Integer Representations towards Efficient Counting in the Bit Probe Model

Gerth Stølting Brodal[a], Mark Greve[a], Vineet Pandey[b], Srinivasa Rao Satti[c,1]

[a] *MADALGO[2], Department of Computer Science, Aarhus University, IT Parken, Åbogade 34, 8200 Aarhus N, Denmark.*
[b] *Computer Science & Information Systems, BITS Pilani, 333031, India.*
[c] *School of Computer Science and Engineering, Seoul National University, 599 Gwanakro, Gwanak-Gu, Seoul 151-744, Republic of Korea.*

Abstract

We consider the problem of representing integers in close to optimal number of bits to support increment and decrement 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. We propose representations, called *counters*, with different trade-offs between the space used and the number of bits probed. A counter is *space-optimal* if it represents any integer 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. This is the first space-optimal representation which supports these operations by always reading strictly less than n bits. For *redundant counters* where we only need to represent integers in the range $[0, \ldots, L-1]$ for some integer $L < 2^n$ using n bits, we define the *space-efficiency* of the counter as the ratio $L/2^n$. We provide representations that achieve different trade-offs between the read/write-complexity and the efficiency.

We also examine the problem of representing integers to support addition and subtraction operations. We propose a representation of integers using n bits and with space efficiency at least $1/n$, which supports addition and subtraction operations, improving the efficiency of an earlier representation by Munro and Rahman [Algorithmica, 2010]. We also show various trade-offs between the operation times and the space complexity.

Email addresses: gerth@cs.au.dk (Gerth Stølting Brodal), mgreve@cs.au.dk (Mark Greve), vineetp13@gmail.com (Vineet Pandey), ssrao@cse.snu.ac.kr (Srinivasa Rao Satti)

[1] This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Grant number 2012-0008241).

[2] Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

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

1. Introduction

We consider the problem of representing integers in close to optimal number of bits to support increment and decrement operations efficiently. We propose data structures for representing integers 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 integers. We study the problem in the bit probe model of computation which was introduced to discuss the average-case bit probe complexity of the *set membership problem* [10]. Even as the cell probe model [13] has become more prevalent, the bit probe model has been studied in theoretical computer science [9]. In the bit probe model, the complexity measure includes only the bitwise accesses to the data structure and not the resulting computations. The running time of an algorithm is given by the number of bits probed (read) to support the operation. We also measure the *write-complexity* as the number of bits written to perform an operation.

1.1. Problem definition and notation

A *counter* is a data structure which represents integers modulo L, for any positive integer L, using d bits where $L \leq 2^d$; we refer to d as the *dimension* of the counter. The data structure supports two operations called *increment* and *decrement* where increment (decrement) refers to changing the counter to represent its next (previous) value modulo L. We define a partial function $\text{Val} : \{0,1\}^d \to \{0,\ldots,L-1\}$, which is used to find the numeric value of a counter.

We represent a counter using a bitstring X and show how to manipulate X to support the counter's operations. For any bitstring X and integer v, if $\text{Val}(X) = v$ then we say that v is *represented* by the bitstring X.

We define the *space-efficiency* of a counter as the ratio $L/2^d$. Space-efficiency equal to one implies that $L = 2^d$ and the counter is called *space-optimal*. A counter with a space-efficiency less than one is called a *redundant counter*. For a redundant counter, some integers can have more than one representation while some representations might not correspond to any integer value.

For counters of dimension d with space-efficiency e, we define a (d, e, r, w)-*scheme* as a description of the increment and decrement operations which can be performed by reading r bits and writing w bits in the worst-case.

We define a *code* as any cyclic sequence of 2^d distinct d-bit strings. We use $X = x_d x_{d-1} \ldots x_1$ to denote a bitstring in a code. To measure the complexity of increment/decrement operations, we use R and W to denote the number of bits read and written respectively. The average number of bits read (written) to perform increment/decrement is computed by adding the total number of bits read (written) to perform L increments/decrements starting from zero and dividing this by L. 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 any integer $c \geq 1$.

Dimension	Space-efficiency	Bits read (R)		Bits written (W)	Inc. & Dec.	Ref.
		Average-case	Worst-case	Worst-case		
n	1	$2 - 2^{1-n}$	n	n	Y	Binary
		n		1	Y	[5]
		$6 \log n$		1	Y	[1]
		$O(\log^{(2c-1)} n)$		c	N	[1]
$n+1$	1/2	$O(1)$	$\log n + 4$	4	Y	[11]
$n + \log n$	$2/n - O(2^{-n+1})$	3	$\log n + 1$	$\log n + 1$	Y	[3]
$n + t \log n$	$1 - O(n^{-t})$	$O(\log^{(2c)} n)$	$O(t \log n)$	$2c + 1$ $(c \geq 1)$	N	[1]
$O(n)$	$1/2^{O(n)}$	$O(\log n)$	$O(\log n)$	1	N	[4]

Table 1: Summary of previous results

1.2. Previous work

The Standard Binary Code (SBC) uses n bits $x_n x_{n-1} \dots x_1$ to represent an integer in the range $[0, \dots, 2^n - 1]$, where $\text{Val}(X) = \sum_{i=1}^{n} x_i 2^{i-1}$. An SBC gives an $(n, 1, n, n)$-*scheme*, but the average number of bits read and written to perform increment/decrement is 2. A Gray code is any code in which successive bitstrings in the sequence differ in exactly one position. Gray codes have been studied extensively owing to their utility in digital circuits [12]. The problem of generating Gray codes has also been discussed by Knuth [8]. The Binary Reflected Gray Code (BRGC) [5] gives an $(n, 1, n, 1)$-*scheme* for increment/decrement. Bose et al. [1, 7, 6] have developed a different Gray code called Recursive Partition Gray Code (RPGC). A counter of dimension n using RPGC requires on average $O(\log n)$ reads to perform increment operations. For the Gray codes BRGC and RPGC, we define $\text{Val}(X)$ as the number of times one needs to increment the string $0 \dots 0$ to obtain X.

For redundant counters, Munro and Rahman gave an $(n+1, 1/2, \log n+4, 4)$-*scheme* [11], i.e., using one additional bit of space the worst-case number of bits read can be reduced form n to $\log n+4$. For efficiency close to one, Bose et al. [1] describe an $(n+t \log n, 1-O(n^{-t}), O(t \log n), 3)$-*scheme* for any parameter $t > 0$. Fredman [4] provided a redundant $(O(n), 1/2^{O(n)}, O(\log n), 1)$-*scheme*, that with a constant factor space overhead supports increments with a logarithmic number of bit reads and a single bit write. The previous results are summarized in Table 1.

To add an integer M to an integer N, where M and N are represented in SBC or BRGC using m and n bits respectively, where $m \leq n$, we need $O(n)$ time in the worst-case. Munro and Rahman [11] gave a representation which uses $n + O(\log^2 n)$ bits to represent an integer in the range $[0, \dots, 2^n - 1]$. Peforming addition or subtraction using this representation takes $O(m + \log n)$ time.

1.3. Our results

For space-optimal counters, we define an (n, r, w)-*scheme* as a (d, e, r, w)-*scheme* with $d = n$ and $e = 1$. We present a $(4, 3, 2)$-*scheme* obtained by exhaustive search and use it to construct an $(n, n-1, 3)$-*scheme* which supports increment and decrement operations reading at most $n - 1$ bits. All previously

Dimension	Space-efficiency	Average-case		Worst-case		Inc. & Dec.	Ref.
		R	W	R	W		
4	1	3	1.25	3	2	Y	Th. 1
n		$O(\log n)$	$1+O(2^{-n})$	$n-1$	3		Th. 2
n	$\geq 1-\frac{1}{2^t}$	$O(\log\log n)$	$1+O(n^{-1})$	$\log n+t+1$	3	N	Th. 3, 5
				$\log n+t+2$	2		Th. 4, 5
				$\log n+t+3$	1		Th. 6
				$\log n+t+2$	3	Y	Th. 7, 8
				$\log n+t+3$	2		

Table 2: Summary of our results for increment/decrement operations

known results for space-optimal counters read n bits in the worst-case. For example, the codes BRGC and RPGC immediately give us $(n, n, 1)$-*schemes*. Bose et al. [1] state the conjecture that for Gray codes, any counter of dimension n requires n bits to be read in the worst-case to increment the counter. If this conjecture is true, this would imply that if there exists a space optimal counter 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.

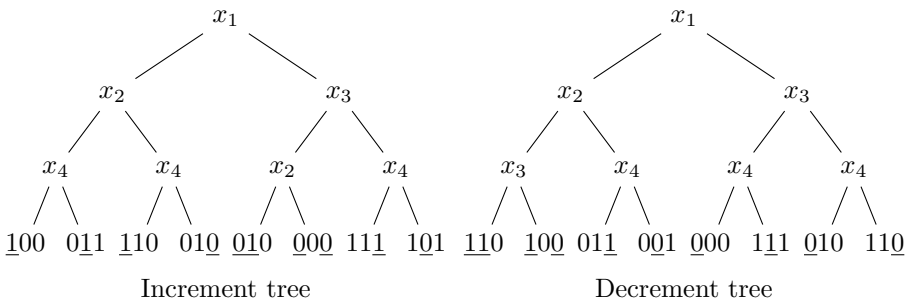
For redundant counters, we provide an $(n, 1/2, \log n+3, 3)$-*scheme*. We can further reduce the number of bits written to 2 by reading $\log n+4$ bits using the one bit read-write trade-off described in Section 3.2. We also provide an $(n, 1-2^{-t}, \log n+t+2, 2)$-*scheme* for any integer $1 \leq t \leq n-\log n-1$. 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 which improves the write-complexity of [1]. Our results are summarized in Table 2.

To support addition and subtraction operations efficiently, we introduce data structures based on the counters mentioned above. We give a representation which uses $n+O(\log n)$ bits to represent an integer in the range $[0, \ldots, 2^n-1]$ and supports addition and subtraction operations by reading $O(m+\log n)$ bits and writing $O(m)$ bits, improving the space complexity of Munro and Rahman [11]. We also show different trade-offs between the number of bits read in the worst-case and the number of bits used to represent a number while maintaining the write-complexity.

2. Space-optimal counters with increment and decrement

In this section, we describe space-optimal counters which are constructed using a $(4, 3, 2)$-*scheme*.

$(4, 3, 2)$-scheme. Figure 1 shows our $(4, 3, 2)$-*scheme* which represents integers from $0 \ldots 15$. The scheme was obtained through a brute force search. Assuming the integer is of the form $x_4x_3x_2x_1$, the increment and decrement trees for our $(4, 3, 2)$-*scheme* are shown in Figure 1. For any internal node corresponding to reading bit x_t, the left edge corresponds to $x_t = 0$ and the right edge corresponds



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

Figure 1: The increment and decrement trees for the $(4,3,2)$-*scheme* and the generated sequence (numbers above the arrows are the number of bits changed)

to $x_t = 1$. The leaves contain information about the new values for the bits read and the modified bits are shown underlined in the tree and the text. It is easy to see that a space-optimal scheme can only write to bits that were read.

As an example, for the fifth leaf from the left in the increment tree, old $x_1x_3x_2 = 100$ and new $x_1x_3x_2 = \underline{01}0$. To increment 9, for example, we take its representation in the $(4,3,2)$-*scheme* 0111 and go through the path $x_1x_3x_4 = 110$ in the increment tree to reach the seventh leaf; so the new values are $x_1x_3x_4 = 11\underline{1}$ and the new representation is $\underline{1}111$ which represents 10 (ten). To decrement 9, we go through the path $x_1x_3x_4 = 110$ in the decrement tree to reach the seventh leaf; so the new values are $x_1x_3x_4 = \underline{0}10$ and the representation is $011\underline{0}$ which represents 8.

Theorem 1. *There exists a space-optimal counter of dimension* 4 *which supports increment and decrement operations with* $R = 3$ *and* $W = 2$ *in the worst-case. On average, an increment/decrement requires* $R = 3$ *and* $W = 1.25$.

2.1. Constructing an $(n, n-1, 3)$*-scheme using a* $(4,3,2)$*-scheme*

We construct an n-bit space-optimal counter for $n > 4$ by dividing the representation for an integer 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 $(4,3,2)$-*scheme* and X_{G} uses RPGC [1]. To increment X, we first increment X_{G} and then check if it represents 0. If X_{G} is 0, then we increment $X_{(4,3,2)}$. To decrement, we decrement X_{G}, and decrement $X_{(4,3,2)}$ if X_{G} was zero before the decrement.

Worst-case analysis. In the worst-case, increment and decrement require $n-4$ reads and 1 write to increment/decrement X_{G} and then 3 reads and 2 writes to increment/decrement $X_{(4,3,2)}$, providing us with $n-1$ reads and 3 writes overall.

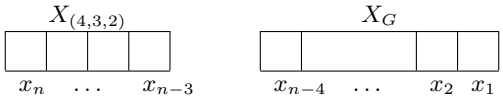


Figure 2: Structure of an $(n, n-1, 3)$-*scheme*

Average-case analysis. X_{G} is represented using RPGC where performing an increment or decrement operation on a counter of dimension n requires $6 \log n$ average number of reads (although [1, Theorem 2] considers only generating the next bitstring, 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 counter 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} bitstrings, 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 space-optimal counter of dimension $n > 4$ which supports increment and decrement operations with $R = n-1$ and $W = 3$ in the worst-case. On average, an increment/decrement requires $R = 6 \log(n-4) + O(2^{-n})$ and $W = 1 + O(2^{-n})$.*

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

2.2. Exhaustive search results

w \ r	1	2	3
1	$\perp$	$\perp$	$+^1$
2	$\perp$	$\perp$	$+$
3	$\perp$	$\perp$	$+$

w \ r	1	2	3	4
1	$\perp$	$\perp$	$\perp$	$+^1$
2	$\perp$	$\perp$	$+^2$	$+$
3	$\perp$	$\perp$	$+$	$+$
4	$\perp$	$\perp$	$+$	$+$

w \ r	1	2	3	4	5
1	$\perp$	$\perp$	$\perp$	?	$+^1$
2	$\perp$	$\perp$	$\perp$	?	$+$
3	$\perp$	$\perp$	$\perp$	$+^2$	$+$
4	$\perp$	$\perp$	$\perp$	$+$	$+$
5	$\perp$	$\perp$	$\perp$	$+$	$+$

Figure 3: Exhaustive search results for (n, r, w)-*scheme* for $n = 3$, 4 and 5 respectively

We used exhaustive search to find (n, r, w)-*schemes* for small values of n. The results are shown in Figure 3 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 (n, r, w)-*scheme* remains unknown for the corresponding value. A value enclosed by a box shows that no counters were found by our brute-force search.

3. Redundant counters with increment

To reduce the number of bits read exponentially, we consider counters with space-efficiency less than one. In this section, we discuss redundant counters supporting increment and which show better results and trade-offs for bits read and written.

3.1. Counters with one-bit redundancy

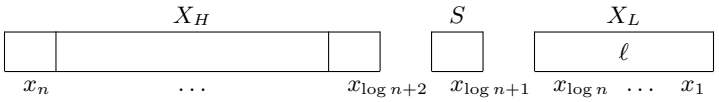


Figure 4: Structure of a one-bit redundant counter

Construction. We use n bits to represent integers in the range $[0 \dots 2^{n-1} - 1]$. An integer X represented by $x_n \dots x_1$ consists of an upper block X_H of the $n - \log n - 1$ bits $x_n \dots x_{\log n+2}$, a *carry bit* $S = x_{\log n+1}$, and a lower block X_L of the $\log n$ bits $x_{\log n} \dots x_1$. We let ℓ refer to the value represented by X_L, and let $p = \ell + \log n + 2$ be a location in X_H. This is used to perform a delayed propagation of the carry as explained below. We represent X_L using the Gray code RPGC, so that an increment operation writes only one bit in X_L. The block X_H is represented using SBC. Using $|X_K|$ to denote the number of bits in X_K, and X_K to also represent $\mathrm{Val}(X_K)$, the value of X is given by:

$$\mathrm{Val}(X) = \ell + 2^{|X_L|} \cdot ((X_H + 2^\ell \cdot S) \bmod 2^{|X_H|}) \ .$$

Increment. The increment step is summarized in Table 3. We determine the number of bits read and written in the worst-case by finding the maximum values of R and W respectively. X_L and S are read at every step, therefore 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 $\ell \geq |X_H|$, we set S to 0. The different cases for increment are the following, where $\ell_{\max} = 2^{|X_L|} - 1$:

Case 1. $S = 0$ and $\ell < \ell_{\max}$: Increment X_L. ($R = \log n + 1$, $W = 1$)

Case 2. $S = 1$, $\ell < |X_H|$, and $x_p = 1$: Propagation of the carry. Change x_p to 0, and increment X_L. ($R = \log n + 2$, $W = 2$)

Case 3. $S = 1$, $\ell < |X_H|$, and $x_p = 0$: Final bit flip in X_H. Change x_p to 1, S to 0, and increment X_L. ($R = \log n + 2$, $W = 3$).

Case 4. $S = 1$ and $\ell = |X_H|$: p is outside X_H, reset carry bit. Set S to 0, and increment X_L. ($R = \log n + 1$, $W = 2$)

Case 5. $S = 0$ and $\ell = \ell_{\max}$: X_L reached the max value, initialize a new carry. Set S to 1, and increment X_L (i.e., set X_L to zero). ($R = \log n + 1$, $W = 2$)

	Previous			New	
	ℓ	S	x_p	S	x_p
1	$< \ell_{\max}$	0	x	0	x
2	$< \lvert X_H \rvert$	1	1	1	$\underline{0}$
3	$< \lvert X_H \rvert$	1	0	$\underline{0}$	$\underline{1}$
4	$= \lvert X_H \rvert$	1	–	$\underline{0}$	–
5	$= \ell_{\max}$	0	–	$\underline{1}$	–

Table 3: Transition table for the increment step where $\ell = \mathrm{Val}(X_L)$, $\ell_{\max} = 2^{|X_L|} - 1$, and $p = \log n + \ell + 2$. Underlines show the changed bits, x represents '*don't care*' condition, and '-' shows that the value does not exist

Average-case analysis. The average number of bits read to increment X_L is $O(\log \log n)$, since X_L is a $(\log n)$-bit RPGC. The bit S is read at every step and it is flipped at most twice out every n steps. When $S = 1$, we also need to read $O(\log n)$ bits to find $\mathrm{Val}(X_L)$, but on average $S = 1$ only for two out of n steps. To test if $\ell = \ell_{\max}$ on average two bits of X_L are read. Thus the average number of bits read is $O(\log \log n)$. The average number of writes is $1 + O(n^{-1})$ since we need to change S at most twice out of n steps and X_H on average at most twice out of n steps. Hence we have the following theorem.

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

3.2. One bit read-write trade-off

We next show how to modify the increment step of the previous section (Theorem 3) to reduce W from 3 to 2 by increasing R by 1.

Increment. The worst-case for the number of writes, W, happens in Case 3 of the increment procedure in Section 3.1 where $W = 3$, since S and one bit each in X_H and X_L are modified. As it turns out, we can improve W further by delaying setting $S = 0$ 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 the carry is complete, else we flip the current bit x_p. The only exception to this case is when $\ell = 0$. In this case, only the first bit of X_H is read and flipped. We modify the cases 2 and 3 of the increment procedure in Section 3.1 as described below. Table 4 shows the resulting transitions.

Cases 2a and 3a $S = 1$ and $\ell = 0$: flip first bit in X_H and increment X_L. $(R = \log n + 2, W = 2)$.

Case 2b and 3b $S = 1$, $0 < \ell < |X_H|$, and $x_{p-1} = 0$: propagation of the 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. $(R = \log n + 3, W = 2)$.

	Previous				New	
	ℓ	S	x_p	x_{p-1}	S	x_p
1	$< \ell_{\max}$	0	x	x	0	x
2a	$= 0$	1	1	–	1	$\underline{0}$
2b	$0 < \ell < \lvert X_H \rvert$	1	1	0	1	$\underline{0}$
3a	$= 0$	1	0	–	1	$\underline{1}$
3b	$0 < \ell < \lvert X_H \rvert$	1	0	0	1	$\underline{1}$
3c	$0 < \ell < \lvert X_H \rvert$	1	x	1	$\underline{0}$	x
4	$= \lvert X_H \rvert$	1	–	x	$\underline{0}$	–
5	$= \ell_{\max}$	0	–	–	$\underline{1}$	–

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

Case 3c $S = 1$ 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. ($R = \log n + 2$, $W = 2$).

This improves the overall worst-case write-complexity to 2. The number of reads and writes required in average-case are $O(\log \log n)$ and $1 + O(n^{-1})$ respectively as described in Section 3.1. The value of a counter is

$$\mathrm{Val}(X) = \Big(\ell + 2^{|X_L|} \cdot \big(X_H + 2^{\ell} \cdot \chi(X)\big)\Big) \bmod 2^n \ ,$$

where $\chi(X) = 1$ if $S = 1 \wedge (\ell = 0 \vee x_{p-1} = 0)$; otherwise $\chi(X) = 0$. We show a sequence of increments for an 8-bit integer using standard binary code and a one-bit redundant counter in Figure 5.

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

3.3. Forbidden-state counter

To increase the space-efficiency of the above proposed representation, we modify the data structure proposed in [1] for a counter of dimension n, which uses a particular set of t bits as a *forbidden state*.

Construction. An integer $X = x_n \dots x_1$ 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 integer 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 $\mathrm{Val}(X_F) < F_{\max}$ 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

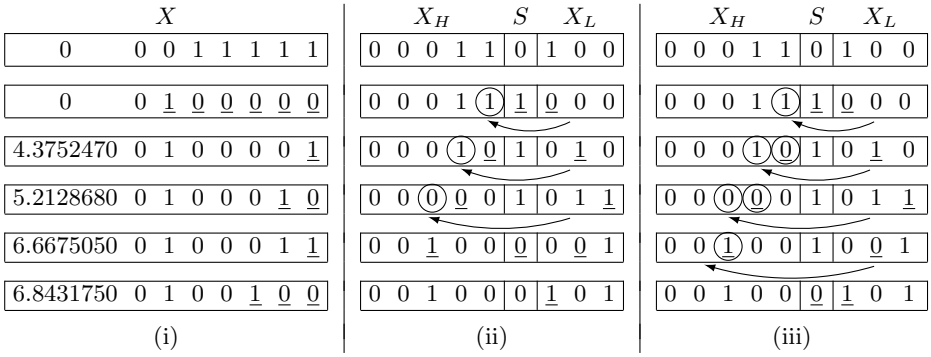


Figure 5: Five increments for an 8-bit integer using (i) Standard binary code (SBC), (ii) One-bit redundant counter with $W = 3$, and (iii) with $W = 2$. X_H and X_L are represented using SBC and RPGC respectively

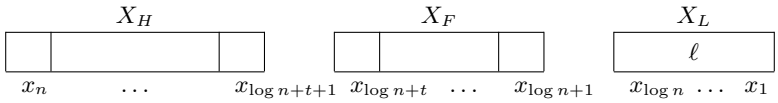


Figure 6: Structure of a forbidden-state counter

(conceptually $\mathrm{Val}(X_F) = F_{\max}$ corresponds to $S = 1$). This representation will allow us to represent a $1 - 1/2^t$ fraction of the 2^n numbers and hence has efficiency equal to $1 - 1/2^t$. The block X_H is represented using SBC while X_F and X_L are each individually represented using RPGC. The value of of a counter is

$$\mathrm{Val}(X) = \ell + 2^{|X_L|} \cdot \begin{cases} F_{\max} \cdot X_H + X_F & \text{if } X_F < F_{\max} \,, \\ F_{\max} \cdot ((X_H + 2^\ell) \bmod 2^{|X_H|}) & \text{if } X_F = F_{\max} \,. \end{cases}$$

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 $X_F = F_{\max}$, block 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 increment X_F (to set $X_F = 0$ and terminate the propagation of the carry). On the other hand, if the value of bit x_p is 0, we set x_p to 1 and X_F is incremented (to set $X_F = 0$). This corresponds to the carry bit S being set to 0 in Section 3.1.

Worst-case analysis. The above scheme gives a representation with $R = \log n + t + 1$ and $W = 3$. Similar to Section 3.2, we can also obtain a representation with $R = \log n + t + 2$ and $W = 2$ by reading x_{p-1} and postponing incrementing X_F from $F_{\max}$ to 0 by one step.

Previous				New			
ℓ	X_F	x_p	x_{p-1}	ℓ	X_F	x_p	x_{p-1}
$< \ell_{\max}$	$< F_{\max} - 1$, even	x	x	$\underline{\ell + 1}$	X_F	x	x
$\ell_{\max}$	$< F_{\max} - 1$, even	x	x	ℓ	$\underline{X_F + 1}$	x	x
> 0	$< F_{\max} - 1$, odd	x	x	$\underline{\ell - 1}$	X_F	x	x
0	$< F_{\max} - 1$, odd	x	x	ℓ	$\underline{X_F + 1}$	x	x
0	$F_{\max} - 1$	0	$-$	ℓ	$\underline{F_{\max}}$	0	$-$
0	$F_{\max} - 1$	1	$-$	$\underline{1}$	$F_{\max} - 1$	1	$-$
$0 < \ell \leq \lvert X_H \rvert$	$F_{\max} - 1$	x	1	ℓ	$F_{\max} - 1$	x	$\underline{0}$
$0 < \ell < \lvert X_H \rvert$	$F_{\max} - 1$	1	0	$\underline{\ell + 1}$	$F_{\max} - 1$	1	0
$0 < \ell < \lvert X_H \rvert$	$F_{\max} - 1$	0	0	ℓ	$\underline{F_{\max}}$	0	0
$\lvert X_H \rvert$	$F_{\max} - 1$	$-$	0	ℓ	$\underline{F_{\max}}$	$-$	0
$0 \leq \ell < \lvert X_H \rvert$	$F_{\max}$	0	x	ℓ	$F_{\max}$	$\underline{1}$	x
$0 \leq \ell < \lvert X_H \rvert$	$F_{\max}$	1	x	ℓ	$\underline{0}$	1	x
$\lvert X_H \rvert$	$F_{\max}$	$-$	x	ℓ	$\underline{0}$	$-$	x

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

Average-case analysis. 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 at most twice in every n steps, this adds only $O(\frac{\log t}{n})$ and $O(\frac{1}{n})$ to the average number of reads and writes, respectively. Similarly, incrementing X_H also takes $O(\frac{1}{n \cdot F_{\max}})$ 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. Finally the cost of reading X_L to find p on average costs at most $O(\frac{1}{n \cdot F_{\max}} \log n)$. Thus we have the following theorem.

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

3.4. Redundant counters with increment using one bit write

In this section we present a scheme supporting increment with only one bit write. The main idea is to modify the scheme of Theorem 5 with $W = 2$ and $R = \log n + t + 2$ to have two forbidden states $F_{\max} - 1$ and $F_{\max}$, for $t \geq 2$. Whenever the increment algorithm in Section 3.3 needs to flip two bits simultaneously, we instead perform it over two increments. More specifically, carry propagation happens in the state $X_F = F_{\max} - 1$ where we alternate between clearing x_{p-1} and incrementing ℓ. For the final carry position we have the state $X_F = F_{\max}$, where we set x_p.

As in Section 3.3 a counter consists of three parts X_H, X_F, and X_L of $n - \log n - t$, t, and $\log n$ bits, respectively, where X_H is represented using

SBC, and X_F and X_L are represented using RPGC (see Figure 6). We let $\ell = \mathrm{Val}(X_L)$. When X_F is not in a forbidden state, we increment X_L in the order 0 to $\ell_{\max} = 2^{|X_L|} - 1$ when X_F is even, and in the reverse order from $\ell_{\max}$ down to 0 when X_F is odd. The transition table for the resulting increment scheme is shown in Table 5.

We do not give an explicit expression for the value of a counter X using this scheme. Instead, let L be the smallest value such that for some bit string of length n, performing L increments on that string returns the same bit string. We define the value of a counter X as the number of times, modulo L, we need to perform the increment operation on the bit string 0^n to reach X. The efficiency of this counter is between $1 - 2/2^t$ and $1 - 1/2^t$, since we do not increment ℓ in each increment of the counter and we have two forbidden states.

Theorem 6. *Given two integers n and t such that $2 \leq t \leq n - \log n$, there exists a counter of dimension n with efficiency at least $1 - 2^{1-t}$ which supports increment operations with $R = \log n + t + 2$ and $W = 1$. On average, an increment requires $R = O(\log \log n)$.*

4. Redundant counters with increment and decrement

Construction. To support decrement operations interleaved with increment operations, we modify the representation of an integer X described in Section 3.1 as follows: an integer $X = x_n \dots x_1$ consists of an upper block $X_H = x_n \dots x_{\log n+3}$, a bit $S = x_{\log n+2}$, an indicator bit $X_I = x_{\log n+1}$, and a lower block $X_L = x_{\log n} \dots x_1$. The bit S is interpreted as either a carry bit or a borrow bit: When the indicator bit X_I is set to 0, S is interpreted as a carry bit, and when the indicator bit is 1, then S is interpreted as a borrow bit.

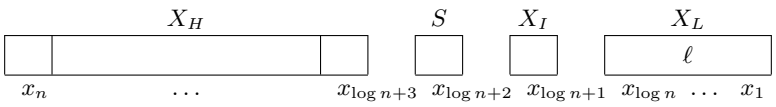


Figure 7: Structure of a one-bit redundant counter with increment and decrement

We use RPGC to represent X_L so that an increment or decrement writes only one bit in X_L. The block X_H is represented using SBC. The value $\ell = \mathrm{Val}(X_L)$ is used to point to a location in X_H to perform a delayed carry or borrow. Borrowing ideas from RPGC used for dealing with counters with an odd number of bits, we will let X_I and X_L together denote a value $\mathrm{Val}(X_I X_L)$ in the range $[0 \dots 2^{\log n+1} - 1]$. To increment $X_I X_L$, we first check the value of X_I. If $X_I = 0$ and ℓ is not the maximal value $\ell_{\max} = 2^{\log n} - 1$, we increment X_L. If $X_I = 0$ and $\ell = 2^{\log n} - 1$, we set $X_I = 1$. If $X_I = 1$ and $\ell > 0$ we decrement X_L. Finally if $X_I = 1$ and $\ell = 0$ we set $X_I = 0$. Decrementing $X_I X_L$ is similar. The relationship between ℓ, X_I and $\mathrm{Val}(X_I X_L)$ is shown in Table 6. The value of the counter X is given by the expression

$$\mathrm{Val}(X) = \mathrm{Val}(X_I X_L) + 2^{|X_L|+1} \cdot \left(\left(X_H + \chi(X) \cdot 2^{\ell} \right) \bmod 2^{|X_H|} \right) ,$$

where $\chi(X)$ is the signed carry

$$\chi(X) = \begin{cases} 0 & \text{if } S = 0\ , \\ 1 & \text{if } S = 1 \wedge X_I = 0\ , \\ -1 & \text{if } S = 1 \wedge X_I = 1\ . \end{cases}$$

$\mathrm{Val}(X_I X_L)$	0	1	2	$\ldots$	$2^{\lvert X_L\rvert} - 1$	$2^{\lvert X_L\rvert}$	$2^{\lvert X_L\rvert} + 1$	$2^{\lvert X_L\rvert} + 2$	$\ldots$	$2^{\lvert X_L\rvert+1} - 1$
X_I	0	0	0	0	0	1	1	1	1	1
ℓ	0	1	2	$\ldots$	$2^{\lvert X_L\rvert} - 1$	$2^{\lvert X_L\rvert} - 1$	$2^{\lvert X_L\rvert} - 2$	$2^{\lvert X_L\rvert} - 3$	$\ldots$	0

Table 6: Relationship between ℓ, X_I and $\mathrm{Val}(X_I X_L)$

Increment and decrement operations. The main ideas behind the representation and the increment/decrement algorithms are as follows: when the carry bit $S = 0$ and $X_I X_L$ is not the maximum/minimum value, we perform the increment/decrement by incrementing/decrementing $X_I X_L$. To increment X when $S = 0$, $X_I = 1$ and $\ell = 0$ (i.e., $X_I X_L$ has the maximum value), we set $S = 1$ and $X_I = 0$. Since X_I is now set to 0, S will be interpreted as a carry bit until it is reset again. Similarly, to decrement X, when $S = 0$, $X_I = 0$ and $\ell = 0$ (i.e., $X_I X_L$ has value zero), we set $S = 1$ and $X_I = 1$. Since X_I is now set to 1, S will be interpreted as a borrow bit.

To increment X when $S = 1$ and $X_I = 0$, 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 S to 0. To decrement X when $S = 1$ and $X_I = 0$, 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 when performing increments, the carry propagation will finish before we need to change the indicator bit X_I 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, i.e., $S = 1$ and $X_I = 1$, are similar. The details of the increment and decrement algorithms are described in Table 7.

Worst-case analysis. Since we read X_L, X_I, S and at most one bit in X_H, the read complexity $R = \log n + 3$. Since we change either X_I or at most one bit in X_L, and one bit in X_H and S, the write-complexity $W = 3$.

Average-case analysis. The above scheme requires $O(\log \log n)$ average number of reads, as X_L is represented using RPGC and incrementing it requires $O(\log \log n)$ reads on average.

Similarly to Section 3.2 we can move the cost of writing one bit to the reading cost by postponing clearing the carry bit S by one increment/decrement, such that we never have to flip both S and x_p during one increment/decrement.

Theorem 7. *There exists a counter of dimension n with efficiency $1/2$ which supports increment and decrement operations with $R = \log n + 3$ and $W = 3$ or $R = \log n + 4$ and $W = 2$. On average, an increment/decrement requires $R = O(\log \log n)$ and $W = 1 + O(n^{-1})$.*

Increment										
Previous					New					Comments
S	X_I	ℓ	x_p	x_{p-1}	S	X_I	ℓ	x_p	x_{p-1}	
0	0	$\ell < \ell_{\max}$	x	x	0	0	$\underline{\ell+1}$	x	x	
0	0	$\ell = \ell_{\max}$	x	x	0	$\underline{1}$	$\ell_{\max}$	x	x	Normal increment
0	1	$\ell > 0$	x	x	0	1	$\underline{\ell-1}$	x	x	
0	1	$\ell = 0$	x	x	$\underline{1}$	$\underline{0}$	$\ell = 0$	x	x	Set carry
1	0	$\lvert X_H\rvert > \ell \geq 0$	1	x	1	0	$\underline{\ell+1}$	$\underline{0}$	x	Carry propagation
1	0	$\lvert X_H\rvert > \ell \geq 0$	0	x	$\underline{0}$	0	$\underline{\ell+1}$	$\underline{1}$	x	(End of carry propagation) Reset S
1	0	$\ell = \lvert X_H\rvert$	–	1	$\underline{0}$	0	$\underline{\ell+1}$	–	1	(End of X_H) Reset S
1	1	$\ell = 0$	x	–	$\underline{0}$	$\underline{0}$	$\ell = 0$	x	–	Reset borrow
1	1	$\lvert X_H\rvert > \ell > 0$	x	1	1	1	$\underline{\ell-1}$	x	$\underline{0}$	Undo one step of borrow
Decrement										
0	1	$\ell < \ell_{\max}$	x	x	0	1	$\underline{\ell+1}$	x	x	
0	1	$\ell = \ell_{\max}$	x	x	0	$\underline{0}$	$\ell_{\max}$	x	x	Normal decrement
0	0	$\ell > 0$	x	x	0	0	$\underline{\ell-1}$	x	x	
0	0	$\ell = 0$	x	x	$\underline{1}$	$\underline{1}$	$\ell = 0$	x	x	Set borrow
1	1	$\lvert X_H\rvert > \ell \geq 0$	0	x	1	1	$\underline{\ell+1}$	$\underline{1}$	x	Borrow propagation
1	1	$\lvert X_H\rvert > \ell \geq 0$	1	x	$\underline{0}$	1	$\underline{\ell+1}$	$\underline{0}$	x	(End of borrow propagation) Reset S
1	1	$\ell = \lvert X_H\rvert$	–	0	$\underline{0}$	1	$\underline{\ell+1}$	–	0	(End of X_H) Reset S
1	0	$\ell = 0$	x	–	$\underline{0}$	$\underline{1}$	$\ell = 0$	x	–	Reset carry
1	0	$\lvert X_H\rvert > \ell > 0$	x	0	1	0	$\underline{\ell-1}$	x	$\underline{1}$	Undo one step of carry

Table 7: Transition table for the increment-decrement counter. $\ell = \mathrm{Val}(X_L)$, $p = \log n + \ell + 3$ and $\ell_{\max} = 2^{|X_L|} - 1$. x represents '*don't care*' condition and '-' shows that the value does not exist. Underlines show the modified values. The cases which cannot occur are not shown in the table

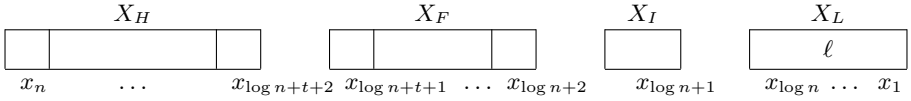


Figure 8: Structure of a forbidden-state counter with increment and decrement

We can improve the efficiency of the counter by adopting the ideas of Section 3.3 to replace S by a t-bit counter X_F with a forbidden state $F_{\max} = 2^t - 1$ representing $S = 1$ (see Figure 8).

The value of a counter becomes $\mathrm{Val}(X) = \mathrm{Val}(X_I X_L) +$

$$2^{|X_L|+1} \cdot \begin{cases} F_{\max} \cdot X_H + X_F & \text{if } X_F < F_{\max}\ , \\ F_{\max} \cdot ((X_H + 2^\ell) \bmod 2^{|X_H|}) & \text{if } X_F = F_{\max} \wedge X_I = 0\ , \\ F_{\max} \cdot ((X_H - 2^\ell) \bmod 2^{|X_H|}) + F_{\max} - 1 & \text{if } X_F = F_{\max} \wedge X_I = 1\ . \end{cases}$$

We can furthermore combine the forbidden-state idea with the one-bit read-write trade-off to decrease the number of bits written by one and increase the number of bits read by one.

Theorem 8. *Given two integers n and t such that $t \leq n - \log n$, there exists a counter of dimension n with efficiency $1 - 2^{-t}$ which supports increment and decrement operations with $R = \log n + t + 2$ and $W = 3$ or $R = \log n + t + 3$ and $W = 2$. On average, an increment/decrement requires $R = O(\log \log n)$ and $W = 1 + O(n^{-1})$.*

Efficiency	Read	Write	Reference
$\Theta(1/n^{\log n})$	$O(m + \log n)$	$O(m)$	[11]
$\geq (1 - 1/2^t)^{\log n}$	$O(m + t \log n)$	$O(m)$	Theorem 9
$\Omega(1/n)$	$O(m + \log n)$	$O(m)$	Corollary 1
$\Theta(1)$	$O(m + \log n \log\log n)$	$O(m)$	Corollary 2

Table 8: Summary of results for addition and subtraction operations

5. Addition and Subtraction

In this section, we give a representation for integers which supports addition and subtraction operations efficiently. Munro and Rahman [11] gave a representation that with n bits achieves efficiency $\Theta(1/2^{\log^2 n})$, i.e., uses $O(\log^2 n)$ bits of redundancy. The addition/subtraction of an m bit integer to an n bit integer in this representation, where $m \leq n$, is supported in $O(m + \log n)$ time. We improve the efficiency to $O(1/n)$, i.e., $O(\log n)$ bits of redundancy, while maintaining the operation times. We also achieve trade-offs between the space efficiency and the number of bits read as summarized in Table 8.

Construction. In the following we let t be an integer parameter, $1 \leq t \leq n$, where increasing values of t will result in increased space efficiency of our representations. Let $b^{(i)} = (t + 2) \cdot 2^{i-1}$ for $i \geq 1$. A counter X of n bits is represented by k *blocks* $B^{(1)}, B^{(2)}, \ldots, B^{(k)}$, where $B^{(i)}$ consists of $b^{(i)}$ bits for $1 \leq i < k$, and $B^{(k)}$ consists of between one and $b^{(k)}$ bits. It follows that $k = \log(1 + n/(t + 2))$. The representation of block $B^{(i)}$ is very similar to the counters in Section 4: Block $B^{(i)}$ consists of the parts $X_L^{(i)}$, $X_I^{(i)}$, $X_F^{(i)}$, and $X_H^{(i)}$, consisting of $1+\log(|B^{(i)}|-t-1)$, 1, t, and $|B^{(i)}|-t-1-|X_L^{(i)}|$ bits, respectively (see Figure 9). $X_H^{(i)}$ is represented using SBC, $X_F^{(i)}$ and $X_L^{(i)}$ using BRGC, and $X_I^{(i)}$ is an indicator bit. The only exception is if $|B^{(k)}| < t+2$, in which case $B^{(k)}$ is represented by a single BRGC. We let $\ell^i = \mathrm{Val}(X_L^{(i)})$ and $\ell_{\max}^{(i)} = 2^{|X_L^{(i)}|} - 1$. Note that since $X_L^{(i)}$ is represented using BRGC, $\ell^{(i)} \geq 2^{|X_L^{(i)}|-1}$ iff the leftmost bit $x_{|X_L^{(i)}|}$ in $X_L^{(i)}$ equals one, and note that $|X_H^{(i)}| \leq 2^{|X_L^{(i)}|-1}$.

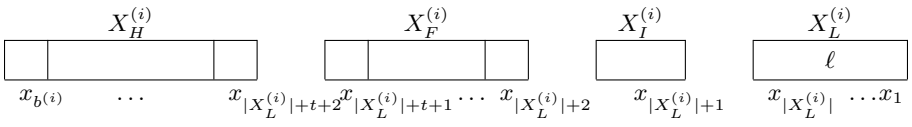


Figure 9: Block B_i of a counter with addition and subtractions

The main difference to the counter in Section 4 is that we have a different number of bits for X_L to ensure $2^{|X_L|} \geq 2|X_H|$, and that we do not compute modulo, to ensure the correct carry/borrow propagation from one block $B^{(i)}$ to the next block $B^{(i+1)}$. The rôle of $B^{(i)}$ is to store values in the range zero to $M^{(i)} - 1$, where $M^{(i)} = F_{\max} \cdot 2^{|X_H|+|X_L|+1}$ and $F_{\max} = 2^t - 1$, except

that $M^{(k)} = 2^{|B^{(k)}|}$ if $|B^{(k)}| < t + 2$. The value of block $B^{(i)}$ is $\mathrm{Val}(B^{(i)}) = \mathrm{Val}(X_I^{(i)} X_L^{(i)}) +$

$$2^{|X_L^{(i)}|+1} \cdot \begin{cases} F_{\max} \cdot X_H^{(i)} + X_F^{(i)} & \text{if } X_F^{(i)} < F_{\max} \,, \\ F_{\max} \cdot (X_H^{(i)} + 2^{\ell^{(i)}}) & \text{if } X_F^{(i)} = F_{\max} \wedge X_I^{(i)} = 0 \wedge \ell^{(i)} < |X_H^{(i)}| \,, \\ F_{\max} \cdot (X_H^{(i)} + 2^{|X_H^{(i)}|}) & \text{if } X_F^{(i)} = F_{\max} \wedge X_I^{(i)} = 0 \wedge \ell^{(i)} \geq |X_H^{(i)}| \,, \\ F_{\max} \cdot (X_H^{(i)} - 2^{\ell^{(i)}}) + F_{\max} - 1 & \text{if } X_F^{(i)} = F_{\max} \wedge X_I^{(i)} = 1 \wedge \ell^{(i)} < |X_H^{(i)}| \,, \\ F_{\max} \cdot (X_H^{(i)} - 2^{|X_H^{(i)}|}) + F_{\max} - 1 & \text{if } X_F^{(i)} = F_{\max} \wedge X_I^{(i)} = 1 \wedge \ell^{(i)} \geq |X_H^{(i)}| \,, \end{cases}$$

where $\mathrm{Val}(X_I^{(i)} X_L^{(i)}) = \mathrm{Val}(X_L^{(i)})$ if $X_I^{(i)} = 0$, and $\mathrm{Val}(X_I^{(i)} X_L^{(i)}) = 2^{|X_L^{(i)}|+1} - 1 - \mathrm{Val}(X_L^{(i)})$ if $X_I^{(i)} = 1$. If $\mathrm{Val}(B^{(i)}) \geq M^{(i)}$ or $\mathrm{Val}(B^{(i)}) < 0$, then we have a *delayed carry/borrow* from $B^{(i)}$ to $B^{(i+1)}$.

Letting $L = \prod_{i=1}^{k} M^{(i)}$, the value of a counter X is

$$\mathrm{Val}(X) = \left(\sum_{i=1}^{k} \mathrm{Val}(B^{(i)}) \prod_{j=1}^{i-1} M^{(j)} \right) \bmod L \,.$$

We say that there is a *critical overflow* in block $B^{(i)}$ if $(X_F^{(i)} = F_{\max}) \wedge (\ell^{(i)} = 2^{|X_L^{(i)}|-1})$. To handle the delayed carries/borrows, we maintain the following invariants. The first invariant

$$\forall i, 1 \leq i \leq k : \ell^{(i)} > 2^{|X_L^{(i)}|-1} \Rightarrow X_F^{(i)} \neq F_{\max} \tag{1}$$

ensures that each block $B^{(i)}$ with a carry/borrow, i.e., $X_F^{(i)} = F_{\max}$, has $\ell^{(i)} \leq 2^{|X_L^{(i)}|-1}$. The second invariant

$$\begin{aligned} \forall i, j, 1 \leq i < j \leq k : (X_F^{(i)} = X_F^{(j)} = F_{\max}) \wedge (\ell^{(i)} = 2^{|X_L^{(i)}|-1}) \wedge (\ell^{(j)} = 2^{|X_L^{(j)}|-1}) \\ \Rightarrow \exists p : (i < p < j) \wedge (X_F^{(p)} \neq F_{\max}) \end{aligned} \tag{2}$$

guarantees that if there is a critical overflow in blocks $B^{(i)}$ and $B^{(j)}$, where $i < j$, then there exists a block $B^{(p)}$, where $i < p < j$, such that there is no carry set in the block $B^{(p)}$, i.e., $X_F^{(p)} < F_{\max}$. This invariant implies that if there is a critical overflow in block $B^{(i)}$, then there is no critical overflow in block $B^{(i+1)}$. The invariant (2) is inspired by a redundant standard binary counter (SBC) from [2], where the "bits" can take values $\{0, 1, 2\}$.

We can check whether a block $B^{(i)}$ has a critical overflow by reading $t + 1$ bits in $B^{(i)}$, namely $X_F^{(i)}$ and the most significant bit of $X_L^{(i)}$.

Addition and subtraction algorithm. Let Y be an m bit integer represented using the above representation stored in k' blocks. To add/subtract Y to/from X, we read Y and the k' first blocks $B^{(1)} \dots B^{(k')}$ of X, using $O(m)$ bit reads. Let $z = \mathrm{Val}(Y) + \sum_{i=1}^{k'} \mathrm{Val}(B^{(i)}) \prod_{j=1}^{i-1} M^{(j)}$ or $z = \mathrm{Val}(Y) - \sum_{i=1}^{k'} \mathrm{Val}(B^{(i)}) \prod_{j=1}^{i-1} M^{(j)}$ when adding or subtracting Y, respectively. Let $L' = \prod_{i=1}^{k'} M^{(i)}$. From the

definition of Val if follows that $-2L' \leq z < 4L'$. We first write $z \bmod L'$ to the first k' blocks of X, using $O(m)$ bit writes, such that there is no carry in any of these blocks. If $0 \leq z < L'$ we are done. Otherwise we have to increment $B^{(k'+1)}$ at most three times or decrement $B^{(k'+1)}$ at most two times (each increment/decrement of $B^{(k'+1)}$ corresponds to incrementing/decrementing X by L').

To perform an increment/decrement on $B^{(k'+1)}$, we first check the blocks from $B^{(k'+1)}$ to $B^{(k)}$ to find the first block $B^{(j)}$ (with the lowest superscript) among these blocks which has a critical overflow. If such a block $B^{(j)}$ exists, we clear the carry/borrow by moving it out of the forbidden state by incrementing or decrementing $X_F^{(j)}$, depending on if $X_I^{(j)}$ is zero or one. If $j < k$, we then perform an increment or decrement on block $B^{(j+1)}$, depending on if $X_I^{(j)}$ is zero or one. To increment $B^{(j)}$ we use the algorithm of Section 4, but where we never change $X_F^{(j)}$ if $X_F^{(j)} = F_{\max}$ and $\ell^{(j)} \geq |X_H^{(j)}|$. From (2), we know that block $B^{(j+1)}$ did not have a critical overflow. Finally, we perform an increment/decrement on $B^{(k'+1)}$ (as on $B^{(j)}$). This ensures that the invariants (1) and (2) are satisfied after the increment.

Analysis. The number of bit reads required to find the block with the critical overflow is $O(tk)$; and the number of bit reads required to increment a block or to move it out of a forbidden state is $O(t + \log n)$. Thus the overall read complexity is $O(m + tk + \log n)$. We modify at most two blocks in addition to the first k' blocks; and in those two blocks, we only modify a constant number of bits. Thus the overall write complexity is $O(m)$. Since the space efficiency of each block is at least $1 - 1/2^t$, the total space efficiency of an n bit integer representation is at least $(1 - 1/2^t)^k$. Since $k \leq \log n$, we have the following theorem.

Theorem 9. *There exists a representation of integers that, with n bits and space efficiency at least $(1 - 1/2^t)^{\log n}$, supports adding/subtracting an m bit integer using $O(m + t \log n)$ bit reads and $O(m)$ bit writes.*

Setting $t = 1$ and $t = \log \log n$ implies the following two corollaries.

Corollary 1. *There exists a representation of integers that, with n bits and space efficiency at least $\Omega(1/n)$, supports adding/subtracting an m bit integer using $O(m + \log n)$ bit reads and $O(m)$ bit writes.*

Corollary 2. *There exists a representation of integers that, with n bits and space efficiency $\Theta(1)$, supports adding/subtracting an m bit integer using $O(m + \log n \log \log n)$ bit reads and $O(m)$ bit writes.*

6. Conclusions

We have shown that a space-optimal counter 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 $(n, n-1, 3)$-*scheme* 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 [4] 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 one bit 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 showed 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.

[1] Prosenjit Bose, Paz Carmi, Dana Jansens, Anil Maheshwari, Pat Morin, and Michiel H. M. Smid. Improved methods for generating quasi-Gray codes. In *Proceedings of the 12th Scandinavian Symposium and Workshops on Algorithm Theory*, volume 6139 of *Lecture Notes in Computer Science*, pages 224–235. Springer-Verlag, 2010.

[2] Michael J. Clancy and Donald E. Knuth. A programming and problem-solving seminar. Technical Report Technical Report STAN-CS-77-606, Computer Science Department, Stanford University, 1977.

[3] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997.

[4] Michael L. Fredman. Observations on the complexity of generating quasi-Gray codes. *SIAM Journal on Computing*, 7(2):134–146, 1978.

[5] Frank Gray. Pulse code communications. U.S. Patent (2632058), 1953.

[6] Dana Jansens. Improved methods for generating quasi-Gray codes. Master's thesis, School of Computer Science, Carleton University, April 2010.

[7] Dana Jansens, Prosenjit Bose, Paz Carmi, Anil Maheshwari, Pat Morin, and Michiel H. M. Smid. Improved methods for generating quasi-Gray codes. *CoRR*, abs/1010.0905, 2010.

[8] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations (Art of Computer Programming)*. Addison-Wesley Professional, 2005.

[9] Peter Bro Miltersen. The bit probe complexity measure revisited. In *Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science*, volume 665 of *Lecture Notes in Computer Science*, pages 662–671. Springer-Verlag, 1993.

[10] Marvin Lee Minsky and Seymour Papert. *Perceptrons*. MIT Press, Cambridge Mass., 1969.

[11] M. Ziaur Rahman and J. Ian Munro. Integer representation and counting in the bit probe model. *Algorithmica*, 56(1):105–127, 2010.

[12] Carla Savage. A survey of combinatorial Gray codes. *SIAM Review*, 39:605–629, 1996.

[13] Andrew Chi-Chih Yao. Should tables be sorted? *J. ACM*, 28:615–628, July 1981.