CrossMark

Fast Prefix Adders for Non-uniform Input Arrival Times

Stephan Held[1] · **Sophie Spirkl**[2] ⓘ

Abstract We consider the problem of constructing fast and small parallel prefix adders for non-uniform input arrival times. In modern computer chips, adders with up to hundreds of inputs occur frequently, and they are often embedded into more complex circuits, e.g. multipliers, leading to instance-specific non-uniform input arrival times. Most previous results are based on representing binary carry-propagate adders as parallel prefix graphs, in which pairs of generate and propagate signals are combined using complex gates called prefix gates. Examples of commonly-used adders are constructed based on the Kogge–Stone or Ladner–Fischer prefix graphs. Adders constructed in this model usually minimize the delay in terms of these prefix gates. However, the delay in terms of logic gates can be worse by a factor of two. In contrast, we aim to minimize the delay of the underlying logic circuit directly. We prove a lower bound on the delay of a carry bit computation achievable by any prefix carry bit circuit and develop an algorithm that computes a prefix carry bit circuit with optimum delay up to a small additive constant. Our algorithm improves the running time of a previous dynamic program for constructing a prefix carry bit from $\mathcal{O}(n^3)$ to $\mathcal{O}(n \log^2 n)$ while simultaneously improving the delay and size guarantee, where n is the number of bits in the summands. Furthermore, we use this algorithm as a subroutine to compute a full adder in near-linear time, reducing the delay approximation factor of 2 from previous approaches to 1.441 for our algorithm.

✉ Sophie Spirkl
sspirkl@princeton.edu

Stephan Held
held@or.uni-bonn.de

1 Research Institute for Discrete Mathematics, University of Bonn, Lennéstr. 2, 53113 Bonn, Germany

2 Program for Applied and Computational Mathematics, Princeton University, Fine Hall, Washington Road, Princeton, NJ 08544, USA

1 Introduction

The addition of binary numbers is one of the most fundamental computational tasks performed by computer chips. Given two binary addends $A = (a_n \ldots a_1)$ and $B = (b_n \ldots b_1)$, where index n denotes the most significant bit, their sum $S = A + B$ has $n + 1$ bits. For each position $1 \leq i \leq n$, we compute a *generate signal* g_i and a *propagate signal* p_i, which are defined as follows:

$$\begin{aligned} g_i &= a_i \wedge b_i, \\ p_i &= a_i \oplus b_i, \end{aligned} \tag{1}$$

where $\wedge$ and $\oplus$ denote the binary AND and XOR functions. The *carry bit* at position $i + 1$ can be computed recursively as $c_{i+1} = g_i \vee (p_i \wedge c_i)$ [4,13]. From the carry bits, we can compute the output S via $s_i = c_i \oplus p_i$ for $1 \leq i \leq n$ and $s_{n+1} = c_{n+1}$.

For two pairs (g_i, p_i) and (g_j, p_j) of generate and propagate signals, we define a binary *prefix operator* as

$$\begin{pmatrix} g_i \\ p_i \end{pmatrix} \circ \begin{pmatrix} g_j \\ p_j \end{pmatrix} = \begin{pmatrix} g_i \vee (p_i \wedge g_j) \\ p_i \wedge p_j \end{pmatrix}. \tag{2}$$

This operator is associative, and it can be used to compute the carry bit c_{i+1} using the identity

$$\begin{pmatrix} c_{i+1} \\ p_i \wedge p_{i-1} \wedge \cdots \wedge p_1 \end{pmatrix} = \begin{pmatrix} g_i \\ p_i \end{pmatrix} \circ \begin{pmatrix} g_{i-1} \\ p_{i-1} \end{pmatrix} \circ \cdots \circ \begin{pmatrix} g_1 \\ p_1 \end{pmatrix}.$$

The prefix operator allows us to simplify notation by combining generate and propagate signals into a single term $z_i = (g_i, p_i)$ and computing c_{i+1} as the first component of $z_i \circ \cdots \circ z_1$. Figure 1 shows a prefix gate computing $z \circ z'$ for the prefix operator in (2) on the left and its underlying logic circuit on the right.

Formally, a *logic circuit* is a non-empty connected acyclic directed graph consisting of nodes that are either *inputs* with at least one outgoing edge and no incoming edges, *outputs* with exactly one incoming edge and no outgoing edges, or *gates* with one or two incoming edges representing one of the 2-bit logical functions AND ($\wedge$), OR ($\vee$), XOR ($\oplus$), NOT and their negations.

The number of gates is the *size* of the circuit. The *(maximum) fan-out of the circuit* is the maximum fan-out (out-degree) of its nodes. The *depth of the circuit* is the maximum number of gates on a directed path.

A logic circuit with inputs $g_1, p_1, \ldots, g_n, p_n$ is called a *prefix carry bit circuit* if it computes c_{n+1} and $p_1 \wedge \cdots \wedge p_n$, it is built from prefix operator gadgets in Fig. 1, and

Fig. 1 Prefix gate and underlying logic circuit
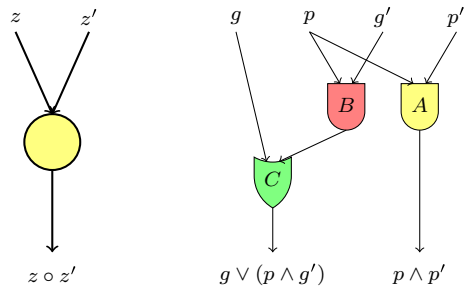
the subcircuit computing $p_1 \wedge \cdots \wedge p_n$ is a tree. Similarly, a *prefix adder* is a logic circuit built using the gadgets in Fig. 1 that computes c_{i+1} and $p_1 \wedge \cdots \wedge p_i$ for all $i = 1, \ldots, n$ at its $2n$ outputs.

A graph that arises from a prefix carry bit circuit by contracting each gadget into a *prefix gate* as in Fig. 1, and by contracting all input pairs (g_i, p_i) into z_i and the output pair $(c_{n+1}, p_n \wedge \cdots \wedge p_1)$, is called a *prefix tree*. Likewise, a *parallel prefix graph* arises from a prefix adder by contracting each gadget, all input pairs $(g_i, p_i) = z_i$ and output pairs $(c_{i+1}, p_1 \wedge \cdots \wedge p_i)$ for all $i = 1, \ldots, n$. For inputs $z_1, \ldots, z_n$, a prefix tree computes the last carry bit of an addition $z_n \circ \cdots \circ z_1$, while a *parallel prefix graph* computes $z_i \circ \cdots \circ z_1$ for all $1 \leq i \leq n$, i.e. all carry bits of an addition.

In a parallel prefix graph every vertex computes a signal of the form $z_{i,j} = z_j \circ z_{j-1} \circ \cdots \circ z_i$ for some $1 \leq i \leq j \leq n$. In accordance with the generate and propagate signals g_k, p_k at the inputs $1 \leq k \leq n$, we call the first component of $z_{i,j}$ the generate signal (sequence) and the second component the propagate signal (sequence) computed at the vertex.

When aiming for a bounded fan-out, we allow the use of repeater gates with fan-in one (a single incoming edge) and fan-out at least two in all types of circuits and graphs.

An example of the transition between parallel prefix graphs and prefix adders is given in Fig. 2. On the left the serial parallel prefix graph with depth 3 is replaced by an AND-OR-path with logic circuit depth 6 known as the ripple-carry adder. For the Kogge–Stone parallel prefix graph [5] on the right, the depth increases from two to four and the maximum fan-out increases from two to three.

Additions are typically not performed as isolated tasks, but the input signals result from preceding computational stages and become available at different fixed *arrival times* $t_i \in \mathbb{N}_0$ ($i \in \{1, \ldots, n\}$), e. g. when used within a multiplier. Here we make the simplifying assumption that g_i and p_i have the same arrival time at the inputs, which is essentially fulfilled if they are generated as in (1). We define the delay of a directed path in a logic circuit starting at an input as its depth plus its input arrival time. The delay of a vertex is the maximum delay of a path ending in the vertex and the *delay of the circuit* is the maximum delay of its outputs. Depth and delay coincide if all input arrival times are zero. Paths and outputs attaining the delay of the circuit are called *critical*. The delay of all vertices can be computed in linear time by a longest path computation in an acyclic network.
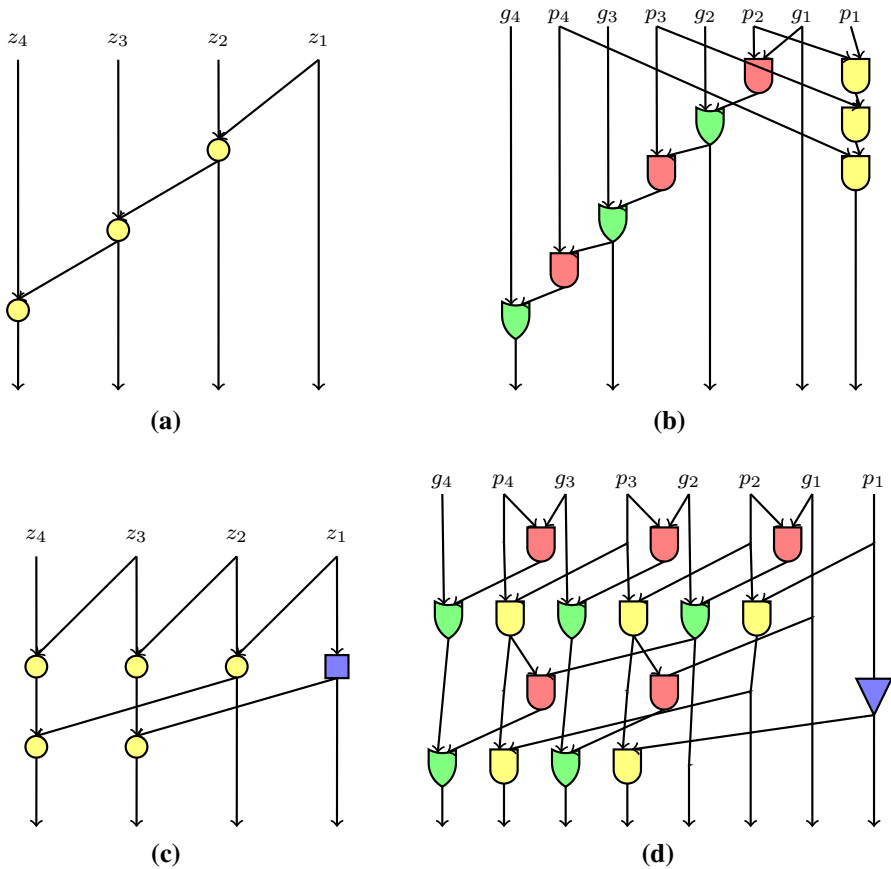
Fig. 2 Prefix graphs as logic circuits. **a** Serial prefix graph. **b** AND- OR-path. **c** Kogge–Stone prefix graph. **d** Kogge–Stone logic circuit

In Fig. 3, we show an example with five inputs and its optimum solutions for different arrival time patterns. Each tree is optimal for neither of the other two arrival times sequences.

We aim for a prefix carry bit circuits and adders with close to minimum delay and small size.

Examples of minimum-depth prefix graphs for uniform input arrival times are the Kogge–Stone graph [5] or the Ladner–Fischer graph [6]. Both have depth $\lceil \log_2 n \rceil$ in terms of prefix gates, but a non-minimal depth of $2\lceil \log_2 n \rceil$ as a logic circuit. For non-uniform arrival times, these circuits might be by a factor of three worse than the lower bound, for example for the arrival time pattern $t_1 = \log_2 n$ and $t_2 = \cdots = t_n = 0$. In Fig. 2c, if z_1 has arrival time two and all other arrival times are zero, the delay of Fig. 2d is 6.

Parallel prefix graphs minimizing the overall prefix graph delay for special input arrival time patterns that occur mostly in certain multipliers were presented by Oklobdzija [7] and Zimmermann [15].
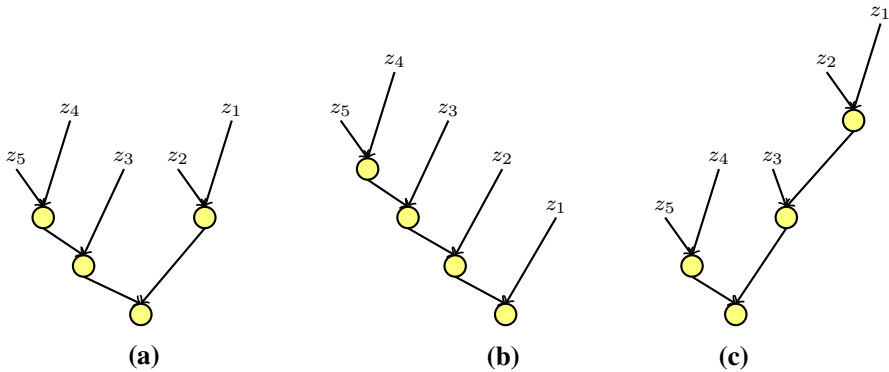
Fig. 3 Different arrival times profiles and their optimum prefix trees. **a** Arrival times 0, 0, 0, 0, 0; delay 4. **b** Arrival times 4, 3, 2, 1, 0; delay 6. **c** Arrival times 0, 1, 2, 3, 4; delay 7

An algorithm for constructing optimum-delay parallel prefix graphs for arbitrary non-uniform input arrival times is given by Choi [1], however this approach may require $\mathcal{O}(n^2)$ gates for a full n-bit adder. Roy et al. [11] enumerate parallel prefix adders with heuristic pruning to achieve good performance-area tradeoffs in practice. In [12] they proposed a variant with polynomial running time. All these approaches minimize the delay of the prefix graph rather than the underlying logic circuit. As the prefix operator contains two subsequent gates, the resulting delay of the underlying circuit may be worse by a factor of two. Variants of the prefix operator have led to improved constructions for small sizes. Examples are the so-called Ling adders or Jackson adders, which were recently compared with other adders by Keeter et al. [3].

As it is common practice in logic synthesis [5,6,11,14], we use a simple technology-independent circuit and delay model in this work. In hardware, the delay of a gate certainly depends on its physical structure. Despite its simplicity, the same model is successfully used in practice for re-optimizing carry bit functions even late in the design flow by Werber et al. [14]. Our lower bounds are based on the definition of delay motivated by the properties of logic circuits; for different computational models, even faster prefix addition (such as by Cole and Vishkin [2]) is possible. In CMOS technology, NAND/NOR gates are faster than AND/OR gates and efficient implementations exist for integrated multi-input AND- OR-Inversion gates and OR- AND-Inversion gates. Therefore, we will also describe techniques for an efficient technology mapping.

1.1 Our Contribution

We will use the delay properties of the prefix operator (2) aiming to minimize the delay of logic circuits for additions instead of the corresponding prefix graphs. This idea was used by [8], who proposed a cubic-time dynamic programming algorithm to compute a fast carry bit circuit.

With a deeper structural analysis of near-optimum prefix trees in Sect. 2, we can construct a carry bit circuit with a better delay bound, size, and running time as shown in the rows with "C" (for carry) as their type (column "T") of Table 1.

Table 1 Improvements over [8,9], where $W = \log_2\left(\sum_{i=1}^{n} 2^{t_i}\right)$ is a lower bound for the delay

	T	Delay	Size	Fan-out	Runtime
[8]	C	$1.441W + 3$	$4n - 3$	$1, 2, 3$	$\mathcal{O}(n^3)$
Here	C	$1.441W + 2.674$	$3n - 3$	$1, 2$	$\mathcal{O}(n \log n)$
[9]	A	$2W + 6\log_2\log_2 n + \mathcal{O}(1)$	$6n\log_2\log_2 n$	$\sqrt{n} + 1$	$\mathcal{O}(n^2)$
Here	A	$1.441W + 5\log_2\log_2 n + 4.5$	$6n\log_2\log_2 n$	$\sqrt{n} + 1$	$\mathcal{O}(n \log n)$

Running times assume constant time for binary addition

In Sect. 3, we apply the carry bit algorithm to substantially improve the delay bound given by Rautenbach et al. [9] for a full n-bit adder with input arrival times $t_1, \ldots, t_n \in \mathbb{N}_0$. The result is listed in the rows with type "A" (for adder) in Table 1. Here, all running times are listed assuming constant-time addition. In Theorem 3, we show that assuming linear-time addition costs only an additional factor of $\mathcal{O}(\log n)$ for all running times. For a comparison with traditional fast adders for uniform input arrival times, recall that adders based on the Kogge–Stone [5] or Ladner–Fischer [6] prefix graphs have delays of $2W$ for uniform input arrival times and $3W$ for arbitrary input arrival times.

Then in Sect. 4, we prove a lower bound on the delay of any prefix carry bit circuit, which shows that our carry bit algorithm is delay-optimal up to an additive constant of 5.

Finally in Sect. 5, we show how to map our new carry bit circuits and full adders to NAND, NOR, and NOT gates economically without increasing the asymptotic delay bound.

2 Algorithm for Single Carry Bit Circuits

We start with a method that given a parallel prefix graph allows us to compute the delay of the underlying logic circuit up to an additive error of one, without inspecting the underlying logic circuit.

Proposition 1 *Given a parallel prefix graph or prefix tree, we propagate the arrival times (which might all be zero) through the prefix gates so that the delay t of a gate with left input (higher indices) l and right input (lower indices) r with delay t_l and t_r, respectively, is defined as $t = \max\{t_r + 2, t_l + 1\}$. Let d be the maximum delay computed with this procedure, maximized over all gates, inputs and outputs, then the delay D of the logic circuit corresponding to the given prefix graph or prefix tree satisfies $d \leq D \leq d + 1$.*

Proof This is a consequence of a longest path computation in acyclic networks. We construct a logic circuit from the prefix graph. For every input pair (g, p) corresponding to an input with arrival time t, we set the arrival time of g and p to be t and $t - 1$, respectively.

We prove by induction that for every signal pair (g, p) corresponding to a signal z at a vertex in the prefix graph, g has delay at least one more than p, we also say that

the pair (g, p) has *skewed arrival times*. Now, if z is not an input, the delay of g is the maximum of two plus the delay of the generate signal of its right predecessor and one plus the delay of the generate signal of its left predecessor. This is clear for inputs. Now consider a signal $z \circ z'$ as in Fig. 1. Let t_g, t_p, $t_{g'}$, and $t_{p'}$ denote the delay of g, p, g' and p', respectively. By induction hypothesis, $t_{p'} + 1 \leq t_{g'}$ and $t_p + 1 \leq t_g$. Therefore, $g \vee (p \wedge g')$ has delay $\max\{t_p + 2, t_{g'} + 2, t_g + 1\} = \max\{t_{g'} + 2, t_g + 1\}$. Furthermore, $\max\{t_p + 2, t_{g'} + 2, t_g + 1\} \geq 1 + \max\{t_p + 1, t_{p'} + 1\}$, which proves that $p' \wedge p$ is indeed by at least one time unit earlier.

Inductive application of the argument above yields that the generate signal of every output arrives at time $\leq d$ under the assumption that all propagate signals of the inputs arrive one time unit earlier than their actual arrival time. Shifting all computed delays up by one time unit yields that the delay of the logic circuit for the actual arrival times is at most $d + 1$.

To show that $d \leq D$, consider only the generate signals, i.e. using the notation of Fig. 1, consider a logic circuit G in which all gates of type A and inputs p_i computing propagate signals are removed, and gates of type B are replaced by repeaters. It follows that for every output c, the subcircuit of G which consists of all ancestors of c is a tree. This is certainly true in the parallel prefix graph, and after the removal of propagate signals, every prefix gate internally corresponds to a tree as well. Removing gates and inputs from a circuit does not increase its delay, because a critical path in G is also a path in the original circuit.

Computing the delay of a signal in a tree is easy: when combining the generate signals of two inputs, one of them has to pass through two gates (a repeater instead of B, and C and the other has to pass through only one (namely C). This shows that the given method for computing d indeed yields a lower bound.   □

For uniform arrival times, $d = D$, but for arbitrary arrival times, $D = d + 1$ is possible, for example by choosing the arrival times of z and z' in Fig. 1 as 1 and 0, respectively.

The prefix graph and its underlying logic circuit can vary greatly in depth and delay. For example, a prefix graph of optimal depth $\lceil \log_2 n \rceil$ as in Fig. 2c contains a balanced binary tree computing its last output, therefore its logic circuit depth is $2 \lceil \log_2 n \rceil$. However, the depth only doubles for the lower (right) input of a prefix gate by Proposition 1, which we exploit in the following.

For a single carry bit computation with arrival times, Rautenbach et al. [8] give a dynamic programming algorithm with cubic running time. The algorithm restructures an AND- OR-path similar to a prefix tree. Here the right-to-left ordering of the leaves of this tree is fixed as $z_1, \ldots, z_n$, because $\circ$ is not commutative. The algorithm recursively splits the sequence of inputs into two parts at an index l attaining the minimum in the recursive delay function

$$\mathcal{D}(t_1, \ldots, t_n) = \min_{l=1,\ldots,n-1} \max\{\mathcal{D}(t_1, \ldots, t_l) + 2, \mathcal{D}(t_{l+1}, \ldots, t_n) + 1\}. \quad (3)$$

This solution can be computed for every subsequence $t_i, t_{i+1}, \ldots, t_j$ of indices via dynamic programming by choosing the $\mathcal{D}$-optimum position l at which to split the sequence, which yields the following result.

Theorem 1 (Rautenbach et al. [8]) *For n input pairs (g_i, p_i) for $1 \leq i \leq n$ with arrival times $t_1, \ldots, t_n \geq 0$, there is a logic circuit computing the carry bit c_{n+1} with*

$$\text{delay}(c_{n+1}) \leq 1.441 \log_2\left(\sum_{i=1}^{n} 2^{t_i}\right) + 3. \tag{4}$$

This circuit can be constructed in $\mathcal{O}(n^3)$ time. It has size at most $4n - 3$, and its maximum fan-out is bounded by two at all gates and bounded by three at all inputs.

Using our definition of a prefix tree, the size of the carry bit circuit can be reduced by n.

Lemma 1 *Any prefix tree computing a single carry bit has an underlying logic circuit size of at most $3n - 3$ and an underlying maximum fan-out of two.*

Proof This is clear as any prefix tree for n inputs has exactly $n - 1$ prefix gates. □

To analyze the structure of fast prefix carry bit circuits we begin with a well-known definition: let F_n be the n-th *Fibonacci number*, where $F_0 = 0$, $F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$. The exact formula for computing the n-th Fibonacci number is $F_n = \frac{1}{\sqrt{5}}(\varphi^n - \psi^n)$, where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden section and $\psi = \frac{1-\sqrt{5}}{2}$.

We first prove a similar delay bound to [8], but instead of bounding the recursive function $\mathcal{D}$, we explicitly construct our solution and obtain useful structural information about it.

Lemma 2 *Let $t_1, \ldots, t_n \in \mathbb{N}_0$ be a sequence of input arrival times for inputs $z_1, \ldots, z_n$, and let F_k be the first Fibonacci number that is at least as large as $\sum_{i=1}^{n}(F_{t_i+3} - 1)$. Then there is a prefix tree computing $z_n \circ \cdots \circ z_1$ with logic gate delay at most k.*

Proof Throughout the proof, we assume that every input signal pair has skewed arrival times, i. e. g_i has arrival time t_i and p_i has arrival time at most $t_i - 1$ for all $1 \leq i \leq n$. By the proof of Proposition 1, all generate and propagate signal pairs in the prefix tree will have skewed arrival times under this assumption. Thus, all prefix gates have depth two for the input with smaller indices and depth one for the input with larger indices, and we will proof that the delay of the circuit is at most $k - 1$. Without the skew assumption, this yields a circuit delay of k and conclude the proof, where we pay a delay of 1 to establish the skew assumption at the inputs.

The proof has two main parts. In the first part, we construct a binary tree T with F_k leaves in such a way that if we consider its internal nodes as prefix gates and its leaves as inputs with arrival time 0, then its overall delay is $k - 1$. During the second step, we replace sections of consecutive leaves and the corresponding subtrees of T with our original inputs so that the arrival time of the input does not exceed the depth of the subtree.

Let T be a tree constructed by starting at the root r and recursively constructing a binary tree with F_{k-1} leaves on the left and one with F_{k-2} leaves on the right as in Fig. 4. We refer to T as a *Fibonacci tree* for k.
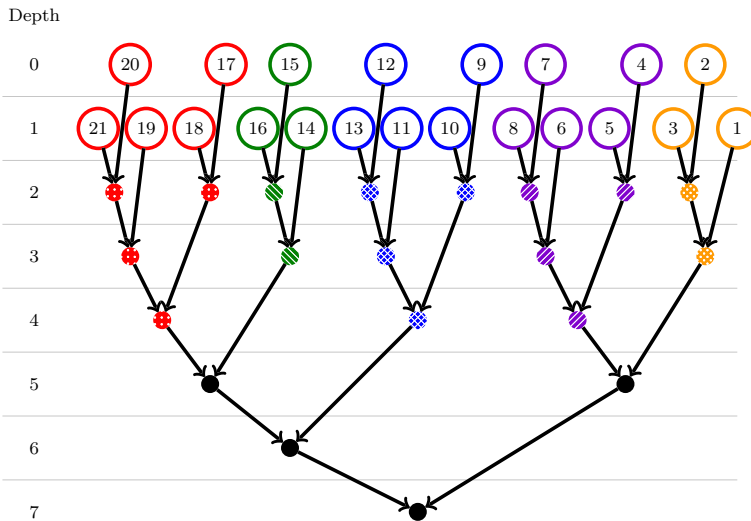
Fig. 4 Fibonacci tree T for $k = 8$

Replacing all non-leaf nodes with prefix gates and leaves with new inputs (with arrival time 0 and unrelated to the original inputs) as well as adding an output at the root yields a prefix tree for F_k inputs with logic gate depth $k - 1$. This can be seen inductively; it is certainly true for $k = 2, 3$ and thus for $k > 3$, the left tree has depth $k - 2$, the right tree has depth $k - 3$, and the last prefix gate has delay $\max\{k - 2 + 1, k - 3 + 2\} = k - 1$. The minimum depth of a prefix tree with l leaves is at most $k - 1$ if and only if $l \le F_k$.

Now we show how to replace parts of the tree by inputs with skewed arrival times $t_1, \ldots, t_n$ without increasing the delay. We start by subdividing the leaves of the tree: from right to left, the first $F_{t_1+3} - 1$ leaves are assigned to the first input, the next $F_{t_2+3} - 1$ leaves are assigned to the second input, and input i gets leaves $1 + \sum_{j=1}^{i-1}(F_{t_j+3} - 1)$ up to $\sum_{j=1}^{i}(F_{t_j+3} - 1)$. Our choice of k ensures that every input i gets $F_{t_i+3} - 1$ successive leaves assigned to it; leftover leaves can be deleted without increasing the delay. The ordering of the inputs is preserved within the tree.

We define a subtree of size l to be a tree obtained by taking a vertex v and all its successors with l leaves in total. By construction, every subtree of size l must be a Fibonacci tree for some j with $F_j = l$. Furthermore, for every F_j with $j \le k - 1$, we can find subtrees of T of size F_j. A vertex v in T is the root of a subtree of size $F_j \ne 1$ if and only if v has depth $j - 1$. For $F_j = 1$, we know that $j \in \{0, 1\}$ and v has depth 0.

Our goal is to show that every input i with arrival time t_i owns all the leaves of a subtree of size F_{t_i+1}. In order to see this, we remove all edges connecting a vertex with depth at most t_i to a vertex with depth more than t_i from the tree. This separates the tree into a connected component containing the root and several subtrees of size at most F_{t_i+1}. For example, if $t_i = 4$, then Fig. 4 would contain the component containing the root as well as subtrees indicated by the coloring and patterns of size 3, 5, 5, 3, 5 in that order. In general, since every gate has depth 1 or 2, each root of such a tree has depth t_i or $t_i - 1$, therefore the subtrees can only have size F_{t_i+1} or F_{t_i}. Our next goal

is to prove that this ordered *subtree sequence* has a special structure. Since only the roots of "big" subtrees of size F_{t_i+1} can be replaced by input i without increasing the delay, we show that there are few small subtrees of size F_{t_i}.

Due to the fact that the depth difference between a node and its left child is always one, the leftmost root in the subtree sequence of a Fibonacci tree for some $k \geq t_i$ has depth t_i and its parent has depth t_{i+1}. Therefore, the subtree rooted here has size F_{t_i+1}. We will now show that in a Fibonacci tree, the ordered subtree sequence of the trees of size F_{t_i+1} and size F_{t_i} never contains two consecutive subtrees of size F_{t_i}. For $k = t_i + 1$, this is clear. For $k = t_i + 2$, there are only two subtrees, and the left one has size F_{t_i+1}. For $k > t_i + 2$, the subtree sequence of a Fibonacci tree for k corresponds to the concatenation of the subtree sequences corresponding to a tree for $k - 1$ and a tree for $k - 2$. As those satisfy the claim by induction hypothesis and each sequence starts with a tree of size F_{t_i+1}, the Fibonacci tree for k has the stated property as well.

We know that input i owns $F_{t_i+3} - 1$ consecutive leaves. In the subtree sequence, at most the first $F_{t_i+1} - 1$ leaves belonging to input i are part of subtrees of which i does not own the first (rightmost) leaf. Of the remaining leaves, the first F_{t_i} might cover a subtree of that size. This accounts for $F_{t_i+1} + F_{t_i} - 1 = F_{t_i+2} - 1$ leaves. The next F_{t_i+1} leaves are owned by i as well, so at that point, at the latest, there must be a subtree of size F_{t_i+1} of which i owns all leaves. For $t_i \neq 1$, we can replace the root of this subtree with one input with arrival time t_i. By construction, this does not increase the delay.

Here we used that $F_{t_i+1} > F_{t_i}$ to give a lower bound of the depth of the owned subtree. The only exception from this is the case $t_i = 1$, which can be treated analogously: every input with $t_i = 1$ owns two leaves, and by similar arguments as for the subtree sequence, one of them must be at depth 1 in the Fibonacci tree.

After removing all leaves that have not been replaced by any original input, we obtain a prefix tree computing $z_n \circ \cdots \circ z_1$ with delay $k - 1$. All of these arguments used the assumption of skewed arrival times also for the inputs, which can be achieved in such a way that the actual delay of the circuit increases to at most k. □

The upper bound of k is tight for the final logic circuit as evident from the example $0, 1, 0$, where $\sum_{i=1}^{3}(F_{t_i+3} - 1) = 1 + 2 + 1 = 4$, so $k = F_k = 5$ (see Fig. 5).
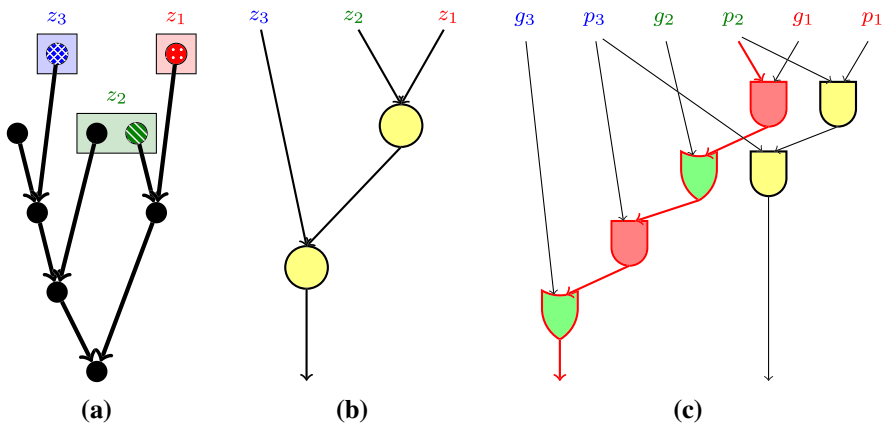

Fig. 5 A tight example. **a** T for $k = 5$. **b** Prefix tree. **c** Logic circuit

For this arrival time profile, the algorithm will (implicitly) construct the Fibonacci tree T and assign leaves to the inputs as in Fig. 5a, where the colored and patterned vertices represent the positions at which the inputs will actually be inserted into the tree. These do not have to be leaves in general. After deleting redundant inputs, we obtain a prefix tree (Fig. 5b) and a corresponding logic circuit (Fig. 5c). Note that p_2 has arrival time 1 and the red path contains four gates, hence the logic circuit has delay 5.

From the proof of Lemma 2, it is easy to see how to avoid the enumeration of all potential splitting positions $l = 1, \ldots, n - 1$ in (3). Since there are F_{k-1} leaves in the left subtree of T and F_{k-2} in the right subtree, let

$$j = \min \left\{ 1 \leq j \leq n : \sum_{i=1}^{j} (F_{t_i+3} - 1) \geq F_{k-2} \right\}$$

and $f = F_{k-2} - \sum_{i=1}^{j-1} (F_{t_i+3} - 1)$, then f counts how many leaves belonging to input j are part of the right subtree, and j is the only input that might have leaves in both subtrees. Since in our decomposition the leftmost F_{t_j+1} leaves of the right subtree belong to a Fibonacci tree of size F_{t_j+1}, j should be on the right side of the decomposition if and only if $f \geq F_{t_j+1}$. Otherwise, there are at least F_{t_j+2} leaves on the left side, hence in our sequence of subtrees j might own all leaves of a subtree of size F_{t_j}, but the remaining leaves must belong to and cover a subtree of size F_{t_j+1}, hence j should be on the left side. Note that it is never optimal to assign all leaves to the same side, thus this partition can always be assumed as proper without increasing the delay. After updating the number of leaves belonging to j on the side it is assigned to, this yields a recursive procedure that terminates when there is only one index left for a subtree.

Lemma 3 *Given input arrival times $t_1, \ldots, t_n \in \mathbb{N}_0$, let F_k be the first Fibonacci number that is at least as large as $\sum_{i=1}^{n} (F_{t_i+3} - 1)$. A prefix tree for these input arrival times with delay at most k can be found with running time $\mathcal{O}(n \log n + k + \max t_i)$ under the assumption that we can perform additions and multiplications by a constant on numbers of arbitrary size in constant time (an assumption we will show how to avoid later). If $t_i \in \mathcal{O}(n)$ for all i, then the running time is $\mathcal{O}(n \log n)$.*

Proof We have already argued that the algorithm achieves the stated delay bound. We show that this partitioning strategy will ensure that every input i is substituted for a subtree of size at least F_{t_i+1}.

If there is only one index i remaining, it was either the rightmost (lowest) or leftmost (highest) index in the previous step. If it was the rightmost index, then the subtree previously contained F_{t_i+2} of its leaves as well as at least one more leaf, hence $k \geq t_i+3$ and the right subtree has size at least F_{t_i+1}, so replacing this subtree by input i leads to the claimed delay by the argument used in Lemma 2. If it was the leftmost index, a similar argument applies.

For the running time estimate, we compute the indices assigned to every leaf and the delay bound k in time $\mathcal{O}(n + k + \max t_i)$. There are $n - 1$ recursive partitioning
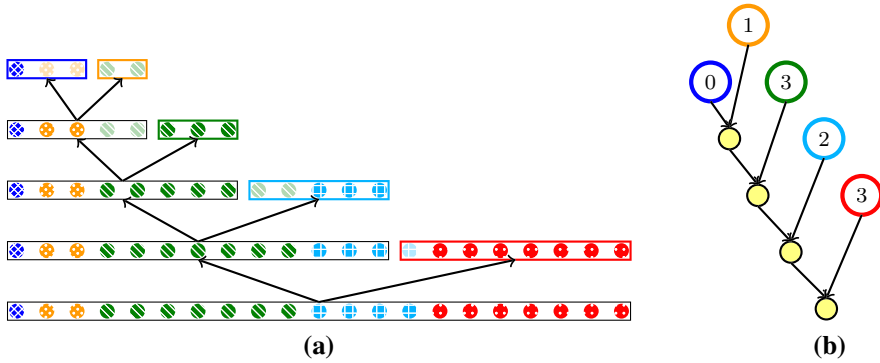
Fig. 6 Example of the algorithm. **a** Output for input arrival times 3, 2, 3, 1, 0. **b** Prefix tree

steps, during each of which we find the input j as the input index to own leaves in the left subtree. This can be done in logarithmic time using binary search in the sorted array of the indices of the first leaf belonging to every input.   □

Figure 6a shows how the algorithm works for the sequence of input arrival times 3, 2, 3, 1, 0. The number of leaves we need is $7 + 4 + 7 + 2 + 1 = 21$, therefore $k = 8$ suffices. We number the leaves from right to left as $1, \ldots, 21$. After the first split, 3 light blue leaves of input 2 are in the left subtree, hence the corresponding input is assigned to the left subtree. Note that for the orange leaves of input 4, we end up assigning them to a right subtree that does not contain any orange leaves in the beginning, but only green leaves of input 3, in order to ensure a proper partition. We obtain the result shown in Fig. 6b.

Lemma 4 *We can construct a prefix tree with a delay bound as in* Lemma 2 *for any instance* $(t_1, \ldots, t_n)$ *by instead constructing a prefix tree for an instance* $(t'_1, \ldots, t'_n)$ *with* $\max t'_i \leq 2n - 1$.

This follows from the fact that the longest path from any input to the output contains at most $n - 1$ prefix gates. The maximum delay difference can be assumed as $2n - 2$, since any input with earlier arrival time will never be critical.

Theorem 2 *For n inputs with arrival times $t_1, \ldots, t_n \in \mathbb{N}_0$, the algorithm finds a prefix carry bit circuit for c_{n+1} with*

$$\text{delay}(c_{n+1}) \leq k \leq \left\lfloor \log_\varphi \left(\sum_{i=1}^n \varphi^{t_i} \right) \right\rfloor + 4.$$

The constructed logic circuit has size at most $3n - 3$ and maximum fan-out two at all logic gates and inputs. Furthermore, the delay is at most

$$k \leq \log_\varphi \left(\sum_{i=1}^n 2^{t_i} \right) + 2.673 \leq 1.441 \log_2 \left(\sum_{i=1}^n 2^{t_i} \right) + 2.673.$$

Proof The size and fan-out bounds follow from Lemma 1. The delay of the constructed circuit is k. By choice of k, we know that $\sum_{i=1}^{n}(F_{t_i+3}-1) \geq F_{k-1}+1$. With $\varphi = \frac{1+\sqrt{5}}{2}$, $\psi = \frac{1-\sqrt{5}}{2}$ and the exact formula $F_n = \frac{1}{\sqrt{5}} \cdot (\varphi^n - \psi^n)$, it follows that $|\sqrt{5}F_n - \varphi^n| \leq 1$ and for $n \geq 1$, $|\sqrt{5}F_n - \varphi^n| \leq |\psi|$.

Now $k - 1 = 0$ can only be true if there is only one input. In this case, the stated delay bound is trivially true. Otherwise, we obtain the estimate:

$$
\begin{aligned}
k - 1 = \log_\varphi\left(\varphi^{k-1}\right) &\leq \log_\varphi\left(\sqrt{5}\,(F_{k-1}+1)\right) \\
&\leq \log_\varphi\left(\sqrt{5}\left(\sum_{i=1}^{n}(F_{t_i+3}-1)\right)\right) \\
&\leq \log_\varphi\left(\sqrt{5}\left(\sum_{i=1}^{n}\left(\frac{1}{\sqrt{5}}\left(\varphi^{t_i+3}+1\right)-1\right)\right)\right) \\
&\leq \log_\varphi\left(\sum_{i=1}^{n}\varphi^{t_i+3}\right) = \log_\varphi\left(\sum_{i=1}^{n}\varphi^{t_i}\right) + 3,
\end{aligned}
$$

which proves the first claim.

For a single input, the second delay bound is trivially true. Furthermore, for $t_i \geq 0$, $F_{t_i+3} - 1 \leq 2^{t_i}$. We obtain the estimate:

$$
\begin{aligned}
k - 1 = \log_\varphi\left(\varphi^{k-1}\right) &\leq \log_\varphi\left(\sqrt{5}\,(F_{k-1}+1)\right) \\
&\leq \log_\varphi\left(\sqrt{5}\left(\sum_{i=1}^{n}(F_{t_i+3}-1)\right)\right) \\
&\leq \log_\varphi\left(\sqrt{5}\left(\sum_{i=1}^{n}2^{t_i}\right)\right) \\
&= \log_\varphi\left(\sum_{i=1}^{n}2^{t_i}\right) + \log_\varphi\sqrt{5} \leq 1.441\log_2\left(\sum_{i=1}^{n}2^{t_i}\right) + 1.673.
\end{aligned}
$$

□

Our proof allows an improvement over the delay bound of [8] due to a refined analysis. A running time of $\mathcal{O}(n\log n)$ follows from Lemma 3 assuming that we can add numbers of linear size and multiply them by a constant in constant time. Under the more practical assumption that these operations take linear time with respect to the number of digits, the algorithm has super-quadratic running time, which can be avoided as follows:

Theorem 3 *For any fixed $\gamma > 1$, a prefix carry bit circuit as in the setting of* Theorem 2 *with*

$$
\text{delay}(c_{n+1}) \leq \log_\varphi\left(\sum_{i=1}^{n}\varphi^{t_i}\right) + 4 + 2.1 \cdot n^{1-\gamma}
$$

can be found in $\mathcal{O}(n\gamma \log^2 n)$ time assuming linear-time addition and multiplication with constants. It satisfies

$$\text{delay}(c_{n+1}) \leq 1.441 \log_2\left(\sum_{i=1}^{n} 2^{t_i}\right) + 2.673 + 2.1 \cdot n^{1-1.4\gamma}.$$

Proof By Lemma 4 and Theorem 2, we can solve instances with $\max t_i - \min t_i \leq \gamma \lceil \log_\varphi n \rceil$ in $\mathcal{O}(n\gamma \log^2 n)$ time with linear-time addition.

Given an instance with arrival times $t_1, \ldots, t_n \in \mathbb{N}_{\geq 0}$, we define a new instance $t_i' = \max \left\{ t_i, \max_{j \in \{1,\ldots,n\}} t_j - \gamma \lceil \log_\varphi n \rceil \right\}$ and compute a circuit for the modified instance in $\mathcal{O}(n\gamma \log^2 n)$. When reverting to the original arrival times, the delay of this solution does not increase, because none of the arrival times do. Therefore,

$$
\begin{aligned}
\text{delay}(c_{n+1}) - 4 &\leq \log_\varphi\left(\sum_{i=1}^{n} \phi^{t_i'}\right) \\
&\leq \log_\varphi\left(n \cdot \phi^{\max t_i - \gamma \lceil \log_\varphi n \rceil} + \sum_{i=1}^{n} \phi^{t_i}\right) \\
&\leq \log_\varphi\left(\phi^{\max t_i + (1-\gamma)\log_\varphi n} + \sum_{i=1}^{n} \phi^{t_i}\right) \\
&\leq \log_\varphi\left(\sum_{i=1}^{n} \phi^{t_i}\right) + \log_\varphi\left(1 + \phi^{(1-\gamma)\log_\varphi n}\right) \\
&\leq \log_\varphi\left(\sum_{i=1}^{n} \phi^{t_i}\right) + \log_\varphi e \cdot n^{1-\gamma},
\end{aligned}
$$

and $\log_\varphi e < 2.1$. For the dual logarithm-based delay bound, we have

$$
\begin{aligned}
\text{delay}(c_{n+1}) - 2.673 &\leq \log_\varphi 2 \cdot \log_2\left(\sum_{i=1}^{n} 2^{t_i'}\right) \\
&\leq \log_\varphi 2 \cdot \log_2\left(n \cdot 2^{\max t_i - \gamma \lceil \log_\varphi n \rceil} + \sum_{i=1}^{n} 2^{t_i}\right) \\
&\leq \log_\varphi 2 \cdot \log_2\left(2^{\max t_i - (1-\gamma \log_\varphi 2)\log_2 n} + \sum_{i=1}^{n} 2^{t_i}\right) \\
&\leq \log_\varphi 2\left(\log_2\left(\sum_{i=1}^{n} 2^{t_i}\right) + \log_2\left(1 + 2^{(1-\gamma \log_\varphi 2)\log_2 n}\right)\right) \\
&\leq \log_\varphi 2 \cdot \log_2\left(\sum_{i=1}^{n} 2^{t_i}\right) + (\log_\varphi e)n^{1-\gamma \log_\varphi 2},
\end{aligned}
$$

and $1.4 < \log_\varphi 2 < 1.441$. $\square$

For $\gamma > 1$, the additional error decreases with growing n. Since the algorithm is only useful if $n \geq 2$, choosing a sufficiently large constant γ yields the delay bound $1.441 \log_2 \left(\sum_{i=1}^{n} 2^{t_i} \right) + 2.674$ with running time $\mathcal{O}(n \log^2 n)$.

3 Algorithm for Prefix Adder Circuits

The naïve parallel prefix graph construction, in which all carry bits are computed separately by a carry bit circuit, might contain a quadratic number of gates. Therefore, Rautenbach et al. also developed a parallel prefix graph construction computing all carry bits [9].

Theorem 4 (Rautenbach et al. [9]) *Given arrival times $t_1, \ldots, t_n \in \mathbb{N}_0$, there is a parallel prefix graph for n inputs of size $\mathcal{O}(n \log \log n)$ with logic delay $2 \log_2 \left(\sum_{i=1}^{n} 2^{t_i} \right) + 6 \log_2 \log_2 n + \mathcal{O}(1)$.*

The primary objective in [9] is to minimize the delay of the prefix graph instead of the underlying logic circuit. We will improve the performance guarantee for a similar construction as in [9] by using a carry bit circuit as in Sect. 2 as a subroutine.

Given n inputs, we partition the set $\{1, \ldots, n\}$ into $l = \lceil \sqrt{n} \rceil$ subsets $V_1, \ldots, V_l$, each containing l or $l-1$ consecutive indices. Let $Z_i = \circ_{j \in V_i} z_j$, where Z_i is computed by a circuit constructed by the carry bit algorithm. This is shown in green, labeled "Best", in Fig. 7. The parallel prefix graph construction is applied recursively to compute prefixes for all groups without their highest index as well as for the $l-1$ inputs $Z_1, \ldots, Z_{l-1}$ (which corresponds to the red boxes labeled "Recursion" in Fig. 7), i.e. we build $l+1$ parallel prefix graphs, each with at most $l-1$ inputs. As a final step, we combine all prefixes from group i with the $(i-1)$-th prefix of the Z_i and add one more prefix gate combining Z_l with the $(l-1)$-th prefix of the Z_i. This yields a parallel prefix graph.

The following two lemmas analyze the size of the resulting parallel prefix graph and the running time of its construction.

Lemma 5 *The parallel prefix graph in [9] and the modified construction above have the same size; for $n \geq 3$, it is bounded by $2n \log_2 \log_2 n$ in terms of prefix gates and $6n \log_2 \log_2 n$ in terms of logic gates.*

Proof Consider Fig. 8 and proceed by induction on the number of inputs. On a level with n inputs, the number of prefix gates in all the green carry bit circuits (labeled
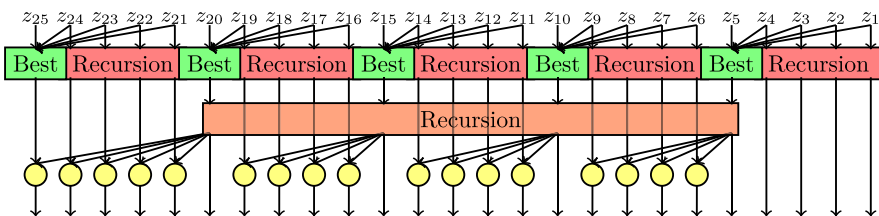

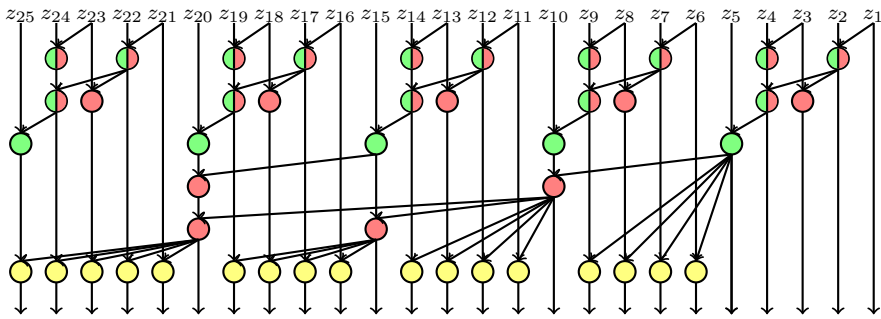
Fig. 7 Prefix graph construction

Fig. 8 Parallel prefix graph for uniform arrival times

"Best") and the number of yellow prefix gates in the bottom row are both at most n. The total number of inputs of recursion blocks can be bounded by n as well: if there are l groups, then $n - l$ original inputs are inputs of recursion blocks; one further recursion block has $l - 1$ inputs. For small n, the correctness follows from Fig. 8, e. g. for $n = 3$, the size bound is 5 and 3 gates are required.

Let $V_1, \ldots, V_l$ be the groups, $l \geq 3$, then by induction hypothesis, the prefix gate size is bounded by

$$2n + 2(l-1)\log_2\log_2(l-1) + \sum_{i=1}^{l} 2(|V_i| - 1)\log_2\log_2(|V_i| - 1)$$

$$\leq 2n + \sum_{i=1}^{l} 2|V_i|\log_2\log_2(l-1)$$

$$\leq 2n + 2n\log_2\log_2\sqrt{n} = 2n\log_2\log_2 n.$$

For logic gates, the size increases by a factor of three.   □

Lemma 6 *The parallel prefix graph above can be computed in $\mathcal{O}(n\log^2 n)$ time.*

Proof As in Theorem 3, we round all running times up to at least $\max t_i - \gamma\lceil \log_2 n\rceil$ for a fixed $\gamma > 1$. For this arrival time profile, we have already shown that

$$1.441\left(\sum_{i=1}^{n} 2^{t_i'}\right) \leq 1.441\log_2\left(\sum_{i=1}^{n} 2^{t_i}\right) + 2.1 \cdot n^{1-1.4\gamma}.$$

Therefore, we can use the rounded arrival time profile to achieve a delay guarantee of

$$1.441\log_2\left(\sum_{i=1}^{n} 2^{t_i}\right) + 5\log_2\log_2 n + 4.5$$

for $\gamma = 3$. This means that all numbers in the computations have size at most $\mathcal{O}(\log n)$, thus it remains to bound the number of operations by $\mathcal{O}(n\log n)$.

For each level $l = 1, \ldots, \log_2 \log_2 n$ of the recursion, we have a partition of the n inputs into groups, where the maximum group size is bounded by $n^{1/2^l}$. Therefore, the prefix trees for the Z_i can be computed in $\mathcal{O}\left(n \log\left(n^{1/2^l}\right)\right) = \mathcal{O}\left(\left(\frac{1}{2}\right)^l n \log n\right)$ time. All Z_i require time $\mathcal{O}(n \log n)$, because this is a geometric series. All remaining gates are prefix gates; they have fixed positions, thus each of them requires only constant time to compute, and there are $\mathcal{O}(n \log \log n)$ such gates in total. □

The new parallel prefix graph construction is summarized in the following theorem.

Theorem 5 *Given $n \in \mathbb{N}$ and arrival times $t_1, \ldots, t_n \in \mathbb{N}_0$, our algorithm finds a parallel prefix graph with logic gate delay at most*

$$\log_\varphi\left(\sum_{i=1}^n \varphi^{t_i}\right) + 5 \log_2 \log_2 n + 4.5 \leq 1.441 \log_2\left(\sum_{i=1}^n 2^{t_i}\right) + 5 \log_2 \log_2 n + 4.5.$$

It can be implemented with running time $\mathcal{O}(n \log^2 n)$ and the computed circuit has size at most $6n \log_2 \log_2 n$ in terms of logic gates.

This theorem implies that if the number n of inputs is sufficiently large, we have a 1.441-approximation algorithm in terms of the delay for a prefix adder. The algorithm of [9] has a running time of $\Omega(n^2)$, which the use of our carry bit algorithm improves to a near-linear running time, even with linear-time addition.

To prove the delay bound, we assume that all arrival times are skewed by one time unit. Under this assumption, let $w = \sum_{i=1}^n \varphi^{t_i}$, and let $\mathrm{delay}(w, n)$ denote the maximum delay for a circuit constructed as above with $n \geq 3$ inputs and an arrival time profile leading to the same w. Then $\mathrm{delay}(w, n) + 1$ is an upper bound on the delay of the constructed circuit, and we have:

Lemma 7 *For n input pairs with skewed arrival times $t_1, \ldots, t_n$, let $w = \sum_{i=1}^n \varphi^{t_i}$. Then we have*

$$\mathrm{delay}(w, n) \leq \log_\varphi w + 5 \log_2 \log_2 n + 3.$$

Proof We may assume that the given arrival time profile achieves the maximum delay, i.e. for $t_1, \ldots, t_n$, the construction has a delay of $\mathrm{delay}(w, n)$.

By Theorem 2 and using the assumption that the propagate signals arrive earlier than the generate signals, we can compute Z_i with a delay of at most $\log_\varphi\left(\sum_{j \in V_i} \varphi^{t_j}\right) + 3$. Therefore, their prefix graph has delay at most

$$\mathrm{delay}\left(\sum_{i=1}^l \varphi^{\log_\varphi\left(\sum_{j \in V_i} \varphi^{t_j}\right)+3}, \left\lceil \sqrt{n} \right\rceil - 1\right) = \mathrm{delay}\left(\varphi^3 \cdot \sum_{j=1}^n \varphi^{t_j}, \left\lceil \sqrt{n} \right\rceil - 1\right).$$

For each of the groups V_i containing $\left\lceil \sqrt{n} \right\rceil$ or $\left\lceil \sqrt{n} \right\rceil - 1$ inputs, the prefix graph of all but its last input (highest index) has delay at most

$$\mathrm{delay}(w, \left\lceil \sqrt{n} \right\rceil - 1) \leq \mathrm{delay}(\varphi^3 w, \left\lceil \sqrt{n} \right\rceil - 1),$$

as delay(w, n) is monotonically increasing in w and n. Therefore, the combination of a prefix of one of the V_i and the corresponding prefix of the Z_i has logic gate delay at most delay$(w, n) \leq$ delay $\left(\varphi^3 w, \lceil \sqrt{n} \rceil - 1\right) + 2$.

We prove the absolute delay estimate by induction on n. For $n \leq 3$, delay$(w, n) - \log_\varphi w \leq 3$ for all input sequences with this parameter w as $\log_\varphi w \geq \max_i t_i$. Therefore, for $n \geq 4$, delay(w, n) is bounded by

$$
\begin{aligned}
\text{delay}(\varphi^3 w, \lceil \sqrt{n} \rceil - 1) + 2 &\leq \log_\varphi(\varphi^3 w) + 5 \log_2 \log_2(\sqrt{n}) + 5 \\
&\leq \log_\varphi w + 5 \log_2(0.5 \log_2 n) + 8 \\
&= \log_\varphi w + 5 \log_2 \log_2 n - 5 + 8 \\
&= \log_\varphi w + 5 \log_2 \log_2 n + 3.
\end{aligned}
$$

$\square$

Without assuming skewed arrival times, this construction has a delay bound of $\log_\varphi w + 5 \log_2 \log_2 n + 4$. For $n = 25$, an example is shown in Fig. 8. Gates are colored by the part of the recursion they represent; in this special case, some gates can be used to compute the Z_i as well as the group prefixes, hence they are colored both red (right half) and green (left half).

The construction in [9] and our variant of it both have a very high fan-out; for n inputs, the fan-out is at least $\lceil \sqrt{n} \rceil$. In a physical implementation such a high fanout induces a significant delay and requires the insertion of duplicate gates into the interconnect to repeat the signals. The high fan-outs occur precisely at the Z_i-prefixes, therefore they accumulate on a critical path. For n inputs, the fan-out can be redistributed to duplicate gates with fan-out 2 using depth $\frac{1}{2} \lceil \log_2 n \rceil + 1$; this will lead to an overall increase in delay of $\lceil \log_2 n \rceil + \mathcal{O}(\log_2 \log_2 n)$ for a given path [9]. Therefore, we obtain a 2.441-approximation algorithm if the fan-out is bounded by 2, improving the 3-approximation achieved by [9] in this scenario. A 3-approximation is also obtained by the Kogge–Stone adder [5] for arbitrary input arrival times. However, it comes with a larger size of $\mathcal{O}(n \log_2 n)$.

4 A Lower Bound for Prefix Adders

Proposition 1 shows that a lower bound for the delay of a prefix tree for a single carry bit is given by an optimal binary tree with depth one for the left child and depth two for the right child in which the leaves represent inputs and their right-to-left order corresponds to the ordering of the inputs. For zero arrival times, this is achieved by a Fibonacci tree. Rautenbach et al. [10] observed that this a special case of a more general concept: alphabetic code trees with unequal letter costs. These can be used to obtain general lower bounds, which we improve and state explicitly by using the specific properties of our application.

Lemma 8 *Given n inputs with integral arrival times $t_1, \ldots, t_n \in \mathbb{N}_0$, a prefix tree computing their carry bit c_{n+1} has logic gate delay at least*

$$\text{delay}(c_{n+1}) \geq \log_\varphi \left(\sum_{i=1}^{n} \varphi^{t_i} \right) - 1.$$

Proof In Sect. 2 we saw that F_{t_i+1} inputs with arrival time zero can be combined with depth t_i. Therefore, an optimal prefix tree for inputs with arrival times $t_1, \ldots, t_n$ of delay k can be restructured into a prefix tree with $\sum_{i=1}^{n} F_{t_i+1}$ inputs with depth k by replacing input i by a Fibonacci tree for $t_i + 1$. If there is only one input, the lemma is trivially true, thus we may assume $\sum_{i=1}^{n} F_{t_i+1} \geq 2$. But a tree of depth k has at most F_{k+1} leaves, hence $k \geq 2$ and

$$
\begin{aligned}
k + 1 = \log_\varphi \left(\varphi^{k+1} \right) &\geq \log_\varphi \left(\sqrt{5} \left(F_{k+1} + \frac{\psi^3}{\sqrt{5}} \right) \right) \\
&\geq \log_\varphi \left(\sqrt{5} \left(\sum_{i=1}^{n} F_{t_i+1} + \frac{\psi^3}{\sqrt{5}} \right) \right) \\
&\geq \log_\varphi \left(\sum_{i=1}^{n} \left(\varphi^{t_i+1} - \psi^2 + \psi^3 \right) \right) \\
&= \log_\varphi \left(\sum_{i=1}^{n} \left(\varphi^{t_i+1} - \varphi\psi^2 \right) \right) \\
&\geq \log_\varphi \left(\varphi(1 - \psi^2) \cdot \sum_{i=1}^{n} \varphi^{t_i} \right) = \log_\varphi \left(\sum_{i=1}^{n} \varphi^{t_i} \right),
\end{aligned}
$$

and $k \geq \log_\varphi \left(\sum_{i=1}^{n} \varphi^{t_i} \right) - 1$ as claimed. $\qquad \square$

This lemma shows that the single carry bit circuits in Sect. 2 have optimum delay up to an additive margin of 5.

5 Technology Mapping

In CMOS technology, AND and OR gates are significantly slower than their negated counterparts. Therefore, in the following we will transform the constructed logic circuits to use only NAND and NOR gates without increasing their delay too much.

To this end, we introduce two new prefix gates, $\overline{\wedge}$ and $\overline{\vee}$ shown in Fig. 9a, b. Much like the non-inverted prefix gate in Fig. 1, they take as input two pairs of generate and a propagate signals, and compute as output their combined generate and propagate signal. Unlike ∘ gates, these gates assume that exactly one g and p is inverted, and the same is true for the output generate and propagate signal.

The new prefix gates also preserve the property that the delay of the gate is one for the left input and two for the right input assuming skewed arrival times as before. Therefore, replacing non-inverted prefix gates with $\overline{\wedge}$ and $\overline{\vee}$ almost preserves the delay, as the following lemma shows.
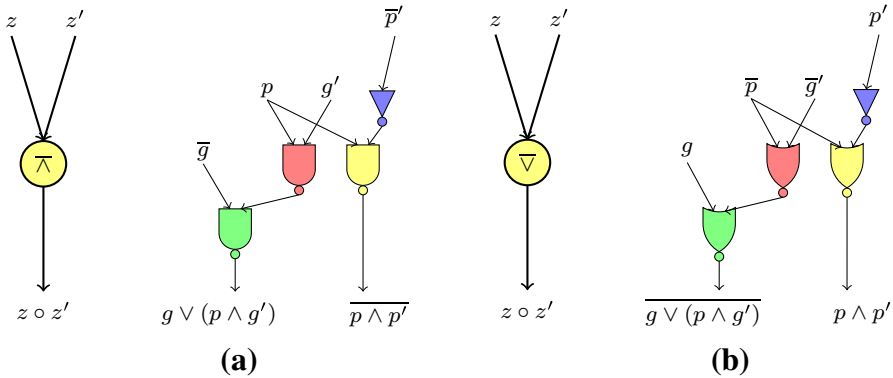
Fig. 9 Inverting prefix gates. **a** A prefix gate using only Nand and Not. **b** A prefix gate using only Nor and Not

Lemma 9 *A non-inverted prefix gate tree computing a single carry bit can transformed into a prefix tree of $\overline{\wedge}$ and $\overline{\vee}$ gates without increasing the delay by more than 1. The transformed circuit has size $5n - 4$.*

Proof We begin at the root of the tree, where we replace the prefix gate by a $\overline{\wedge}$ gate, thus ensuring that the output carry bit (generate signal) is non-inverted. For the predecessors we proceed as follows. The right child of an inverted prefix gate requires the same inversion for g and p as the output pair, so the right child of a $\overline{\wedge}$ gate should be a $\overline{\wedge}$ gate, and the right child of a $\overline{\vee}$ gate should be a $\overline{\vee}$ gate. The left child requires the reverse inversion of g and p, so the left child of a $\overline{\vee}$ should be a $\overline{\wedge}$ and vice versa. Making these replacements as needed is possible since the original prefix circuit is a tree in terms of prefix gates. Since the delay of the non-inverted and inverted prefix gates are the same, so is the delay of the resulting tree. However, since each pair input signals (g_i, p_i) will be used with exactly one of the components inverted, each input has to use an additional inverter, thus increasing the overall delay by one. The size is the sum of $4(n - 1)$ logic gates in the $n - 1$ $\overline{\vee}$ and $\overline{\wedge}$ gates, and n inverters at the inputs.   □

Similarly, we can map the non-inverted full adder from Fig. 7 to Nand, Nor-, and Not gates, by applying Lemma 9 to the prefix carry bit circuits labeled "Best".

Lemma 10 *Given $n \in \mathbb{N}$ and arrival times $t_1, \ldots, t_n \in \mathbb{N}_0$, we can compute a full adder (computing all carry bits) using only Nand, Nor and Not gates with delay at most*

$$1.441 \log_2 \left(\sum_{i=1}^{n} 2^{t_i} \right) + 6 \lceil \log_2 \log_2 n \rceil + 6.5$$

and size $\mathcal{O}(n \log_2 \log_2 n)$.

Proof We require an additional delay of one to ensure that in the beginning all signals are available inverted and non-inverted as needed, and an additional delay of one at the outputs as well. The recursion has $\lceil \log_2 \log_2 n \rceil$ levels. For each level, we need to correct inversions at most once, after the first level of "Best" and "Recursion" computations, thus also supplying each input signal of the next level of the recursion in inverted and non-inverted form. Depending on how the outputs are inverted after the recursion is applied to the carry bits of the groups as computed by "Best", the prefix gates in Fig. 7 are replaced by $\widehat{\wedge}$ or $\widehat{\vee}$ gates. After correcting the inversion at the outputs, the resulting circuit computes all carry bits with an additional delay of $\log_2 \log_2 n + 2$ compared to the non-inverted construction. The additional number of gates is at most linear at every level, and thus $\mathcal{O}(n \log_2 \log_2 n)$ overall. □

This shows how the given constructions that were formulated in terms of non-inverted prefix gates can be transformed into circuits using only inverting gates, which are more useful for practical applications. The transformations do not increase the asymptotic delays. For the carry bit circuit the delay increases just by one. Thereby, the size bounds grow by a small constant factor.

Compliance with Ethical Standards

References

1. Choi, Y.: Parallel Prefix Adder Design. Dissertation, University of Texas at Austin (2004)
2. Cole, R., Vishkin, U.: Faster optimal parallel prefix sums and list ranking. Inf. Comput. **81**(3), 334–352 (1989)
3. Keeter, M., Harris, D.M., Macrae, A., Glick, R., Ong, M., Schauer, J.: Implementation of 32-bit Ling and Jackson adders. In: Proceedings of the Forty Fifth Asilomar Conference on Signals, Systems and Computers, pp. 170–175 (2011)
4. Knowles, S.: A family of adders. In: Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH-15), pp. 277–281 (2001)
5. Kogge, P.M., Stone, H.S.: A parallel algorithm for the efficient solution of a general class of recurrence equations. IEEE Trans. Comput. **100**(8), 786–793 (1973)
6. Ladner, R.E., Fischer, M.J.: Parallel prefix computation. J. ACM **27**(4), 831–838 (1980)
7. Oklobdzija, V.G.: Design and analysis of fast carry-propagate adder under non-equal input signal arrival profile. In: Proceedings of the Twenty-Eighth Asilomar Conference on Signals, Systems and Computers, vol. 2, pp. 1398–1401 (1994)
8. Rautenbach, D., Szegedy, C., Werber, J.: Delay optimization of linear depth Boolean circuits with prescribed input arrival times. J. Discrete Algorithms **4**(4), 526–537 (2006)
9. Rautenbach, D., Szegedy, C., Werber, J.: The delay of circuits whose inputs have specified arrival times. Discrete Appl. Math. **155**(10), 1233–1243 (2007)
10. Rautenbach, D., Szegedy, C., Werber, J.: On the cost of optimal alphabetic code trees with unequal letter costs. Eur. J. Comb. **29**(2), 386–394 (2008)
11. Roy, S., Choudhury, M., Puri, R., Pan, D.Z.: Towards optimal performance-area trade-off in adders by synthesis of parallel prefix structures. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **33**(10), 1517–1530 (2014)
12. Roy, S., Choudhury, M., Puri, R., Pan, D.Z.: Polynomial time algorithm for area and power efficient adder synthesis in high-performance designs. In: Proceedings of the 20th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 249–254 (2015)
13. Weinberger, A., Smith, J.L.: A logic for high-speed addition. Natl. Bur. Stand. Circul. **591**, 3–12 (1958)

14. Werber, J., Rautenbach, D., Szegedy, C.: Timing optimization by restructuring long combinatorial paths. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 536–543 (2007)
15. Zimmermann, R.: Binary Adder Architectures for Cell-Based VLSI and Their Synthesis. Dissertation, Swiss Federal Institute of Technology (ETH) in Zurich (1998)