Valiant's Universal Circuits Revisited: an Overall Improvement and a Lower Bound

Shuoyao Zhao[1,3], Yu Yu[1], and Jiang Zhang[2]

[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
[2]State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
[3]PlatON CO., Limited

Abstract

A universal circuit (UC) is a general-purpose circuit that can simulate arbitrary circuits (up to a certain size n). At STOC 1976 Valiant presented a graph theoretic approach to the construction of UCs, where a UC is represented by an edge universal graph (EUG) and is recursively constructed using a dedicated graph object (referred to as supernode). As a main end result, Valiant constructed a 4-way supernode of size 19 and an EUG of size $4.75n \log n$ (omitting smaller terms), which remained the most size-efficient even to this day (after more than 4 decades).

Motivated by the emerging applications of UCs in various privacy preserving computation scenarios, we revisit Valiant's universal circuits, and propose a 4-way supernode of size 18, and an EUG of size $4.5n \log n$. As confirmed by our implementations, we reduce the size of universal circuits (and the number of AND gates) by more than 5% in general , and thus improve upon the efficiency of UC-based cryptographic applications accordingly. Our approach to the design of optimal supernodes is computer aided (rather than by hand as in previous works), which might be of independent interest. As a complement, we give lower bounds on the size of EUGs and UCs in Valiant's framework, which significantly improves upon the generic lower bound on UC size and therefore reduces the gap between theory and practice of universal circuits.

Universal Circuits Private Function Evaluation Multiparty Computation.

1 Introduction

A universal circuit (UC)[1] refers to a circuit that can be programmed to simulate any Boolean circuit C up to a given size. That is, a UC takes as input program bits p_{C} (that encodes C) in addition to an input x, and produces as output $\mathsf{UC}(x, p_{\mathsf{C}}) = \mathsf{C}(x)$. This is analogous to a central processing unit (CPU) that carries out the computations specified by the instructions of a computer program.

1.1 Applications of Universal Circuits

Universal circuits have received sustained research interests and have been found useful in various privacy-preserving computation applications. We recall a few below, whose efficiency would benefit from the improvement of universal circuits.

[1] As a slight abuse of abbreviation, we use UC as the shorthand for universal circuit, and the readers should not confuse it with universal composability.

1.1.1 Program Obfuscation

Garg et al. [GGH$^+$16] used UCs to construct universal branching programs which was in turn used to build a candidate indistinguishability obfuscation (iO). More recently Zimmerman [Zim15] proposed an approach to obfuscation by viewing UC as a keyed program for circuit families.

1.1.2 Private Function Evaluation

Universal circuits are an essential tool to transform a multi-party computation (MPC) protocol into one for private function evaluation (PFE). UC-based PFE was studied in [KS08b] and was later improved and extended in [LMS16, BBKL17]. A general framework for PFE protocols that allows for instantiations from various concrete protocols in different settings was proposed in [MS13] and was then extended to malicious adversary setting in [MSS14]. Furthermore, the actively secure non-interactive secure computation (NISC) technique [AMPR14] can be applied to UC to realize actively secure non-interactive PFE, which is beyond the reach of the framework of [MS13, MSS14].

1.1.3 Batched Execution of 2PC

Another interesting application of UC is efficient batch execution for secure two-party computation (2PC). The batch execution techniques [HKK$^+$14, LR15] were originally intended for amortizing the cost of maliciously secure garbled circuits for the same function, and UCs can now enable batched execution for circuits of different functions (realized by the same UC).

1.1.4 Universal Models of Computation

Valiant's UCs motivated the design of universal parallel computers [GP81, Mey83]. Both depth-optimized [CH85] and size-optimized [Val76] approaches to UCs were adapted in [BFGH10] to universal quantum circuits.

1.1.5 Other Applications

UCs were used to hide the functions in verifiable computation [FGP14] and multi-hop homomorphic encryption [GHV10], to hide queries in database management systems (DBMSs) [PKV$^+$14, FVK$^+$15] and to reduce verifier's preprocessing costs in NIZK argument [GGPR13]. Attrapadung [Att14] used UCs to transform the attribute-based encryption (ABE) schemes for any polynomial-size circuits [GGH$^+$13, GVW15] into ciphertext-policy ABE. UCs were also used to build the ABE scheme in [GGHZ14].

1.2 Related Works

Valiant viewed a Boolean circuit as a directed acyclic graph (DAG) and introduced an edge-universal graph (EUG) that edge embeds arbitrary DAGs (of a certain size) in a way that is analogous (and can be translated) to a universal circuit and its simulation of arbitrary circuits. Following Valiant and his follow-up works [Val76, LMS16, KS16, GKS17], we assume WLOG that the circuit has s inputs, t outputs, g gates of fan-in and fan-out 2, and let $n = s + g$ be the main parameter. Valiant gave a recursive construction of EUGs (and UCs) based on a k-way supernode (a graph object based on EUG, abbreviated as SN) parameterized by some constant k. As the main results, Valiant constructed a 2-way supernode of size 5 and a 4-way supernode of size 19, which gives rise to EUGs of size $5n \log n$ and $4.75n \log n$ respectively (and UCs of size approximately four times that of the corresponding EUGs, all omitting non-dominant terms). Later Cook and Hoover [CH85] gave a depth-preserving construction of UC with optimal depth $O(d)$ but larger size $O(n^3 d/ \log n)$, where d is the depth of circuit simulated. More recently,

Table 1: A comparison of previous results and ours in terms of the sizes of 4-way supernodes, EUGs, UCs and the number of AND gates, omitting non-dominant terms.

	$\vert\mathsf{SN}(4)\vert$	$\vert\mathsf{EUG}_2(n)\vert$	$\vert\mathsf{UC}^g_{s,t}\vert$	#(AND gates)
Valiant's UC [Val76]	19	$4.75n \log n$	$19n \log n$	$4.75n \log n$
Kolesnikov et al.[KS08b]	N/A	$0.25n \log^2 n$	$n \log^2 n$	$0.25n \log^2 n$
Lipmaa et al. [LMS16]	19	$4.75n \log n$	$18n \log n$	$4.75n \log n$
Our result	18	$4.5n \log n$	$17.75n \log n$	$4.5n \log n$

there have been ongoing efforts of implementations and optimizations of UC under Valiant's framework. Kolesnikov and Schneider [KS08b] proposed a practical UC with size-complexity roughly $0.25n \log^2 n$ and gave a first implementation of UC-based PFE under the Fairplay 2PC framework [MNPS04]. Despite not being asymptotically optimal their construction [KS08b] outperforms Valiant's UC for small scale circuits. Lipmaa et al. [LMS16, Sad15] further brought down the size of Valiant 4-way UC from $19n \log n$ to $18 \log n$ by reducing the number of XOR gates (while keeping the same number of AND gates). Moreover, Lipmaa et al. gave a general construction of k-way supernode and showed that their design has smallest size when $k = 3.147$. Independent of Lipmaa et al.'s work [LMS16], Kiss and Schneider [KS16] mainly focused on PFE, a prominent application of UC, for which the size of UC (and especially the number of AND gates) is significantly optimized. Further, they [KS16] borrowed building blocks from [KS08b] and proposed hybrid constructions of UCs for circuits with long inputs and outputs. Günther et al. [GKS17] implemented Valiant's 4-way UC and then provided a hybrid UC construction with further improved practical efficiency by combining Valiant's 2-way and 4-way UCs.

Valiant's 4-way universal circuits remained to date the most efficient construction (i.e., $4.75n \log n$). Motivated by aforementioned UC-based cryptographic applications, the efficiency improvement efforts towards making them practical and the trend of circuit size towards 10-million-gate or even billion-gate scale (e.g., [ABF$^+$17, ZCSH18]), it is natural to raise the following question:

Can we build more efficient UCs with better constant factors (i.e., smaller than 4.75) and is there a tighter bound on the size of EUG in Valiant's framework?

1.3 Our Contributions

We propose an algorithm that automates the search for optimal k-way supernodes (practical for $k \leq 4$), which yields a 4-way supernode of size 18 and depth 13 (as shown in Figure 1), improving upon the counterpart by Valiant [Val76] of size 19 and depth 14. Plugging it into Valiant's framework immediately brings down the size complexity of Valiant's UC (resp., EUG) from $19n \log n$ (resp., $4.75n \log n$) to $18n \log n$ (resp., $4.5n \log n$), where the size of UC $18n \log n$ can be further reduced to $17.75n \log n$ using the techniques from [LMS16]. In general, our 4-way supernode achieves an overall improvement of more than 5% in graph (circuit) size, along with a reduction of over 6% in graph (circuit) depth as a by-product. We refer to Table 1 for a detailed comparison with related works. As far as secure computation scenarios such as MPC and PFE are concerned, a practical efficiency indicator would be the number of AND gates (i.e., excluding XOR gates) and in this respect our work is also currently the best (more than 5% improvement over previous works). We implement our UC [Zha18a], evaluate its performance with a comparison to existing implementations (see Table 3) based on circuits of basic functions suitable for MPC and FHE, suggested by Tillich and Smart [TS15].

Furthermore, our supernode can be plugged into Valiant's 4-way UC or any applications that use the 4-way supernode as a blackbox to achieve improvements accordingly. For example,

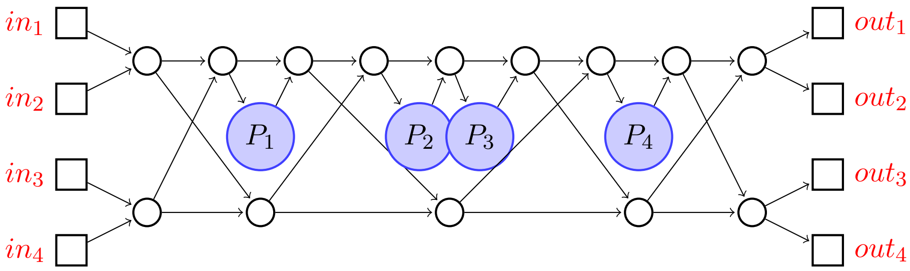


Figure 1: A 4-way supernode that consists of 18 nodes (excluding inputs and outputs).

our 4-way supernode was used in the recent hybrid UC [AGKS19], which was based on the hybrid UC from [GKS17] by replacing Valiant's 4-way counterpart. The engineering efforts of adapting the existing implementations to ours are affordable by replacing the supernode components, thanks to the modularity of Valiant's framework.

Our approach to the design of supernodes is computer aided (rather than by hand as in previous works), which could be of independent interest. Although not specific to 4-way supernodes, the time complexity of our algorithm when used in search of optimal k-way supernodes for $k \geq 5$ becomes impractically large. We stress that the implementations of k-way UC for $k \geq 5$, even if they exist with smaller size, are less desirable in practice. This is because the complexity of the conversion from an arbitrary circuit to the corresponding UC (which includes EUG generation, edge embedding, etc.) blows up dramatically with respect to k. This justifies why Valiant's 2-way UCs were implemented in [KS16] earlier than its 4-way counterpart in [GKS17] despite that the latter has slightly smaller circuit size. Still, for theoretical interests, we give a lower bound on the size of k-way supernodes (over all k's) as a complement, which in turn implies a lower bound on the size of universal circuit in Valiant's framework. That is, the size of an $\mathsf{EUG}_2(n)$ (resp., UC) is lower bounded by $3.644n \log n$ (resp., $14.576n \log n$). We note that a generic lower bound on UC size $\Omega(n \log n)$ was folklore, where the hidden constant (implicit in [Weg87, Theorem 8.1]) is quite small (about 1 as sketched in Section 4.1). We attribute this gap (14.576 vs. 1) to that either the generic bound is not tight or Valiant's approach to UC construction, despite its generality and modularity, might be only asymptotically optimal (i.e., not having a good constant factor). Given that most existing UC constructions were built upon Valiant's framework, we believe that our lower bound can be of practical relevance. Finally, it is left as an interesting open problem whether the gap between our construction and proved lower bound, $4.5n \log n$ vs. $3.644n \log n$, can be further reduced.

2 Preliminaries and Valiant's UC Construction

In this section, we give basic notations and definitions about universal circuits and explain Valiant's construction of universal circuits for completeness and accessibility. We refer to [LMS16] for an excellent exposition on Valiant's framework.

2.1 Notations and Definitions

2.1.1 Notations

$|G|$ (resp., $|C|$) refers to the size of a graph G (resp., circuit C), namely, the number of nodes (resp., gates) in G (resp., C). In this paper, we stick to the graph theoretical (rather than the standard electronics) terminology, where a circuit is represented by a Directed Acyclic Graph (DAG), inputs, outputs and gates are considered as nodes and wires are seen as edges of the DAG. $\mathsf{C}^g_{s,t}$ denotes a circuit with s inputs, t outputs and size up to g, and $\mathsf{UC}^g_{s,t}$ denotes a universal circuit which simulates arbitrary $\mathsf{C}^g_{s,t}$. $\mathsf{DAG}_d(n)$ is a DAG of size n and fan-in (and

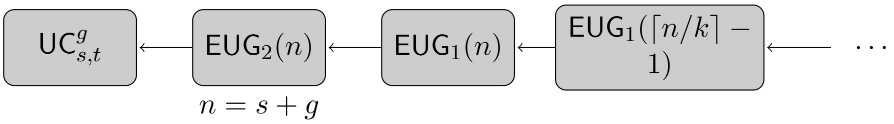


Figure 2: A high-level view of Valiant universal circuit construction [LMS16].

fan-out) d. Valiant [Val76] introduced Edge-Universal Graph (EUG) as defined in Definition 2.2 below. Loosely speaking, Universal Circuits to circuits are like Edge-Universal Graphs to Directed Acyclic Graphs. We use $\mathsf{EUG}_d(n)$ to denote an edge-universal graph that edge-embeds arbitrary $\mathsf{DAG}_d(n)$. Note that we have $|\mathsf{UC}^g_{s,t}| > g$ (resp., $|\mathsf{EUG}_d(n)| > n$) because $\mathsf{UC}^g_{s,t}$ (resp., $\mathsf{EUG}_d(n)$) simulates (resp., edge-embeds) any $\mathsf{C}^g_{s,t}$ (resp., $\mathsf{DAG}_d(n)$). We refer to the nodes of $\mathsf{EUG}_d(n)$ which are mapped from the corresponding vertices in $\mathsf{DAG}_d(n)$ as "poles" and other nodes which are used to simulate the structure of $\mathsf{DAG}_d(n)$ as common nodes.

Definition 2.1 (Universal Circuit) *A circuit $\mathsf{UC}^g_{s,t}$ is called a universal circuit, if for any circuit with s inputs, t outputs, size up to g (denoted by $\mathsf{C}^g_{s,t}$), there exists a set of program bits $p \in \{0,1\}^m$ such that $\mathsf{UC}^g_{s,t}$ can be programmed to realize $\mathsf{C}^g_{s,t}$, i.e., $\forall x \in \{0,1\}^s, \mathsf{UC}^g_{s,t}(x,p) = \mathsf{C}^g_{s,t}(x)$.*

Definition 2.2 (Edge-Universal Graphs) *An edge-embedding ϱ of $G = (V,E)$ into $G^* = (V^*, E^*)$ is a mapping that maps V into V^* one to one, and E into directed paths in G^* (i.e., $(i,j) \in E$ maps to a path from $\varrho(i)$ to $\varrho(j)$) that are pairwise edge-disjoint. A graph G^* is an edge-universal graph for $\mathsf{DAG}_d(n)$ if it has distinguished poles $P_1, \ldots, P_n$ such that every $G \in \mathsf{DAG}_{d_0}(n_0)$, with $d_0 \leq d$ and $n_0 \leq n$, can be edge-embedded into G^* by a mapping ϱ such that $\varrho(i) = P_i$ for each $i \in V$. This should hold for any labeling of G.*

2.2 From Edge-Universal Graphs to Universal Circuits

As depicted in Fig 2, Valiant's UC construction consists of the following steps:

1. Construct a $\mathsf{UC}^g_{s,t}$ from an $\mathsf{EUG}_2(n)$, where $n = g + s$;
2. Construct an $\mathsf{EUG}_2(n)$ from an $\mathsf{EUG}_1(n)$;
3. Construct an $\mathsf{EUG}_1(n)$ given an $\mathsf{EUG}_1(\lceil n/k \rceil - 1)$ for some constant k;
4. Repeat Step 3 recursively until reaching an EUG of some small size that can be trivially constructed.

2.2.1 Construct $\mathsf{UC}^g_{s,t}$ from $\mathsf{EUG}_2(n)$

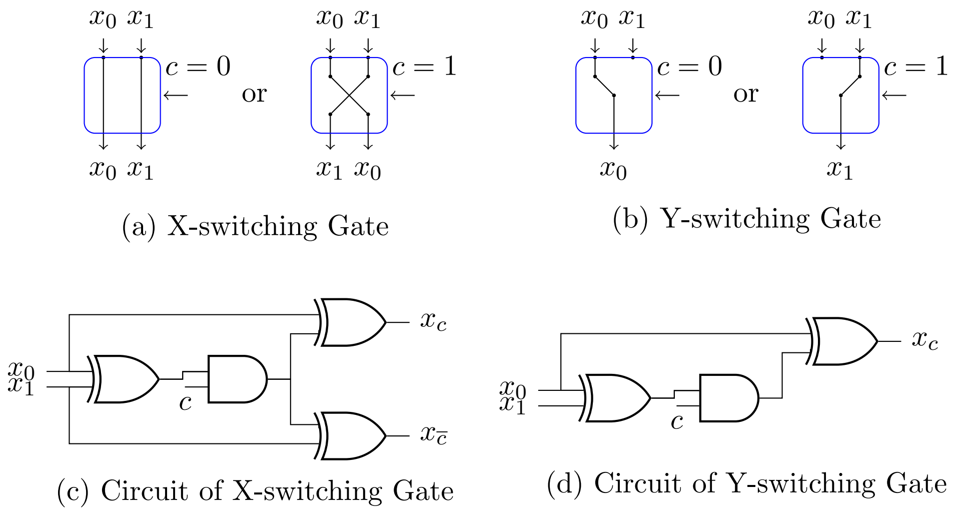
To build a universal circuit $\mathsf{UC}^g_{s,t}$ from a $\mathsf{EUG}_2(n)$ [2], each node in $\mathsf{EUG}_2(n)$ should be implemented by Boolean gates and each edge is a wire of $\mathsf{UC}^g_{s,t}$. The details are as follows.

- Each pole is implemented by a universal gate (UG). A 2-input UG supports any of the 16 possible gate types represented by the 4 control bits of the gate table (c_1, c_2, c_3, c_4). It computes function ug: $\{0,1\}^2 \times \{0,1\}^4 \to \{0,1\}$ as follows:

$$ug(x_1, x_2, c_1, c_2, c_3, c_4) = \overline{x_1}\overline{x_2}c_1 + \overline{x_1}x_2c_2 + x_1\overline{x_2}c_3 + x_1x_2c_4 \tag{1}$$

  A UG can be implemented with 3 AND and 6 XOR gates [LMS16]. The control bits c_1, c_2, c_3, c_4 are part of the program bits of the universal circuit.

[2] Definition 2.2 puts no limits on the fan-in/fan-out of EUG, but Valiant's UC construction requires the underlying EUG to be a DAG_2.



(a) X-switching Gate

(b) Y-switching Gate

(c) Circuit of X-switching Gate

(d) Circuit of Y-switching Gate

Figure 3: Switching gates and their circuit implementations.

- Each common node with indegree and outdegree both 2 can be implemented by an X-switching gate [KS08a], that computes $f_X : \{0,1\}^2 \times \{0,1\} \to \{0,1\}^2$ (Figure 3a). The inputs of an X-switching gate are forwarded to its outputs, switched or not switched, depending on control bit c. This block can be implemented with 1 AND gate and 3 XOR gates (Figure 3c).
- Each common node with indegree 2 and outdegree 1 can be implemented by a Y-switching gate [KS08a] , that computes $f_Y : \{0,1\}^2 \times \{0,1\} \to \{0,1\}$ (Figure 3b). A Y-switching gate takes as input two bits and produces one of them as output, depending on control bit c. This block can be implemented with 1 AND gate and 2 XOR gates (Figure 3d).
- Each common node with indegree 1 and outdegree 2 (i.e., splitter gate) is replaced by two outgoing wires to copy its input to the two outputs.
- Each common node with indegree 1 and outdegree 1 is replaced by a wire.

This completes the construction of $\mathsf{UC}^g_{s,t}$ from $\mathsf{EUG}_2(n)$. It remains to show how $\mathsf{UC}^g_{s,t}$ simulates a given circuit $\mathsf{C}^g_{s,t}$ (as intended for a universal circuit), where simulation is essentially setting the input wires and the program (and control) bits for all universal gates and switching gates.

2.2.2 Simulate $\mathsf{C}^g_{s,t}$ using $\mathsf{UC}^g_{s,t}$

Following [Val76, LMS16, GKS17], we assume WLOG that the circuits have fan-in/fan-out bounded by two, and it is well-known that any circuit of unbounded fan-in/fan-out can be transformed into a functionally equivalent one by paying reasonable prices in size ($\mathsf{C}^g_{s,t} \subset \mathsf{C}^{2g+t,2}_{s,t}$). [Val76, Cor 3.1].

We model the circuit $\mathsf{C}^g_{s,t}$ as a graph $G_C = (V_C, E_C)$ where each input wire and each gate are represented as a node and each wire is represented by an edge in the graph. The derived graph is a $\mathsf{DAG}_2(n)$ with $n = s + g$. By Deition 2.2, it is possible to embed G_C into an $\mathsf{EUG}_2(n)$, such that for every edge $(v_i, v_j) \in E_C$, there is a path from v_i to v_j that is edge-disjoint to other paths. These paths constitute set $Q = \{Q_1, Q_2, \ldots, Q_{|E_C|}\}$, which will be used to determine the control bits of the switching gates in $\mathsf{UC}^g_{s,t}$, the universal circuit corresponding to the $\mathsf{EUG}_2(n)$ above. We set the control bits and input wires as follow.

- **Control bits of switching gates.** For an X-(/Y-)switching gate G_S of $\mathsf{UC}^g_{s,t}$, we denote by N_S the corresponding node in $\mathsf{EUG}_2(n)$. If a path $Q_i \in Q$ passes through N_S, we set the control bit of G_S to satisfy the direction of Q_i through N_S.[3] If no paths go through N_S, we can set arbitrary binary value for the control bit of G_S.

- **Control bits of universal gates and input wires of universal circuit.** For a universal gate G_U of $\mathsf{UC}^g_{s,t}$, we denote by N_U the corresponding pole in $\mathsf{EUG}_2(n)$. If N_U represents a gate of the given circuit $C^g_{s,t}$, we set the control bits of G_U to realize the gate. If N_U represents an input of $C^g_{s,t}$, we can set arbitrary binary values for the control bits of G_U and set the output wire of G_U as an input wire of $\mathsf{UC}^g_{s,t}$.

This completes the simulation. Now we analyze the complexity of $\mathsf{UC}^g_{s,t}$.

Lemma 2.1 *$|\mathsf{UC}^g_{s,t}| \le 4|\mathsf{EUG}_2(n)| + 5n$, where $n = s + g$*

Proof. From the construction of $\mathsf{UC}^g_{s,t}$, we know that the size of $\mathsf{UC}^g_{s,t}$ is related to the numbers of X-switching gates (denoted by n_X), Y-switching gates (denoted by n_Y) and the universal gates (exactly n), which can be expressed as: $|\mathsf{UC}^g_{s,t}| = 4n_X + 3n_Y + 9n \le 4(n_X + n_Y + n) + 5n \le 4|\mathsf{EUG}_2(n)| + 5n$, as switching gates (which amount to $n_X + n_Y$) are part of the common nodes in $\mathsf{EUG}_2(n)$. □

In Valiant's supernode design, the fan-in/fan-out of every common node is two, meaning that there are no Y-switching gates and splitters in the corresponding UC (i.e., $n_Y = 0$). In that case, the inequality in Lemma 2.1 can be used as an equality. Later, the supernode designed by Lipmaa et al. [LMS16] additionally utilized Y-switching gates and splitters to reduce the number of XOR gates, which we will elaborate in the next section. In summary, we reduce the construction of UC to that of $\mathsf{EUG}_2(n)$, which will be our focus for the remainder of this section.

2.3 Edge-Universal Graphs: from $\mathsf{EUG}_1(n)$ to $\mathsf{EUG}_2(n)$

Next we show how to construct from $\mathsf{EUG}_1(n)$ to $\mathsf{EUG}_2(n)$.

Lemma 2.2 (Lemma 2.1 from [Val76]) *For any $\mathsf{DAG}_d(n) = (V, E)$, E can be regarded as the union of d disjoint set E_i, i.e., $E = \cup_{i=1}^d E_i$, such that each (V, E_i) is a $\mathsf{DAG}_1(n)$.*

Lemma 2.3 ([LMS16]) *An $\mathsf{EUG}_2(n)$ can be constructed from two instances of $\mathsf{EUG}_1(n)$.*

Proof. An $\mathsf{EUG}_2(n)$ is constructed from two $\mathsf{EUG}_1(n)$, which can be achieved by merging every two poles in the same positions of the two $\mathsf{EUG}_1(n)$. Then we prove that any $\mathsf{DAG}_2(n) = (V, E)$ can be edge-embedded into the $\mathsf{EUG}_2(n)$. By Lemma 2.2 we can divide E into two sets E_1 and E_2 such that each (V, E_i) is a $\mathsf{DAG}_1(n)$, and therefore we can embed each in a separate $\mathsf{EUG}_1(n)$. The edge-embedding from (V, E) to $\mathsf{EUG}_2(n)$ is the combination of two edge-embeddings from (V, E_i) to the respective $\mathsf{EUG}_1(n)$. This completes the $\mathsf{EUG}_2(n)$ construction. □

As we mentioned before, when constructing a $\mathsf{UC}^{s,t}_g$ we need the $\mathsf{EUG}_2(n)$ to be a DAG_2. So the $\mathsf{EUG}_1(n)$ used to construct this $\mathsf{EUG}_2(n)$ also needs to be a DAG_2 and the indegree (outdegree) of poles of $\mathsf{EUG}_1(n)$ should be 1. Therefore, when we talk about Valiant's construction, the edge-universal graphs $\mathsf{EUG}_1(n)$ and $\mathsf{EUG}_2(n)$ should meet the requirements above.

[3] Since N_S is a common node, it cannot be an endpoint of a path. For a X-switching gate G_S, there may be two paths passing through N_S, for which only a single control bit is needed as paths in Q are edge-disjoint by definition.

2.4 Edge-Universal Graphs: from $\mathsf{EUG}_1(\lceil n/k\rceil - 1)$ to $\mathsf{EUG}_1(n)$

Now that we reduce the construction of $\mathsf{UC}^g_{s,t}$ to the design of $\mathsf{EUG}_1(n)$. What we will show next is a reduction of $\mathsf{EUG}_1(n)$ to itself of smaller sizes (which can be done recursively until reaching an EUG_1 of trivial size we have on hand). The recursion relies on an essential building block called supernode (see Definition 2.3) and we use it to reduce $\mathsf{EUG}_1(n)$ to $\mathsf{EUG}_1(n/k)$ in each step.

Definition 2.3 (Supernode) *A k-way supernode $\mathsf{SN}(k)$ is an edge-universal-graph with k inputs $\{in_1, \ldots, in_k\}$, k outputs $\{out_1, \ldots, out_k\}$, k poles $P = \{P_1, \ldots, P_k\}$ and m other nodes (called common nodes), such that any graph $G = (V, E) \in \mathsf{DAG}_1(3k)$, where $V = \{in_1, \ldots, in_k\}$ $\cup \{P_1, \ldots, P_k\} \cup \{out_1, \ldots, out_k\}$, and every edge $e = (v_1, v_2) \in E$ satisfies the conditions below:*

1. *If $v_1 \in \{in_1, \ldots, in_k\}$ then $v_2 \in P$.*
2. *If $v_2 \in \{out_1, \ldots, out_k\}$ then $v_1 \in P$.*
3. *$v_1 \notin \{out_1, \ldots, out_k\}$.*
4. *$v_2 \notin \{in_1, \ldots, in_k\}$.*

can be edge embedded into $\mathsf{SN}(k)$. The size[4] of $\mathsf{SN}(k)$ is the defined as $m + k$.

As an example, Figure 1 is a 4-way supernode. Given a k-way supernode, we can reduce the problem of EUG construction to itself (of smaller sizes) in a recursive way. This is stated as the theorem below and for self-containedness we sketch its main idea (visualized in Figure 4) and refer to the appendix for a full proof. That is, given an $\mathsf{EUG}_1(\lceil \frac{n}{k} \rceil - 1)$ and $\mathsf{SN}(k)$, we construct a $\mathsf{EUG}_1(n)$ as follows. We connect $\lceil \frac{n}{k} \rceil$ k-way supernodes together by merging the inputs and outputs of two adjacent supernodes one by one (e.g. merge out^1_1 and in^2_1 into one [5]). We divide those merged nodes into k groups and invoke $\mathsf{EUG}_1(\lceil \frac{n}{k} \rceil - 1)$ for each group (see Figure 4).

Theorem 2.1 ([Val76, LMS16]) *Given an $\mathsf{EUG}_1(\lceil \frac{n}{k} \rceil - 1)$ and a k-way supernode $\mathsf{SN}(k)$, there exists an explicit construction of $\mathsf{EUG}_1(n)$ of size*

$$k \cdot |\mathsf{EUG}_1(\lceil \frac{n}{k} \rceil - 1)| + \lceil \frac{n}{k} \rceil \cdot |\mathsf{SN}(k)| \ .$$

With $\mathsf{SN}(k)$ we recursively reduce the problem to itself of smaller sizes, and we just need an EUG_1 of small size, say $\mathsf{EUG}_1(k)$, at initialization. Note that $\mathsf{EUG}_1(k)$ is already implied by and can be extracted from $\mathsf{SN}(k)$. In summary, $\mathsf{SN}(k)$ can be used to build EUGs of arbitrary size. We refer to this approach to UC construction (from supernodes) as Valiant's construction (or Valiant's framework) and see Figure 4 for the high-level overview. Clearly, the complexity of Valiant framework is related to the size of the supernode used, which will be analyzed in the next subsection.

2.5 Circuit Complexity in Valiant's Framework

Valiant's approach to universal circuits remains the most efficient to date, and thus we consider the complexity of UC and EUG constructed in Valiant's framework. The following equations are from Theorem 2.1 and Lemma 2.3:

$$|\mathsf{EUG}_2(n)| = 2|\mathsf{EUG}_1(n)| - n \ , \tag{2}$$

[4] As a slight abuse of definition, the size of a supernode is different from that of a graph by excluding input and output nodes. As we will see, it comes in handy when composing the components to build a large EUG and calculating its size.

[5] in^i_j (out^i_j) denotes the j-th input (output) of the i-th supernode (denoted by $\mathsf{SN}(k)_i$)

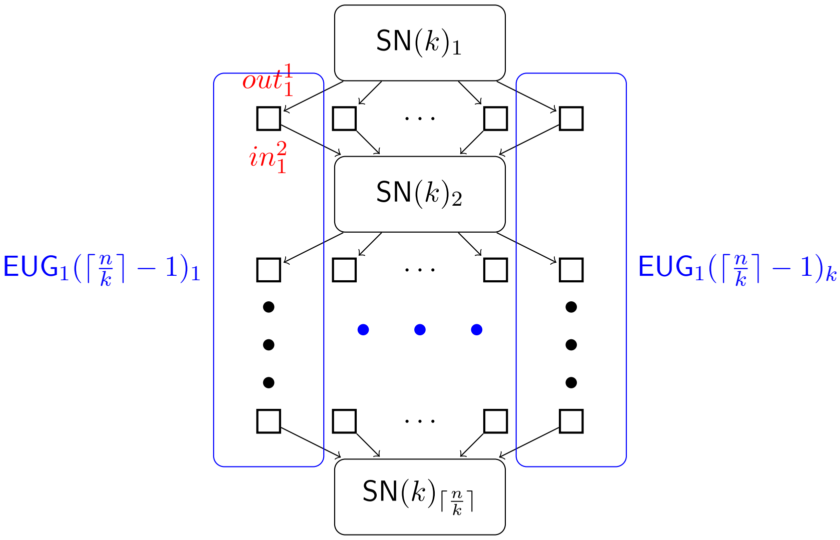

$\mathsf{SN}(k)_1$
out_1^1
in_1^2
$\mathsf{SN}(k)_2$
$\mathsf{EUG}_1(\lceil\frac{n}{k}\rceil - 1)_1$
$\mathsf{EUG}_1(\lceil\frac{n}{k}\rceil - 1)_k$
$\mathsf{SN}(k)_{\lceil\frac{n}{k}\rceil}$

Figure 4: Valiant's construction of $\mathsf{EUG}_1(n)$ based on $\mathsf{EUG}_1(\lceil\frac{n}{k}\rceil - 1)$ and $\mathsf{SN}(k)$.

$$|\mathsf{EUG}_1(n)| = k|\mathsf{EUG}_1(\lceil\frac{n}{k}\rceil - 1)| + \lceil\frac{n}{k}\rceil|\mathsf{SN}(k)| \ . \tag{3}$$

By using recurrence relation above, we get

$$|\mathsf{EUG}_2(n)| = \frac{2|\mathsf{SN}(k)|}{k\log k}n\log n - O(n) \ , \tag{4}$$

$$|\mathsf{CircuitEUG}_2(n)| = \frac{2|\mathsf{CircuitSN}(k)|}{k\log k}n\log n - O(n) \ , \tag{5}$$

where $\mathsf{CircuitEUG}_d(n)$ denotes the circuit counterpart of $\mathsf{EUG}_2(n)$ in Equation 4. The size of UC can be estimated by combining Equation 4 with Lemma 2.1 [Val76]:

$$|\mathsf{UC}_{s,t}^g| = \frac{8|\mathsf{SN}(k)|}{k\log k}n\log n - O(n), \text{where } n = s+t+2g \ . \tag{6}$$

Next, we consider depth and from Figure 4 we know:

$$\begin{aligned}\mathsf{depth}(\mathsf{EUG}_1(n)) &= \lceil\frac{n}{k}\rceil\mathsf{depth}(\mathsf{SN}(k)) + (\lceil\frac{n}{k}\rceil - 1)\\ &= \frac{n}{k}(\mathsf{depth}(\mathsf{SN}(k)) + 1) + O(1) \ .\end{aligned} \tag{7}$$

Combining with Lemma 2.3, we have:

$$\begin{aligned}\mathsf{depth}(\mathsf{UC}_{s,t}^g) &= \mathsf{depth}(\mathsf{CircuitEUG}_1(n))\\ &= \lceil\frac{n}{k}\rceil\mathsf{depth}(\mathsf{CircuitSN}(k)) + (\lceil\frac{n}{k}\rceil - 1)\mathsf{depth}(\mathsf{X\text{-}switching}) \ .\end{aligned} \tag{8}$$

The depth of the circuit of $\mathsf{SN}(k)$ is $3 \times \mathsf{depth}(\mathsf{SN}(k))$ [6] as the X- and Y-switching gates are both of depth 3 (see Figure 3). Thus, its depth complexity is:

$$\mathsf{depth}(\mathsf{UC}_{s,t}^g) = \frac{3 \times \mathsf{depth}(\mathsf{SN}(k)) + 3}{k}n + O(1) \ . \tag{9}$$

We summarize in Table 2 known results about the size and depth of supernode and corresponding UCs. As we can see, the size and depth of Valiant's universal circuits crucially depend on the respective size and depth of the underlying k-way supernode. This motivates our search for a smaller supernode for some practical value of k.

[6] Similar to the size of supernode, we define the depth of $\mathsf{SN}(k)$ as the length of the longest path minus 2 (i.e., excluding inputs and outputs), denoted by $\mathsf{depth}(\mathsf{SN}(k))$.

Table 2: The known results of UC size and depth.

k	Supernode size	Supernode depth	$\lvert\mathsf{UC}^g_{s,t}\rvert$	$\mathsf{depth}(\mathsf{UC}^g_{s,t})$
2-way	5	5	$20n\log n$[Val76]	$9n$
3-way	12	7	$20.19n\log n$ [GKS17]	$8n$
Valiant's 4-way	19	14	$19n\log n$[Val76, GKS17]	$11.25n$
Our 4-way	18	13	$18n\log n$	$10.5n$

3 A New Design of Supernode via Automated Search

In this section, we introduce an automated approach to the design of supernodes. As a main end result, we get a better 4-way supernode with an overall improvement of more than 5% on the efficiency of UC constructions and their applications, stated as the theorem below. We refer to the external link [Zha18b] for a lengthy (computer generated) proof that Figure 1 gives a 4-way supernode, where all effective DAGs are exhausted and their edge-embeddings into the supernode are provided. As we will show, it is already size optimal (as a 4-way supernode) as 4-way supernodes of size 17 do not exist.

Theorem 3.1 (4-way SN and EUG, revisited) *The graph in Figure 1 is a 4-way supernode with 18 nodes (excluding inputs and outputs), which implies an* $\mathsf{EUG}_2(n)$ *of size* $4.5n\log n - O(n)$ *and depth* $3.5n + O(1)$.

3.1 Construction of Supernodes

While giving constructions of 2-way and 4-way supernodes in his work [Val76], Valiant gave no details on how the constructions were obtained. Lipmaa et al. [LMS16] formalized and explained the k-way supernode construction methodology in a modular and intuitive way. As depicted in the right-hand of Figure 5, a general design of k-way supernode consists of two layers of permutation-networks (PNs) at both ends and an EUG augmented with $k-1$ additional nodes in between. For $k = 4$, the size of $\mathsf{SN}(4)$ following the general design is

$$2|\mathsf{PN}| + |\mathsf{EUG}_1(k)| + k - 1 = 10 + 7 + 3 = 20 \ .$$

Looking back, Valiant's 4-way supernode can be regarded as an optimized version of the general design by saving a node from one of the permutation networks (see the comparison in Figure 5). One might think that by exploiting the symmetry it is possible to save two nodes (one from each permutation network) to get a 4-way supernode of smaller size (i.e., 18). Unfortunately, this intuition does not work because the resulting graph would not be a supernode any more, which was refuted by our supernode testing algorithm (presented in the next subsection). It remained open if one can construct more size-efficient supernodes. Next we will present an algorithm for testing whether a graph is supernode or not, and an automated searching algorithm for more size-efficient supernodes.

3.2 Supernode Test for Graphs

As the first step, we propose a method to check whether a graph (with k inputs, k outputs, k poles and m common nodes) is a k-way supernode or not. A k-way supernode is an edge-universal-graph that edge embeds any graph $G \in \mathsf{DAG}_1(3k)$ (see Definition 2.3) and thus it

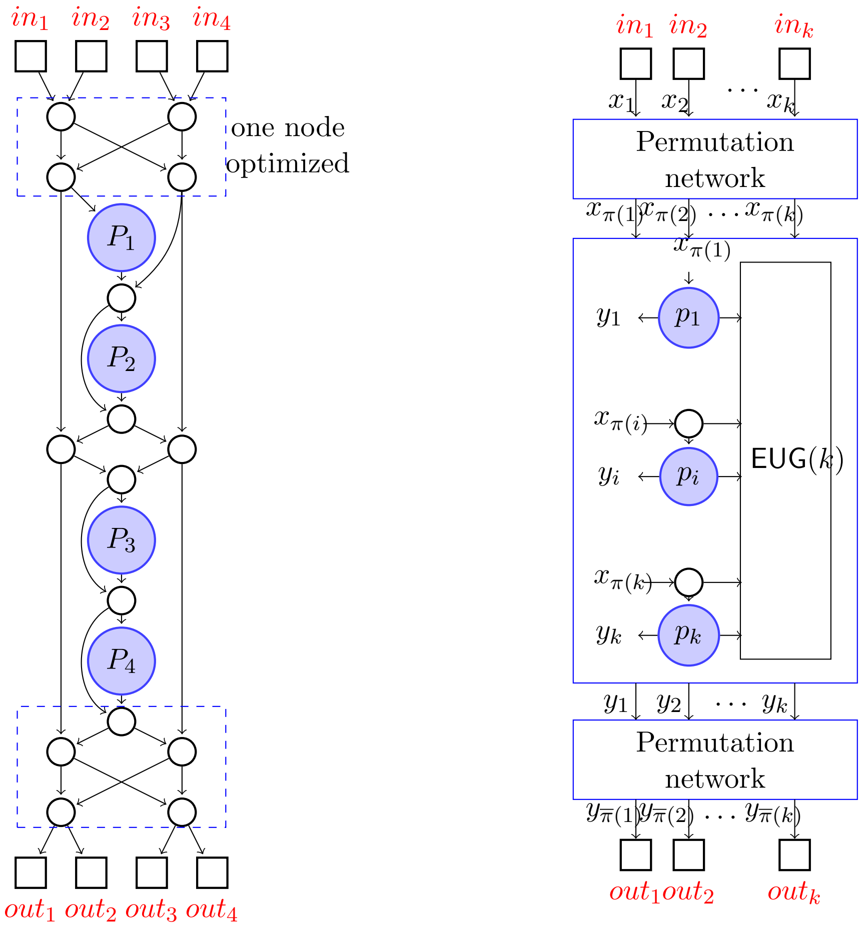


Figure 5: A comparison of Valiant's $\mathsf{SN}(4)$ and the general design of $\mathsf{SN}(k)$ from [LMS16].

seems necessary to enumerate all $G \in \mathsf{DAG}_1(3k)$. For efficiency, we observe that it suffices to enumerate over a special type of graph called pole-complete graphs, and the remaining graphs can be omitted as they are already implied. As we will see in the next section, the notion of pole-complete graphs will also be useful for proving the lower bound.

Definition 3.1 (Pole-complete Graph) *For $G = (V, E) \in \mathsf{DAG}_1(3k)$ with k inputs, k outputs, and k poles $P_1, \dots, P_k$ that are topologically ordered, we say that G is pole-complete if*

1 *Every edge of G satisfies the four properties stated in Definition 2.3;*

2 *For any pole $p \in \{P_1, \dots, P_k\}$, there exist $e_1 = (v_1, v_2), e_2 = (v_3, v_4) \in E$ such that $v_2 = p$ and $v_3 = p$.*

We denote by F_k the number of all the k-way pole-complete graphs $G \in \mathsf{DAG}_1(3k)$.

Informally, for any $G = (V, E) \in \mathsf{DAG}_1(3k)$ to be edge-embedded into the candidate supernode (Definition 2.3), we can see G as a set of paths. We call G pole-complete if for each path its start-node is an input (from $\{in_1, \dots, in_k\}$), the middle-nodes (poles) are topological sorted, and its end-node is an output. "Pole-complete" means that all the k poles are in the paths.

Lemma 3.1 *A graph G_0 with k inputs $\{in_1, \dots, in_k\}$, k outputs $\{out_1, \dots, out_k\}$, and k poles $P = \{P_1, \dots, P_k\}$ is a $\mathsf{SN}(k)$ if any pole-complete graph $G \in \mathsf{DAG}_1(3k)$ can be edge-embedded into G_0.*

Proof. First, we observe that if graph $G = (V, E)$ can be edge-embedded into graph G_0, then so can any subgraph $G' = (V', E')$ of G since the edge-embedding of G' is implied by that of G by ignoring those edges $e \in E \setminus E'$ (recall $E' \subset E$). Next, we prove that for any graph $G' = (V', E') \in \mathsf{DAG}_1(3k)$ satisfying Definition 2.3 but is not pole-complete, there exists a pole-complete $G \in \mathsf{DAG}_1(3k)$ such that G' is a subgraph of G. We construct such G by adding edges into G'. As mentioned before, $G' \in \mathsf{DAG}_1(3k)$ can be regarded as a set of several paths. Since G' is not pole-complete, there must be one or more isolated poles not in the paths, or there are one or more paths start (or end) with poles, called starting poles (or ending poles). We put all isolated poles in a path and add the path to G'. Then, for each starting pole (or ending pole), we add an edge that connects an isolated input to (or output from) it. Note that we can always find such isolated input/output nodes as the number of input/output nodes equals to the number of poles. At last, we construct a supergraph of G' which is pole-complete. □

We use a depth-first-search algorithm to find an edge-embedding of pole-complete G, and repeat the process on all pole-complete ones. In a pole-complete graph, the precursor-node (abbreviated as pre-node) of the first pole P_1 should reside in the k inputs, denoted by in_i, and the pre-node of P_2 should be in $\{P_1, in_1, \ldots, in_{i-1}, in_{i+1}, \ldots, in_k\}$, with k possibilities as well. Therefore, the pre-node of every pole each has k different possibilities and there are k^k possibilities to enumerate. Then, we connect inputs and the poles to form several (no greater than k) paths. Finally, we enumerate the arrangement of outputs for the paths to get the pole-complete graph G.

3.3 Search for More Size-efficient k-way Supernodes

As given in Definition 2.3, we define the size of a supernode $\mathsf{SN}(k)$ as the sum of the numbers of poles and common nodes and we find it convenient to compute the size of EUG in Valiant's framework (see Footnote 4). Thus, the supernode of size n has $n+2k$ nodes (k inputs, k outputs, k poles and $n-k$ common nodes). To search for $\mathsf{SN}(k)$ of size n, we number the nodes in $\mathsf{SN}(k)$ as $N_1, N_2, \ldots, N_{n+2k}$ with $N_1, N_2, \ldots, N_k$ as inputs, $N_{n+k+1}, N_{n+k+2}, \ldots, N_{n+2k}$ as outputs and $N_{k+1}, N_{k+2}, \ldots, N_{n+k}$ as poles and common nodes (collectively referred to as middle nodes). The idea of searching for a $SN(k)$ of size n is to enumerate the pre-nodes of each node in the graph, and output if it is a supernode (using the supernode test method from the last subsection). For example, if the inputs have no pre-nodes, we can just set the k inputs as isolated nodes at initialization. For a middle node N_i ($k < i < n + k + 1$), the number of its pre-nodes can be one (if N_i is a pole) or two (otherwise), so we must consider both possibilities. Upon the enumeration of N_j as N_i's pre-node candidate, we should check whether N_j is legal or not, in particular, if N_j's out-degree is 2 or N_j is an input or pole and its out-degree is 1, then N_j is not a pre-node of N_i (because the $\mathsf{SN}(k)$'s fan-out is 2 and the out-degree of an input or pole must be 1). This condition for N_j is described as "N_j's out-degree is not full" in line 8 and line 18 of Algorithm 3.3. At last, we add the k outputs as the successor nodes of the nodes whose out-degree is not full. The steps above allow for an automated search over all candidates. However, the above search is not efficient as it enumerates all candidates, many of which could have been ruled out from supernode tests. So we add the pruning method to improve efficiency. After choosing a middle node as the j-th pole, we check whether graph G we construct can be a part of $\mathsf{SN}(k)$ or not, for which we need to enumerate all the $\mathsf{DAG}_1(k + j)$ (with k inputs and j poles, see Definition 2.3) and check whether those $\mathsf{DAG}_1(k + j)$s can be edge-embedded into G or not. We refer to Algorithm 3.3 for the pseudocode of search for supernode $\mathsf{SN}(k)$ of size n, where the pruning method is invoked in line 10.

3.4 New Constructions

We run the automated tool on a PC to search for k-way supernodes. We start with 3-way supernodes (the case of $k = 2$ is trivial). The search for $\mathsf{SN}(3)$ of size 11 failed, and an outcome

Algorithm 1 The search algorithm for $\mathsf{SN}(k)$ of size n

```
Require: $k, n$
Ensure: All $k$-way supernodes of size $n$ (if exists)
Initialize the graph $G$
ADDNODE($G$,$k+1$)

function AddNode($G, i$)
    if $i \geq k+n$ then
        if #(G's pole)$< k$ then
            for $j = 1 \to i-1$ do
                if $N_j$'s outdegree is not full then
                    Addedge($N_j, N_i$) to $G$
                    if G passes the pruning method test then
                        ADDNODE($G$,$i+1$)
                    end if
                end if
            end for
        end if
        for $j = 1 \to i-1$ do
            for $k = 1 \to j-1$ do
                if ($N_j$'s outdgree is not full) and ($N_k$'s outdgree is not full) then
                    Addedge($N_j, N_i$) to G
                    Addedge($N_k, N_i$) to G
                    ADDNODE($G, i+1$)
                end if
            end for
        end for
    else
        Add the output nodes for $G$;
        if $G$ is a Supernode then
            output $G$;
        end if
    end if
end function
```

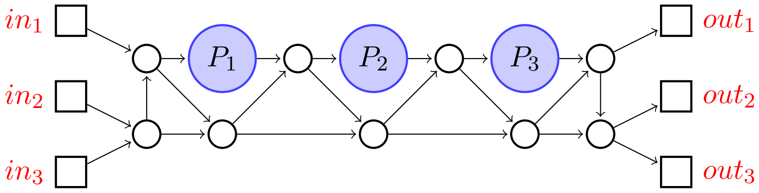


Figure 6: A 3-way supernode that consists of 12 nodes.

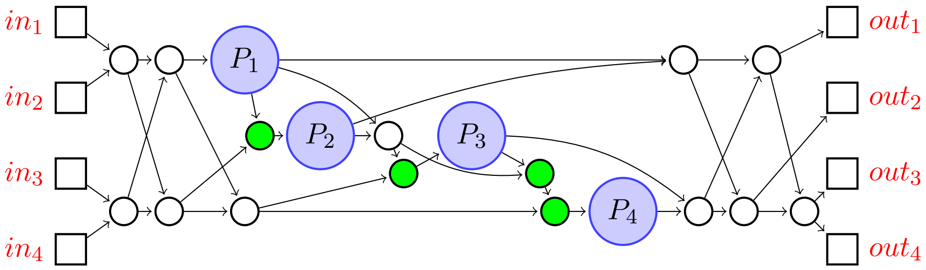


Figure 7: The 4-way split supernode construction from [LMS16], where each green node can be implemented by a Y-switching gate.

of $\mathsf{SN}(3)$ of size 12 is illustrated in Figure 6, which is already known in literature [GKS17].

We proceed to the case $k = 4$. For the 4-way supernode of size 17, the search exits in a couple of minutes without any outcome, meaning that no such exist. For the 4-way supernode of size 18, the search runs in a number of minutes and returns the outcomes [7], which are depicted in the Figure 1. This beats the best previously known result by Valiant [Val76] of size 19. As a result, we improve the size of $\mathsf{EUG}_2(n)$ from $4.75n \log n$ to $4.5n \log n$ (omitting smaller terms).

Moving from $k = 4$ to $k = 5$ seems a tiny step. However, for $k = 5$ the search algorithm is not terminating due to the substantially higher time complexity. For the 4-way supernode of size 18, we search for 6859734 candidate graphs (already after pruning) and for each candidate we should enumerate 5056 $\mathsf{DAG}_1(3\times4)$s to decide whether it is a supernode or not. That justifies why it takes several minutes to get the results. Nevertheless, for $k = 5$ we target at supernodes of size 26 (any 5-way supernode with size 27 or more yields an $\mathsf{EUG}_2(n)$ of size greater than $4.5n \log n$), then the number of candidate graphs grows rapidly to almost 2^{47} , and for each candidate we need to enumerate about 2^{18} $\mathsf{DAG}_1(3 \times 5)$s, where the product 2^{65} is beyond the reach of a PC. We did try other methods (e.g. SAT solvers) to improve the efficiency for $k = 5$. But the attempt failed due to the difficulty of finding out the SAT formula determining whether a DAG can be embedded into a supernode candidate or not.

By replacing each common node with an X-switching gate and each pole with a universal gate, we immediately convert the $\mathsf{EUG}_2(n)$ to a universal circuit of size $18n \log n+O(n)$ and thus improve upon the Valiant's UC of size $19n \log n$. However, while our UC size seems the same as $18n \log n$ achieved by Lipmaa et al. [LMS16], their UC construction was based on Valiant's supernode and decreased its total number of gates by replacing 4 X-switching gates with 4 Y-switching gates (see Figure 7). In other words, their construction reduces only the number of XOR gates (and that of AND gates remain the same as [Val76]) and thus the improvement may not be appreciated by applications such as MPC and PFE with UC, where XOR gates can be evaluated for free [KS08b]. Further, we can use the same idea from [LMS16] to save some XOR gates. For example, based on our supernode we change an X-switching gate to Y-switching gate (the black node in Figure 8), and the size of universal circuit now becomes $17.75n \log n + O(n)$, which is better than [LMS16].

At last, our 4-way UCs are also shallower than the counterparts in literature [Val76, LMS16]. The depth of Valiant's $\mathsf{SN}(4)$ is 14 but ours is 13. From Equation 7 and Equation 9, we know that the depth of the EUG (resp. UC) based on our 4-way supernode is $3.5n$ (resp. $10.5n$), which is better than (and improves by 6.67%) Valiant's $3.75n$ (resp. $11.25n$). However, if one only cares about depth, then he would just use 3-way supernode of depth 7 (see Figure 6) to get a UC of depth $8n$. Otherwise said, the depth improvement on 4-way UC is considered as a by-product (instead of a main advantage) of our UC construction.

[7] The search algorithm outputs a few hundred of outcomes many of which are isomorphic to each other, but our verification is by hand and is certainly not exhaustive.

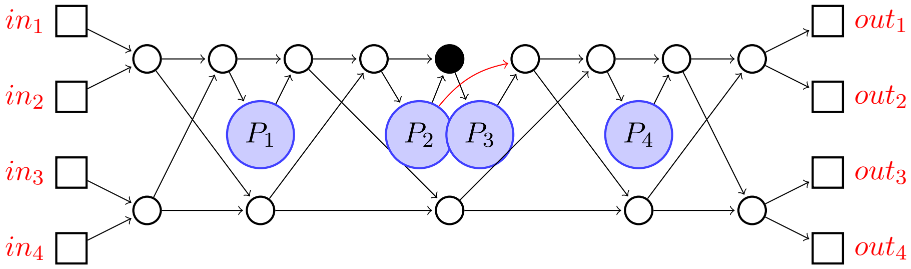


Figure 8: Our 4-way supernode can be improved (in the sense of circuit size) by replacing an X-switching gate with a Y-Switching gate at the black node.

Table 3: A comparison (in terms of the number of AND gates) of the (Kiss et al.'s 2-way, Günther et al.'s 4-way and hybrid, and our 4-way) UC implementations to simulate sample circuits from [TS15].

Circuit	$n = g + s$	2-way UC[KS16]	4-way UC [GKS17]	Hybrid UC[GKS17]	Our 4-way UC
Credit Checking	82	$1.50 \cdot 10^3$	$1.51 \cdot 10^3$	$1.49 \cdot 10^3$	$1.50 \cdot 10^3$
Mobile Code	160	$3.65 \cdot 10^3$	$3.88 \cdot 10^3$	$3.61 \cdot 10^3$	$3.82 \cdot 10^3$
ADD-32	342	$9.58 \cdot 10^3$	$9.55 \cdot 10^3$	$9.44 \cdot 10^3$	$9.30 \cdot 10^3$
MULT-32X32	12202	$6.54 \cdot 10^5$	$6.50 \cdot 10^5$	$6.35 \cdot 10^5$	$6.24 \cdot 10^5$
AES-exp	38518	$2.39 \cdot 10^6$	$2.38 \cdot 10^6$	$2.31 \cdot 10^6$	$2.27 \cdot 10^6$
DES-exp	32207	$1.98 \cdot 10^6$	$1.94 \cdot 10^6$	$1.90 \cdot 10^6$	$1.87 \cdot 10^6$
SHA-256	201206	$1.49 \cdot 10^7$	$1.46 \cdot 10^7$	$1.44 \cdot 10^7$	$1.39 \cdot 10^7$

3.5 Implementation and Performance Evaluation

As we mentioned before, the universal circuits based on our 4-way supernode have smaller circuit size than other constructions especially for large n (when emulating large-size circuits). We implement our 4-way construction [Zha18a] and compare it with the implementations of Valiant's 2-way [KS16], 4-way and their hybrid [GKS17]. Table 3 evaluates the performances based on circuits of basic functions suitable for MPC and FHE, provided by Tillich and Smart [TS15]. In particular, Table 3 compares the number of AND gates in our universal circuits with other works[8], where our work is tabulated in the last column of Table 3 and the statistics of other works are picked from [GKS17, Table 5].

As seen from Table 3, our construction has no advantage over (and is even worse than) the implementations of Kiss et al.s and Günther et al.'s for small circuits (n up to up to a few hundreds). But with the growth of circuit size, our construction starts to outperform the rest by a few percentage points. Curiously, in the case of SHA-256, the number of AND gates in our 4-way universal circuit is about $1.39 \cdot 10^7$ and Valiant' 4-way is $1.46 \cdot 10^7$. Their ratio is about 0.952, which is very close to 18/19 and therefore confirms our analysis that the constant factor (in the of number of AND gates, as well as the size of the EUG) has been improved from 4.75 to 4.5. Even taking into consideration the optimization (e.g., using the hybrid of 2-way and 4-way) [GKS17], our construction still has its advantage [AGKS19].

4 A Lower Bound on Circuit Size in Valiant's Framework

Our search algorithm is intended for arbitrary k-way supernodes, but the time complexity is too large to be practical for $k \geq 5$. In this section, we aim to find a lower bound (for all k's) on the size of Valiant's EUG (and UC), which is in turn based on that of the supernode.

[8] Recall that the number of AND gates of Lipmaa et al.'s circuits (Fig 7) remains the same with Valiant's 4-way construction since it saves only XOR gates, so the comparison does not include the Lipmaa et al.'s work.

4.1 A Generic Lower Bound on Circuit Size

Valiant showed a generic bound $\Omega(n \log n)$ to argue the asymptotic optimality of his construction [Val76], where constant behind Ω could be extracted from Wegener's book [Weg87, Theorem 8.1] by carefully checking its (somewhat nested) proof. We mention that this could be seen directly from a counting argument which we informally sketch below (and stress that it is not a proof and refer to [Weg87] for formal details). That is, consider an arbitrary $\mathsf{C}^g_{s,t}$ with inputs and gates topologically sorted (inputs followed by gates), i.e., $in_1, \cdots, in_n, g_{s+1}, \cdots, g_{n=s+g}$, and assume that they are c different symmetric gates (e.g., XOR and AND) of fan-in 2. Then, for each g_i ($i > s$) there are $\binom{i-1}{2}$ choices of inputs and therefore the logarithm of the cardinality:

$$\log|\mathsf{C}^g_{s,t}| \geq \log\Big(\frac{(n!)^2 \cdot (\frac{c}{2})^{n-s}}{n!}\Big) = n \log n - O(n) \ ,$$

where the $n!$ in the denominator accounts for that the topological sorting of inputs and gates are not unique (but up to the permutation of the nodes). Finally, the input length of the universal circuit is lower bounded by $\log|\mathsf{C}^g_{s,t}|$ and so is the size of UC. Apparently, there are some loose steps, such as the order of gates cannot be arbitrarily permuted but this does not affect the lower bound by a factor of more than 2. A major lossy step is that we only require the size of the UC (of fan-in 2) to be at least the same as that of the input (in order for every input to contribute to the output the UC must be a connected DAG). In fact, a UC would need much more gates than its inputs to accomplish the simulation, and therefore additional knowledge about a specific UC framework could be helpful to improve this generic bound.

There remains a substantial gap between the constant factor in the generic (not specific to Valiant's UC framework) lower bound (i.e., 1) and that of known constructions (19 for Valiant's UC [Val76] and reduced to 18 in this work). Further, the generic bound sheds no light on the lower bound on the size of Valiant's EUG. Motivated by that most existing UCs are constructed under Valiant's framework, we aim to find a better (much lifted) lower bound on the size of EUG (and UC) in Valiant's framework.

4.2 Size of k-way Supernode

Recall that sizes of EUG and UC can both be based on that of the supernode (see Equation 4 and Equation 6 reproduced below):

$$|\mathsf{EUG}_2(n)| = \frac{2|\mathsf{SN}(k)|}{k \log k} n \log n - O(n) \ ,$$

$$|\mathsf{UC}^g_{s,t}| = \frac{8|\mathsf{SN}(k)|}{k \log k} n \log n - O(n) \ ,$$

where the smaller term $O(n)$ is often omitted. Thus, our task is to lower bound $\frac{2|\mathsf{SN}(k)|}{k \log k}$ by some constant. Recall that F_k denotes the number of all the k-way pole-complete graphs (Definition 3.1). We use the following lemma to reduce our task to the approximation of F_k.

Lemma 4.1 $|\mathsf{SN}(k)| \geq \lceil \log(F_k) + k \rceil$.

Proof.

Every pole-complete graph G can be configured (by setting the control bits) to be edge-embedded into $\mathsf{SN}(k)$, and the common nodes should be switching gates. Therefore, for an $\mathsf{SN}(k)$ we need set the control bits of its $|\mathsf{SN}(k)| - k$ common nodes to cater for all pole-complete graphs (amount to F_k), i.e., $2^{|\mathsf{SN}(k)|-k} \geq F_k$, where $|\mathsf{SN}(k)|$ is an integer. This completes the proof. □

$|\mathsf{EUG}_2(n)| = \frac{2|\mathsf{SN}(k)|}{k \log k} n \log n - O(n) \geq \frac{2\lceil \log(F_k)+k \rceil}{k \log k} n \log n - O(n)$

Our next job is to lower bound $g(k) \overset{\mathsf{def}}{=} \frac{2\lceil \log(F_k)+k \rceil}{k \log k}$ as a function of $k \in N^+$.

Table 4: The values of $\lceil \log(F_k) + k \rceil$ and $g(k)$ for $k < 100$.

k	2	3	4	5	...	68	69	70	...	98	99
$\lceil \log(F_k) + k \rceil$	5	11	17	23	...	755	768	782	...	1182	1197
$g(k) = \frac{2\lceil \log(F_k)+k \rceil}{k \log k}$	5	4.63	4.25	3.96	...	3.6478	3.6442	3.6453	...	3.6468	3.6477

4.3 A Guess for the Constant Factor

In order to lower bound $g(k)$, it would be ideal to give an approximation of F_k and then take the minimum over all k's. However, a general closed-form expression for F_k seems difficult. We further define $A_{i,k}$ in Definition 4.1 and give the relation between F_k and $A_{i,k}$ in Lemma 4.2. We also provide a recursion formula for $A_{i,k}$ in Lemma 4.3, which facilitates the computation of $A_{i,k}$ (by dynamic programming) for small values of i and k. With the above, we are able to compute $g(k)$ for k up to a few thousand (see Table 4 for values when $k < 100$). Based on the values computed, we have the guess that $g(k) > 3.644$, where $g(k)$ is monotonically decreasing for $k \leq 69$ and monotonically increasing for $k \geq 69$ with minimum $g(k) \approx 3.6442$ achieved at $k = 69$. The former (monotonic decreasing) statement is verified by computing all $g(k)$ for all $k \leq 69$ and a proof of the latter (monotonic increasing) is deferred to the next subsection.

Definition 4.1 *Let $A_{i,k}$ denote the number of ways to spread k different balls into i $(i \leq k)$ identical boxes with the condition that no boxes are empty.*

Lemma 4.2 $F_k = \sum_{i=1}^{k} (\frac{k!}{(k-i)!})^2 A_{i,k}$.

Proof. If $G = (V, E) \in \mathsf{DAG}_1(3k)$ is a k-way pole-complete graph, by Definition 3.1, we know that G can be regarded as a set of paths. It remains to sum up the numbers of pole-complete graphs for $1 \leq i \leq k$ paths: the number of ways to "put" k poles into i paths is $A_{i,k}$ by Definition 4.1, and there are $\frac{k!}{(k-i)!}$ ways to link i start-nodes (resp., end-nodes) to k inputs (resp., outputs) for these paths. Thus, $(\frac{k!}{(k-i)!})^2 A_{i,k}$ different pole-complete graphs for each value of i and we sum up (for $i = 1$ to $i = k$) to get the final result. □

Lemma 4.3 *1. $A_{1,k} = 1, \forall k \in \mathbb{N}^+$;*

2. $A_{i,k} = \sum_{j=0}^{k-i} \binom{k-1}{j} A_{i-1,k-j-1}$.

Proof. The first statement is trivial and we just need to prove the second one. Recall that in Definition 4.1 balls are all distinct while boxes are identical. We assume WLOG that ball #1 is in box #1, and let j be the number of other balls (in addition to ball #1) in box #1, where $j \leq k - i$ is required to make sure that no boxes are empty. After choosing these j balls ($\binom{k-1}{j}$ different choices), it remains to put the rest $k - j - 1$ balls into the remaining $i - 1$ boxes, which can be done in $A_{i-1,k-j-1}$ different ways by definition. □

We compute the values of $g(k)$ and other functions of k for k up to a few thousand, and list only partial results (up to $k = 99$) in Table 4 due to lack of space, from which we guess $g(k) > 3.644$ (recall that $g(69)$ is actually greater than 3.644). Note that it is tight at $k = 2$ ($g(2) = 5$) but not tight at $k = 4$ as $g(4)$=4.25 but the constant factor of our size optimal UC is 4.5.

4.4 The Lower Bound

We proceed to the proof of $g(k) = \frac{2\lceil \log(F_k)+k \rceil}{k \log k} > 3.644$ for $k \geq 69$. We give its proof in Lemma 4.4 but only for $k \geq 1478$, and gap (values of $g(k)$ for $70 \leq k \leq 1477$) is verified by computer. Note that there is nothing special with 1478, which is attributed to the loss of

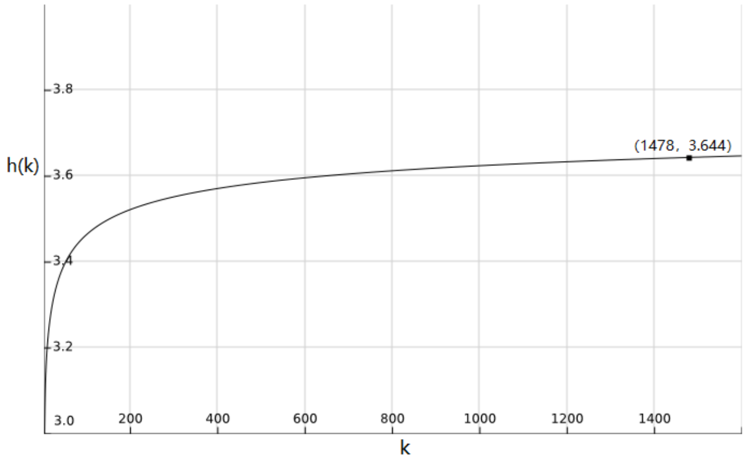


Figure 9: The graph of $h(k)$ as a function of k.

tightness by some inequality applied in its proof (such that 3.644 can only be obtained when $k = 1478$ in the right-hand of the inequality).

Lemma 4.4 $g(k) = \frac{2\lceil \log(F_k)+k \rceil}{k \log k} > 3.644$ *for all* $k \geq 1478$.

Proof. From Lemma 4.2, we have

$$F_k = \sum_{i=1}^{k} (\frac{k!}{(k-i)!})^2 A_{i,k} \geq \sum_{i=k-1}^{k} (\frac{k!}{(k-i)!})^2 A_{i,k} = (A_{k-1,k} + A_{k,k})(k!)^2 \ ,$$

and $A_{k,k} = 1, A_{k-1,k} = \binom{k}{2} = \frac{(k-1)k}{2}$ (Definition 4.1). Thus, $F_k \geq (\frac{(k-1)k}{2} + 1)(k!)^2$. It follows from Stirling's formula $k! \geq \sqrt{2\pi k}(\frac{k}{e})^k$ that

$$F_k \geq (2\pi k)\Big(\frac{(k-1)k}{2} + 1\Big)\left(\frac{k}{e}\right)^{2k} \ ,$$

and therefore

$$\begin{aligned} g(k) &\geq \frac{2\log(F_k) + k}{k \log k} \geq \frac{2\log(\pi k((k-1)k+2)(\frac{k}{e})^{2k}) + k}{k \log k} \\ &= 4 - \frac{(4\log e - 1)k - \log(\pi k((k-1)k+2))}{k \log k} \overset{\mathsf{def}}{=} h(k) \ , \end{aligned}$$

where by taking the derivative we know that $h(k)$ in the right-hand is monotonically increasing for $k \geq 2$, as also visualized in Figure 9, and the conclusion follows by finding the threshold T such that $h(k) \geq h(T) \approx 3.644$ for all $k \geq T$. By enumeration we find out $T = 1478$. Recall that values of $g(k)$ for $70 \leq k \leq 1477$ have been verified by computer. □

Combining Equation 4, Lemma 4.1 and Lemma 4.4, we have the following theorem:

Theorem 4.1 *We have the following lower bound on the size of* $\mathsf{EUG}_2(n)$*:*

$$|\mathsf{EUG}_2(n)| > 3.644 n \log n \ ,$$

for all sufficiently large n.

5 Concluding remarks

We revisit Valiant's graph theoretic approach to the construction of universal circuits, and show that its supernode can be improved in both size and depth, which yields more efficient universal circuits (with a more than 5% improvement). We give a lower bound on the size of UC to complement our explicit constructions, which reduces the gap between theory and practice of UCs.

Acknowledgments

Shuoyao Zhao is funded by the privacy-preserving computation project from PlatON Network. Yu Yu was supported by the National Natural Science Foundation of China (Grant numbers 61872236 and 61572192), the National Cryptography Development Fund (Grant number MMJJ20170209), and the Anhui Initiative in Quantum Information Technologies (Grant number AHY150100). Jiang Zhang is supported by the National Key Research and Development Program of China (Grant No. 2017YFB0802005, 2018YFB0804105), the National Natural Science Foundation of China (Grant Nos. 6160204661932019), and the Young Elite Scientists Sponsorship Program by CAST (2016QNRC001). We thank the reviewers of ASIACRYPT17 for their helpful comments.

References

[ABF$^+$17] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In *2017 IEEE Symposium on Security and Privacy (SP 2017)*, pages 843–862, 2017.

[AGKS19] Masaud Y. Alhassan, Daniel Günther, gnes Kiss, and Thomas Schneider. Efficient and scalable universal circuits. Cryptology ePrint Archive, Report 2019/348, 2019. https://eprint.iacr.org/2019/348.

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In *Advances in Cryptology - EUROCRYPT 2014*, pages 387–404, 2014.

[Att14] Nuttapong Attrapadung. Fully secure and succinct attribute based encryption for circuits from multi-linear maps. Cryptology ePrint Archive, Report 2014/772, 2014. https://eprint.iacr.org/2014/772.

[BBKL17] Osman Bicer, Muhammed Ali Bingol, Mehmet Sabir Kiraz, and Albert Levi. Towards practical pfe: An efficient 2-party private function evaluation protocol based on half gates. Cryptology ePrint Archive, Report 2017/415, 2017. https://eprint.iacr.org/2017/415.

[BFGH10] Debajyoti Bera, Stephen A. Fenner, Frederic Green, and Steven Homer. Efficient universal quantum circuits. *Quantum Information & Computation*, 10(1&2):16–27, 2010.

[CH85] Stephen A. Cook and H. James Hoover. A depth-universal circuit. *SIAM J. Comput.*, 14(4):833–839, 1985.

[FGP14] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS 2014)*, pages 844–855, 2014.

[FVK$^+$15] Ben A. Fisch, Binh Vo, Fernando Krell, Abishek Kumarasubramanian, Vladimir Kolesnikov, Tal Malkin, and Steven M. Bellovin. Malicious-client security in blind seer: A scalable private DBMS. In *2015 IEEE Symposium on Security and Privacy (SP 2015)*, pages 395–410, 2015.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *Advances in Cryptology - CRYPTO 2013, Part II*, pages 479–499, 2013.

[GGH$^+$16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM J. Comput.*, 45(3):882–929, 2016.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure attribute based encryption from multilinear maps. *IACR Cryptology ePrint Archive*, 2014:622, 2014.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology - EUROCRYPT 2013*, pages 626–645, 2013.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *Advances in Cryptology - CRYPTO 2010*, pages 155–172, 2010.

[GKS17] Daniel Günther, Ágnes Kiss, and Thomas Schneider. More efficient universal circuit constructions. In *Advances in Cryptology - ASIACRYPT 2017, Part II*, pages 443–470, 2017.

[GP81] Zvi Galil and Wolfgang J. Paul. An efficient general purpose parallel computer. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC 1981)*, pages 247–262, 1981.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. *J. ACM*, 62(6):45:1–45:33, 2015.

[HKK$^+$14] Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. Amortizing garbled circuits. In *Advances in Cryptology - CRYPTO 2014, Part II*, pages 458–475, 2014.

[KS08a] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. pages 486–498, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[KS08b] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography and Data Security, 12th International Conference, FC 2008, Revised Selected Papers*, pages 83–97, 2008.

[KS16] Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. In *Advances in Cryptology - EUROCRYPT 2016, Part I*, pages 699–728, 2016.

[LMS16] Helger Lipmaa, Payman Mohassel, and Saeed Sadeghian. Valiant's universal circuit: Improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017, 2016. https://eprint.iacr.org/2016/017.

[LR15] Yehuda Lindell and Ben Riva. Blazing fast 2pc in the offline/online setting with security for malicious adversaries. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS 2015)*, pages 579–590, 2015.

[Mey83] Friedhelm Meyer auf der Heide. Efficiency of universal parallel computers. In *Theoretical Computer Science*, pages 221–241, 1983.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *Proceedings of the 13th USENIX Security Symposium*, pages 287–302, 2004.

[MS13] Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *Advances in Cryptology - EUROCRYPT 2013*, pages 557–574, 2013.

[MSS14] Payman Mohassel, Seyed Saeed Sadeghian, and Nigel P. Smart. Actively secure private function evaluation. In *Advances in Cryptology - ASIACRYPT 2014, Part II*, pages 486–505, 2014.

[PKV$^+$14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steven M. Bellovin. Blind seer: A scalable private DBMS. In *2014 IEEE Symposium on Security and Privacy (SP 2014)*, pages 359–374, 2014.

[Sad15] S. S. Sadeghian. *New Techniques for Private Function Evaluation.* PhD thesis, University of Calgary, 2015.

[TS15] Stefan Tillich and Nigel Smart. Circuits of basic functions suitable for MPC and FHE, 2015.

[Val76] Leslie G. Valiant. Universal circuits (preliminary report). In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing (STOC 1976)*, pages 196–203, 1976.

[Weg87] Ingo Wegener. The complexity of boolean functions. ECCC BOOKS, LECTURES AND SURVEYS, 1987.

[ZCSH18] Ruiyu Zhu, Darion Cassel, Amr Sabry, and Yan Huang. nanoPI: Extreme-scale actively-secure multi-party computation. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS 2018)*, pages 862–879, 2018.

[Zha18a] Shuoyao Zhao. The c++ source code of our 4-way uc implementation, 2018.

[Zha18b] Shuoyao Zhao. A proof for that the graph in Figure 1 is a 4-way supernode. shared in a double-blind way (registration /log-in not required for upload and download), 2018. https://www.filedropper.com/sn-proof.

[Zim15] Joe Zimmerman. How to obfuscate programs directly. In *Advances in Cryptology - EUROCRYPT 2015, Part II*, pages 439–467, 2015.

A Proofs omitted in the main body

A.1 Proof of Theorem 2.1

To prove the graph in Figure 4 is an $\mathsf{EUG}_1(n)$, we need to prove that any $\mathsf{DAG}_1(n) = (V, E)$ can be edge-embedded into it. At first, we sort the nodes of a given $\mathsf{DAG}_1(n)$ in their topological order: $V_1, V_2, \ldots, V_n$. And the edge-embed mapping ϱ can be defined as: $\varrho(V_i)$ is the i-th pole of the supernodes from top to bottom, or formally, the (i mod k)-th pole of $\mathsf{SN}(k)_{\lceil \frac{i}{k} \rceil}$. For each node V_i in the $\mathsf{DAG}_1(n)$, it may have a precursor-node (denote by V_i^{pre}) and a successor-node (denote by V_i^{suc}). Then we assign the $[V_i]_{in}$-th input and the $[V_i]_{out}$-th output of $\mathsf{SN}(k)_{\lceil \frac{i}{k} \rceil}$ ($in_{[V_i]_{in}}^{\lceil \frac{i}{k} \rceil}$ and $out_{[V_i]_{out}}^{\lceil \frac{i}{k} \rceil}$) to V_i to make sure that $[V_i]_{in} = [V_i^{pre}]_{out}, [V_i]_{out} = [V_i^{suc}]_{in}$ and no inputs and outputs of supernodes are reused. The method for assignment can be find in [GKS17]. At last, for every edge $(V_i, V_j) \in E$ ($i < j$ due to the topological sorting), we give an edge-disjoint path from $\varrho(V_i)$ to $\varrho(V_j)$ as follow. Due to $V_i^{suc} = V_j$ and $V_j^{pre} = V_i$, we know that $[V_i]_{out} = [V_j]_{in}$, which means $out_{[V_i]_{out}}^{\lceil \frac{i}{k} \rceil}$ and $in_{[V_j]_{in}}^{\lceil \frac{j}{k} \rceil}$ are both in the edge-universal graph: $\mathsf{EUG}_1(\lceil \frac{n}{k} \rceil - 1)_{[V_i]_{out}}$, so there is an edge-disjoint path from $out_{[V_i]_{out}}^{\lceil \frac{i}{k} \rceil}$ to $in_{[V_j]_{in}}^{\lceil \frac{j}{k} \rceil}$. As $\mathsf{SN}(k)_{\lceil \frac{i}{k} \rceil}$ is a supernode, there must be a edge-disjoint path from $\varrho(V_i)$ to $out_{[V_i]_{out}}^{\lceil \frac{i}{k} \rceil}$. Similarly, the edge-disjoint path from $in_{[V_j]_{in}}^{\lceil \frac{j}{k} \rceil}$ to $\varrho(V_i)$ can also be found. We connect these three paths to complete edge-embedding.