2^{n-4}$ respectively.

Theorem 2. *There exists a representation of integers of dimension $n \geq 4$ with efficiency 1 that supports increment and decrement operations with $B_R = n - 1$ and $B_W = 3$ in the worst-case. On average, an increment/decrement requires $B_R = 6 \log(n-4) + O(2^{-n})$ and $B_W = 1 + O(2^{-n})$.*

To the best of our knowledge, this is the first space-optimal counter with B_R strictly less than n.

Exhaustive search results We used exhaustive search to find (n, r, w)-*counters* for small values of n. The results are shown in Fig. 2 for $n = 3$, 4 and 5 respectively. For a combination of n, r and w, a '$\perp$' shows that no counter exists and a '+' refers to its existence. A superscript of 1 shows that this is a Gray code while 2 refers to Theorem 2. A '?' shows that the existence of counters remains unknown for the corresponding (n, r, w) value. An enclosed value shows that no counters were found by our brute-force search.

3 Redundant counters with increment

To reduce the number of bits read exponentially, counters with space-efficiency less than one have been considered [1, 4]. In this section, we discuss redundant counters which show better results and trade-offs for bits read and written and use these in Section 5 to obtain representations that support addition and subtraction efficiently.

3.1 Counters with one bit redundancy

To represent numbers from $0 \ldots 2^n - 1$, we select $n + 1$ bits. A number X represented by $x_n x_{n-1} \ldots x_1 x_0$ consists of a *carry bit* $S = x_0$, a lower block X_L

Previous			New	
ℓ	S	x_p	S	x_p
$= \ell_{\max}$	0	x	$\underline{1}$	x
$< \ell_{\max}$	0	x	0	x
$< \ell_{\max}$	1	0	$\underline{0}$	$\underline{1}$
$< \ell_{\max}$	1	1	1	$\underline{0}$

Table 3: Transition Table for the increment step where $\ell = \mathrm{Val}(X_L)$ and $p = \log n + \ell$. Underlines show the changed bits and x represents '*don't care*' condition

of the $\log n$ bits $x_{\log n} \dots x_1$ and the upper block X_H of the last $n - \log n$ bits. $p = \log n + \ell$ is a location in X_H where ℓ refers to the value represented by X_L. This is used to perform a delayed addition of the carry as explained below. We use Gray codes for representing the numbers in X_L so that increment writes only one bit. The block X_H is represented using SBC. The value of X is given by $(\ell + (\mathrm{Val}(X_H) + 2^\ell \cdot S) \cdot 2^{|X_L|}) \bmod 2^n$.

We determine the number of bits read and written in the worst-case by finding the maximum values of B_R and B_W respectively. The increment step is summarised in Transition Table 3.

Increment: X_L and S are read at every step, therefore B_R is at least $\log n + 1$. $S = 1$ implies that the carry needs to be propagated and we will read one bit from X_H, whereas $S = 0$ implies no carry propagation and we do not need to access X_H. If $\ell > n - \log n$, we reset S to 0. The different cases for increment are described below:

Case 1. $S = 0$ and X_L contains its *largest* value ($100 \dots 0$ in Gray code): This implies that a new incremental increment of X_H should be initiated. Increment X_L and set the carry bit S to 1. ($B_R = \log n + 1$, $B_W = 2$)

Case 2. $S = 0$ and X_L is any other value: Increment X_L. ($B_R = \log n + 1$, $B_W = 1$)

Case 3. $S = 1$ and $x_p = 1$: Propagation of carry. Change x_p to 0. Increment X_L.($B_R = \log n + 2$, $B_W = 2$)

Case 4. $S = 1$ and $x_p = 0$: Final bit flip in X_H. Change x_p to 1, S to 0 and increment X_L. ($B_R = \log n + 2$, $B_W = 3$).

The average number of reads to increment X_L is $O(\log \log n)$. The bit S is read at every step and it is set to 1 on the average 2 out of every n steps. When $S = 1$, we also need to read $O(\log n)$ bits to find $\mathrm{Val}(X_L)$. Thus the average number of bits read is $O(\log \log n)$. The average number of writes can be shown to be $1 + O(n^{-1})$. Hence we have the following theorem.

Theorem 3. *There exists a representation of integers of dimension $n + 1$ with efficiency $1/2$ that supports increment operations with $B_R = \log n + 2$ and $B_W = 3$. On average, an increment requires $B_R = O(\log \log n)$ and $B_W = 1 + O(n^{-1})$.*

3.2 One bit read-write trade-off

We show how to modify the representations of the previous section (Theorem 3) to reduce B_W from 3 to 2 by increasing B_R by 1.

The worst-case of B_W for increment is given by *Case* 4 where $B_W = 3$ since S and one bit each in X_H and X_L are modified. As it turns out, we can improve

Previous				New	
ℓ	S	x_p	x_{p-1}	S	x_p
$= \ell_{\max}$	0	x	–	$\underline{1}$	x
$= 0$	1	1	–	1	$\underline{0}$
$= 0$	1	0	–	1	$\underline{1}$
> 0	1	x	1	$\underline{0}$	x
> 0	1	0	0	1	$\underline{1}$
> 0	1	1	0	1	$\underline{0}$

Table 4: Transition Table for the increment step for read-write trade-off where $\ell = \text{Val}(X_L)$, $\ell_{\max} = 2^{|X_L|} - 1$ and $p = \log n + \ell + 1$. Underlines show the changed bits and x represents '*don't care*' condition

B_W further by delaying the resetting of S by one step if we read another bit. Instead of reading just one bit x_p from X_H when $S = 1$, we can read the pair $(x_p,\ x_{p-1})$. If the previously modified bit $x_{p-1} = 1$, then the propagation of carry is complete, else we flip the current bit x_p. The only exception to this case is when $X_L = 0 \ldots 0$ which implies that $p = \log n + 1$ which is the first position in X_H. In this case, only one bit $x_{\log n+1}$ is read and flipped. We modify the increment step as:

Case 3. $S = 1$ and $x_{p-1} = 0$: propagation of carry to continue. $x_{p-1} = 0$ implies that the previous bit was 1 before getting modified. Therefore, flip x_p irrespective of its value and increment X_L. ($B_R = \log n + 3$, $B_W = 2$).

Case 4. $S = 1$ and $x_{p-1} = 1$: The previous bit was 0 before modification, hence carry has been propagated and x_p is not read. Reset S to 0 and increment X_L. ($B_R = \log n + 2$, $B_W = 2$).

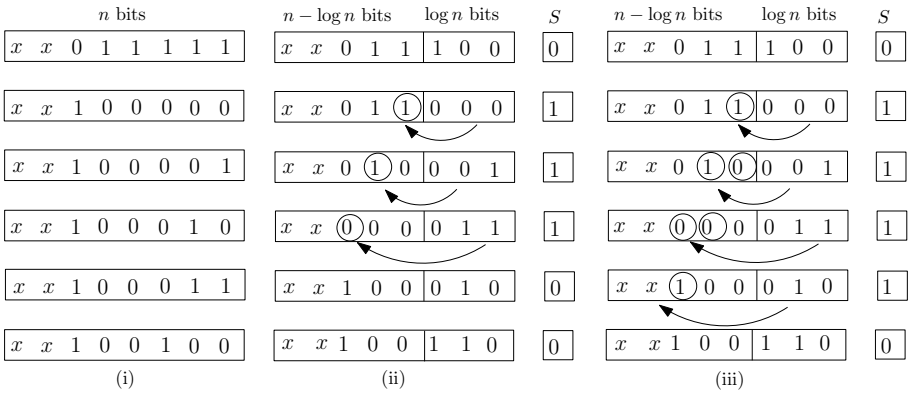


Fig. 3: Increment for a 8-bit number using (i) Standard binary counter (ii) One-bit redundant counter with $B_W = 3$ (iii) with $B_W = 2$. $\log n$ bits are represented using BRGC and x represents '*don't care*' condition

Theorem 4. *There exists a representation of integers of dimension* $n + 1$ *with efficiency* $1/2$ *that supports increment operations with* $B_R = \log n + 3$ *and* $B_W = 2$. *On average, an increment requires* $B_R = O(\log \log n)$ *and* $B_W = 1 + O(n^{-1})$.

3.3 Forbidden state counter with increment

To increase the space-efficiency of the above proposed representation, we modify the data structure proposed in [1] where a particular value of t bits in a dimension n code is used as a forbidden state. A number $X = x_n \dots x_1$ consists of X_H ($x_n \dots x_{\log n+t+1}$), X_F ($x_{\log n+t} \dots x_{\log n+1}$) and X_L ($x_{\log n} \dots x_1$) of $n-\log n-t$, t and $\log n$ bits respectively. Similar to the one-bit redundant counter discussed in Section 3.1, X_H and X_L represent the upper and lower blocks in the number while X_F acts as an alternative to the carry bit S. We use ℓ to refer to the value represented by X_L and $F_{\max}$ refers to the value $2^t - 1$.

All the states for which $\mathrm{Val}(X_F) \leq F_{\max} - 1$ are considered as normal states for X_F and the state where $\mathrm{Val}(X_F) = F_{\max}$ is used to propagate the carry over X_H (conceptually $X_F = F_{\max}$ corresponds to $S = 1$). This representation will allow us to represent $1 - 1/2^t$ of the 2^n numbers. The block X_H is represented using SBC while X_F and X_L are each individually represented using RPGC. Using X_K to represent $\mathrm{Val}(X_K)$, we obtain $\mathrm{Val}(X) = X_L + (X_F + X_H \cdot F_{\max}) \cdot 2^{|X_L|}$ if $X_F < F_{\max}$ and $\mathrm{Val}(X) = X_L + (X_H + 2^\ell) \cdot 2^{|X_L|} \cdot F_{\max}$ if $X_F = F_{\max}$.

Increment. The increment scheme is similar to the one-bit redundant counter of Section 3.1. We first read X_L and X_F. If $X_F \neq F_{\max}$, we increment X_L. If X_L now becomes 0, we also increment X_F. For the case $X_F = F_{\max}$, X_L is used to point to a position p in X_H. If the bit x_p at position p is equal to 1, it is set to 0 and X_L is incremented to point to the next position in X_H. This corresponds to the increment scheme in the one-bit redundant counter when S is set to 1. If X_L now equals $n - \log n - t$, then we incremement X_F (to set $X_F = 0$ and terminate the propagation of carry). On the other hand, if the value of bit x_p is 0, then we set x_p to 1 and X_F is incremented to the next value (which represents state $X_F = 0$. This corresponds to the carry bit S being set to 0 in Section 3.1.

This scheme gives a representation with $B_R = \log n + t + 1$ and $B_W = 3$. Similar to Section 3.2, we can also obtain a representation with $B_R = \log n + t + 2$ and $B_W = 2$ by reading x_{p-1}. The average number of reads and writes to increment the $\log n$ bits in X_L are $O(\log \log n)$ and 1 respectively. The average number of reads and writes to increment X_F are $O(\log t)$ and 1 respectively. Since X_F is incremented once in every n steps, this adds only $o(1)$ to the average number of reads and writes. Similarly, incrementing X_H also takes $o(1)$ reads and writes on average. In addition, at every step we need to check if $\mathrm{Val}(X_F)$ is equal to either $F_{\max}$ or $F_{\max} - 1$ which requires an average of $O(1)$ reads, and finally the cost of reading X_L to find p on average costs at most $O(\frac{1}{2^t}\frac{1}{n}\log n)$. Thus we have the following theorem.

Theorem 5. *Given two integers n and t such that $t \leq n - \log n$, there exists a representation of an integer of dimension n with efficiency $1 - O(2^{-t})$ that supports increment operations with $B_R = \log n + t + 1$ and $B_W = 3$ or $B_R = \log n + t + 2$ and $B_W = 2$. On average, an increment requires $B_R = O(\log \log n)$ and $B_W = 1 + O(n^{-1})$.*

4 Counters with increment and decrement

To support decrement operations interleaved with increment operations, we modify the representation of a number X described in Section 3.1 as follows: a number $X = x_n \dots x_1 x_0$ consists of an upper block X_H $(x_n \dots x_{\log n+2})$, a lower block X_L $(x_{\log n+1} \dots x_1)$ and the bit $S = x_0$ which is used as either a carry bit or a borrow bit. We further split the lower block X_L into two parts: an indicator bit I which consists of the bit $x_{\log n+1}$ and a pointer block X_P consisting of the remaining $\log n$ bits. When the indicator bit I is set to 0, S is interpreted as a carry bit, and when the indicator bit is 1, then S interpreted as a borrow bit.

The $\log n$ bits in X_P are used to point to a location in X_H to perform a delayed carry or borrow. We use BRGC for representing X_L so that an increment or decrement writes only one bit. The block X_H is represented using SBC. Since X_L is represented using BRGC, when $\text{Val}(X_L) < 2^{|X_P|}$, the indicator bit I is equal to 0 and I is equal to 1 otherwise. When $I = 1$, incrementing block X_L corresponds to decrementing the block X_P (unless $X_P = 0$) due to the reflexive property of BRGC [3]. We use these observations in our algorithms for increment and decrement.

The main ideas behind the representation and the increment/decrement algorithms are as follows: when the carry bit S is not set, we perform the increment/decrement in the normal way by incrementing/decrementing X_L. When $S = 0$, $\text{Val}(X_L) = 2^{\log n+1} - 1$ and we perform an increment, we set the bit S and reset the block X_L to $0 \dots 0$. Since I is now set to 0, S will be interpreted as a carry bit untill it is reset again. Similarly, when $S = 0$, $\text{Val}(X_L) = 0$ and we perform a decrement, we set the bit S and decrement X_L to $2^{\log n+1} - 1$. Since I is now set to 1, S will be interpreted as a borrow bit.

To increment X when the carry bit is set, we perform one step of carry propagation in X_H, and then increment X_L. If the propagation finishes in the current step, then we also reset the bit S to 0. To decrement X when the carry bit is set, we first decrement X_L and "undo" one step of carry propagation (i.e., set the bit x_p in X_H to 1). Note that the when performing increments, the carry propagation will finish before we need to change the indicator bit from 0 to 1 (as the length of X_H is less than $2^{\log n}$). The increment and decrement algorithms when the borrow bit are set are similar.

The increment and decrement algorithms are described in the Transition Table 5. Since we read X_L, S and at most one bit in X_H, the read complexity $B_R = \log n + 3$. Since we change at most one bit in each of X_L, X_H and S, the write complexity $B_W = 3$.

The above scheme requires $O(\log n)$ average number of reads as X_L is represented using BRGC and incrementing it requires $O(\log n)$ reads. To get better average-case bounds, we can represent X_P using RPGC. This increases the number of worst-case writes by 1 as now I and X_P are incremented independently. Thus we get a structure with $B_R = O(\log \log n)$ and $B_W = 1 + O(n^{-1})$ on the average but in the worst-case $B_W = 4$.

Increment									
Previous					New				Comments
S	ℓ	I	x_p	x_{p-1}	S	I	x_p	x_{p-1}	
0	$= \ell_{\max}$	0	x	x	0	$\underline{1}$	x	x	Increment X_L (sets I)
0	$= \ell_{\max}$	1	x	x	$\underline{1}$	$\underline{0}$	x	x	Increment X_L (resets X_L), Set S
0	$< \ell_{\max}$	x	x	x	0	x	x	x	Only increment X_P
1	$< \ell_{\max}$	0	-	x	$\underline{0}$	0	-	x	(Position p beyond n) Reset S
1	$< \ell_{\max}$	0	0	x	$\underline{0}$	0	$\underline{1}$	x	(Last step of carry propagation) Reset S
1	$< \ell_{\max}$	0	1	x	1	0	$\underline{0}$	x	(Carry propagation)
1	$< \ell_{\max}$	1	x	1	1	1	x	$\underline{0}$	Undo previous borrow
1	$< \ell_{\max}$	1	x	0	–	–	–	–	Does not occur
Decrement									
0	$= 0$	0	x	x	$\underline{1}$	$\underline{1}$	x	x	Decrement X_L, Set S
0	$= 0$	1	x	x	0	$\underline{0}$	x	x	Decrement X_L (Resets I)
0	> 0	x	x	x	0	x	x	x	Only decrement X_P
1	> 0	1	-	x	$\underline{0}$	1	-	x	(Position p beyond n) Reset S
1	> 0	1	0	x	1	1	$\underline{1}$	x	(Borrow Propagation)
1	> 0	1	1	x	$\underline{0}$	1	$\underline{0}$	x	(Last step of borrow propagation) Reset S
1	> 0	0	x	0	1	0	x	$\underline{1}$	Undo previous carry
1	> 0	0	x	1	–	–	–	–	Does not occur

Table 5: Transition Table for the increment-decrement counter. For increment, new $p = p + 1$ and for decrement, new $p = p - 1$. x represents '*don't care*' condition and - shows that the value does not exist. $\ell = \mathrm{Val}(X_P)$, $p = \log n + \ell + 1$ and $\ell_{\max} = 2^{|X_P|} - 1$. Underlines show the modified values

Theorem 6. *There exists a representation of integers of dimension $n + 1$ with efficiency $1/2$ that supports increment and decrement operations with $B_R = \log n + 3$ and $B_W = 3$. On average, an increment/decrement requires $B_R = O(\log \log n)$ and $B_W = 1 + O(n^{-1})$.*

We can extend the result of Theorem 5 to support decrement operations using an indicator bit as described in Section 4.

Corollary 1. *Given two integers n and t such that $t \leq n - \log n$, there exists a representation of an integer of dimension n with efficiency $1 - O(2^{-t})$ that supports increment and decrement operations with $B_R = \log n + t + 2$ and $B_W = 3$. On average, an increment/decrement requires $B_R = O(\log \log n)$ and $B_W = 1 + O(n^{-1})$.*

5 Addition and Subtraction

In this section, we give a representation for integers which supports addition and subtraction operations efficiently. A number N is said to have a span n if it can take values in the range $[0, \ldots, 2^n - 1]$. Munro and Rahman [4] gave a representation that uses $n + O(\log^2 n)$ bits to represent a number N of span n, and supports adding/subtracting a M of span m to/from N in $O(m + \log n)$ time. We improve the space to $n + O(\log n)$ bits while maintaining the operation

time. We describe the data structure and scheme for addition and introduce suitable modifications to support subtaction as well.

We divide the representation of the number into $k = O(\log n)$ *blocks*: $B_1, B_2, \ldots, B_k$ with $b_1, b_2, \ldots, b_k$ bits respectively, where $b_1 = 2$ and for $2 \leq i \leq k$, $b_i = 2^{i-1}$ (if n is not a power of 2, then the last block has size $b_k = n - 2^{\lfloor \log n \rfloor}$ instead of 2^{k-1}). Note that the block sizes satisfy the property that $\sum_{j=1}^{i} b_j = 2^i = b_{i+1}$, for $1 \leq i \leq k-2$. Each block B_i is maintained using the increment counter of Section 3.1 using $b_i + 1$ bits and a constant number of flag bits as described below. Hence, a number is represented using k blocks of sizes $b_1, b_2, \ldots, b_k$ bits along with $O(k)$ additional bits. The value of the representation is $\mathrm{Val}(B_1) + \sum_{i=2}^{k} \mathrm{Val}(B_i) \cdot 2^{b_i}$. Thus the overall space used is $n + O(k) = n + O(\log n)$ bits.

We now describe the modifications to the increment counter described in Theorem 3. Let X be the counter to be incremented. We introduce two additional bits max and V_H. The bit max indicates whether X represents its maximum value. Assuming $p = \mathrm{Val}(X_L)$ represents a position in X_H, V_H (verifier for block X_H) $= 1$ if all positions in X_H from $0 \ldots p$ are 1. By this definition, when p points to any location beyond X_H and $V_H = 1$ then X_H represents its maximum value. In Section 3.1 we used p to point to a location in X_H only when $S = 1$ but now we use p as a pointer in all steps. When $S = 1$, we perform the delayed increment in X_H and when $S = 0$, we read the bit x_p and use it to set/reset V_H. V_H is set to 0 if $x_p = 0$. If $V_H = 0$, then we set it to 1 if $S = 1$ and $x_p = 0$. This case happens when $X_L = 0 \ldots 0$ for a delayed increment. The bit max is set to 1 when $V_H = 1$ and X_L represents its maximum value. When X represents its maximum value, max $= 1$, $V_H = 1$ and $S = 0$. Incrementing the maximum value of X sets $S = 1$, max $= 0$ and resets X_L to its minimum value. The bit $V_H = 1$ is maintained till S is reset to 0, i.e. throughout the delayed increment process.

We represent every block B_i using the above modified counter. To add M to N, for some $m \leq n$, we first find the largest i such that $\sum_{j=1}^{i-1} b_j < m \leq \sum_{j=1}^{i} b_j$ (i.e., $b_i < m \leq b_{i+1}$). We add M to the number represented by the first i blocks of N in $O(m)$ time. If any of the first i blocks has a carry bit set, then we first perform the necessary work and reset the carry bit in the block, and if necessary propagate the carry to the next block. If there is a carry from B_i to B_{i+1}, we propagate this by modifying the bit max_j of the successive blocks until we find the first block B_j such that max_j is set to 0, and increment B_j, altogether in $O(\log n)$ time. The total running time is $O(m + \log n)$ since incrementing the block and propagation of the carry take $O(\log n)$ time each.

The read and write complexities of the addition algorithm can be shown to be $O(m + \log n)$ and $O(m)$ respectively. Since incrementing a counter of span n has a $\Omega(\log n)$ lower bound for the read complexity, these bounds are optimal.

To support subtraction, we use the increment/decrement counter of Theorem 6 to represent each block, along with additional bits to check for the maximum and minimum values of a number. The details shall be provided in the extended version. M can be subtracted from N in $O(m + \log n)$ time similarly,

since the representation of a block supports both increment and decrement operations in $O(\log b)$ time, where b is the length of the block.

Theorem 7. *An integer of span n can be represented by a data structure which uses $n + O(\log n)$ bits such that adding or subtracting an integer of span m can be perfomed by reading $O(m + \log n)$ bits and writing $O(m)$ bits.*

6 Conclusion

We have shown that a number of dimension n can be incremented and decremented by reading strictly less than n bits in the worst-case. For an integer in the range $[0, \ldots, 2^n - 1]$ represented using exactly n bits, our $(n, n-1, 3)$-*counter* reads $n - 1$ bits and writes 3 bits to perform increment/decrement operations. One open problem is to improve the upper bound of $n - 1$ reads for such space-optimal counters. Fredman [2] has shown that performing an increment using BRGC requires n bits to be read in the worst-case but the same is not known for all Gray Codes.

For the case of redundant counters, we have improved the earlier results by implementing increment operations using counters with space-efficiency arbitrarily close to one which write only 2 bits with low read complexity. We have obtained representations which support increment and decrement operations with fewer number of bits read and written in the worst-case and show trade-offs between the number of bits read and written in the worst-case and also between the number of bits read in the average-case and the worst-case. Finally we have also improved the space complexity of integer representations that support addition and subtraction in optimal time.

References

1. Prosenjit Bose, Paz Carmi, Dana Jansens, Anil Maheshwari, Pat Morin, and Michiel H. M. Smid. Improved methods for generating quasi-gray codes. In Haim Kaplan, editor, *SWAT*, volume 6139 of *Lecture Notes in Computer Science*, pages 224–235. Springer, 2010.
2. Michael L. Fredman. Observations on the complexity of generating quasi-gray codes. *SIAM Journal on Computing*, 7(2):134–146, 1978.
3. F. Gray. Pulse code communications. U.S. Patent (2632058), 1953.
4. M. Ziaur Rahman and J. Ian Munro. Integer representation and counting in the bit probe model. *Algorithmica*, 56(1):105–127, 2010.