Valiant's Universal Circuit: Improvements, Implementation, and Applications

Helger Lipmaa[1], Payman Mohassel[2], and Saeed Sadeghian[3]

[1] University of Tartu
helger.lipmaa@gmail.com
[2] Yahoo Labs
pmohassel@yahoo-inc.com
[3] University of Calgary
sadeghian@gmail.com

Abstract. A Universal Circuit (UC) is a circuit that can simulate any circuit of a maximum size, given its description as input. In this work, we look back at Valiant's universal circuit construction from Valiant (STOC 1976). Although it yields asymptotically optimal UC, and has implications for important problems in cryptography such as "private function evaluation" (PFE) and "cryptographic program obfuscation", somewhat surprisingly, no implementations of the construction exist. We provide a more approachable description, improve its constant factors, and put forth the first complete implementation. We observe that our improved implementation of Valiant's UC performs better than estimated and in fact, is almost always smaller than UC construction of Kolesnikov and Schneider (FC 2008). The UC circuits generated by our implementation can be used for benchmarking MPC protocols, and provide a point of comparison for any future PFE. We also observe, for the first time, that the same construction can be adapted to yield size optimized *universal arithmetic circuit* (UAC).

Keywords: Universal Circuits, Universal Arithmetic Circuits, Secure Computation, Private Function Evaluation, Program Obfuscation

1 Introduction

A Universal Circuit UC_{g} is a circuit that can simulate any circuit $\mathcal{C}$ of maximum size g, given the description of $\mathcal{C}$ as input. Universal circuits had been used in the cryptography community for various applications. We review some of these applications and propose new ones, that can benefit from more efficient UC constructions and implementation.

Program Obfuscation. Universal circuits have been used in some of the recent work related to program obfuscation[GGH$^+$13,Zim15]. Garg et al. [GGH$^+$13] employ UC as part of their quest to construct universal branching programs which was used in their construction for a candidate indistinguishability obfuscation. A

more efficient UC construction improves their construction. Zimmerman[Zim15] suggests a solution for direct program obfuscation, by proposing an approach to obfuscate what is called a keyed program. A universal circuit can be observed as a keyed program for any circuit family. We believe that our universal arithmetic circuit can be used to improve their performance as our key size is quasilinear $\mathcal{O}(\mathsf{g} \log \mathsf{g})$, compared to their quadratic complexity.

Private Function Evaluation. Private Function Evaluation is a natural application as UC can be used to transform any MPC result to a PFE result. A general-purpose secure computation in any setting applied to a UC provides a PFE for the same setting. PFE based on UC has been studied in [KS08a]. This work improves and extends their results, by providing a more efficient UC construction, and suggesting an efficient universal arithmetic circuit, for improved arithmetic circuit PFE. In [MS13], the authors propose a general framework for private function evaluation protocol that can be instantiated with several concrete protocols for different settings. An extension of their work with security against malicious adversaries is given in [MSS14]. A main advantage of their work was that it was also applicable to arithmetic circuits with the same efficiency. The significance of their result can be observed by comparing it to the alternative approach of applying MPC to the only known universal arithmetic circuit with worst case complexity $\mathcal{O}(\mathsf{g}^5)$ [Raz08]. Our efficient UAC construction based on Valiant's UC, provide an alternative solution to this problem by reducing the complexity from $\mathcal{O}(\mathsf{g}^5)$ to $\mathcal{O}(\mathsf{g} \log \mathsf{g})$.

Actively secure Non-Interactive Secure Computation (NISC) [AMPR14] applied to Valiant's UC, provides a protocol for actively secure non-interactive PFE, which is not possible through the framework of [MS13,MSS14]. In particular the first message of the first party in their NISC which encodes the signal bits fed to the UC, can be seen as an obfuscation of the circuit and can be evaluated on multiple inputs of the second party. The only drawback compared to a full obfuscation scheme is that one more message from the first party to the second party is needed (per evaluation) to decode the output. In applications where this extra round of interaction is possible, this provides a more efficient alternative to obfuscation.

While the current state of the art custom PFE provides more efficient protocols compared to UC based PFE, UC based PFE might improve and become more feasible due to the following reasons: First, the various optimizations that are recently proposed for MPC[KS08b,KMR14,ZRE15] are making general 2PC more pracitcal and it is not obvious if their techniques can also be combined with custom PFE solutions. Second, UC based PFE offers easier integration inside secure computation implementations, since a universal circuit can be treated similarly to any other circuit.

Practical 2PC. Another interesting application of UC is in efficient batch execution for 2PC. [HKK$^+$14,LR15] proposed techniques to amortize the cost of maliciously secure Yao's garbled circuits in batch execution for the same function. In particular, their techniques demand that one uses the same circuit in all

executions. By applying their technique to a UC, we can enable batch execution for different circuits. It is interesting to study, at what stage, the use of batch execution with UC circuits, yields more efficient 2PC than executing 2PC on the original (but different) circuits and without batch execution techniques.

1.1 Our Contribution

Motivated by the above applications, we look back at the Valiant's universal circuit construction [Val76] from STOC 1976. Although the construction yields universal circuits with the best known asymptotic complexity, and has been employed in several recent cryptographic constructions/protocols, some details are unspecified, and somewhat surprisingly, no implementations of the construction exist.

We give a complete and modular description of Valiant's UC construction based on our understanding. Roughly, Valiant's UC involves finding a "universal graph", in which we can embed the graph representation of circuits of a certain maximum size. The circuit implementation of the universal graph, yields the universal circuit. We provide a top-down description of the construction to a universal graph where we recursively pack multiple graph nodes into "supernodes". Each step is described through a lemma with complete details and description of any missing steps from the original work [Val76]. We also demonstrate each step by applying it to a running example. Our intention is to make the construction more accessible to researchers, and hence facilitate efficiency improvements and software implementations for this important combinatorial object.

We propose a general and improved method for constructing supernodes, borrowing techniques from [KS08a] UC. Figure 1 shows our improved supernode for packing 4 graph nodes. Using this supernode, we obtains the most efficient variant of our UC with $18\mathfrak{s}\log\mathfrak{s}$ gates.

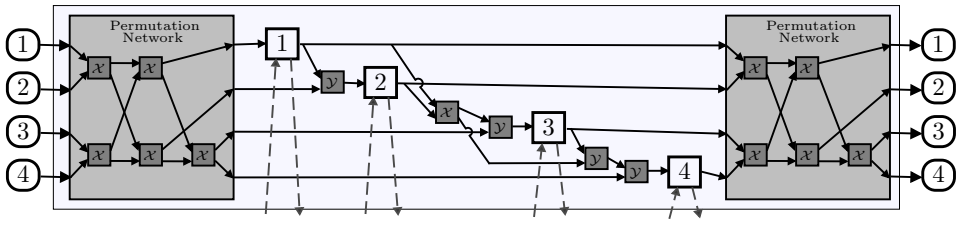

Fig. 1. 4-way split supernode construction ($k = 4$).

We also observe that a simple adaption of the same ideas to arithmetic circuits, yields the first size optimized Universal Arithmetic Circuit (UAC).

Finally, we provide the first implementation of Valiant's UC to our knowledge. Our implementation employs our improved building blocks to construct the

UC and is also capable of programming the UC given an input circuit. Previously, the upper bound based on the estimated complexity was the only point of comparison with Valiant's UC. Our implementation provides an opportunity to perform a more accurate comparison with UC construction of [KS08a]. Table 1 demonstrates a list of a few benchmark circuits [TS15] and their corresponding UC size using Valiant's UC and the [KS08a] UC construction. Note that Valiant's UC can be only applied to circuits with maximum fan-in/out of two. The UC construction of [KS08a] is claimed to produces smaller UCs compared to Valiant's UC for circuits with less than 5000 gates. We observe that our implementation for Valiant's UC produce smaller UCs than what is estimated by the theoretical complexity upper bound. In fact, our experiments identify 400 gates to be a more accurate breakpoint. This is when we only leak the size of the original circuit and need to consider an upper bound for the size of reduced fan-out circuit. If we leak the size of the circuit after the fan-out reduction, as our benchmark circuits in Table 1 suggest, Valiant's UC construction outperforms [KS08a] UC construction for almost all circuit sizes.

Table 1. A List of circuits and their corresponding UC size using Valiant's UC construction. $\mathtt{g}$ denotes the number of gates for original circuit, $\mathfrak{s}$ is the size of equivalent circuit with fan-out two plus the number of inputs, Valiant's UC is the size of Valiant's UC corresponding to each row's circuit found through implementation, and #AND denotes the number of AND gates in the Valiant's UC. [KS08a] UC is the size of UC computed using [KS08a] circuit complexity.

	$\mathtt{g}$	$\mathfrak{s}$	Valiant's $\mathsf{UC}(\mathfrak{s})$	#AND	[KS08a] $\mathsf{UC}(\mathtt{g})$
$\mathcal{X}$ switch	4	7	171	49	194
32bit comparator	300	365	41851	11257	102311
32bit adder	375	497	62231	16685	135968
64bit adder	759	985	141719	37901	329973
32bit multiplier	12374	17422	3913610	1039614	9669330
AES	33616	47663	11794323	3135833	31236200

2 Preliminaries and Notations

We denote the universal circuit which accepts circuit of maximum size $\mathtt{g}$ with $\mathtt{n}$ inputs and $\mathtt{m}$ outputs by $\mathsf{UC}_{\mathtt{g}}^{\mathtt{n},\mathtt{m}}$.

Definition 1 (Universal Circuit). *([Weg87])A circuit $\mathsf{UC}_{\mathtt{g}}^{\mathtt{n},\mathtt{m}}$ is called a universal circuit, if it contains $\mathtt{n}$ true input variables, $\mathtt{m}$ true output variables and $\mathtt{g}$ distinguished universal gates, such that for any circuit $\mathcal{C}$ of size $\mathtt{g}_c \leq \mathtt{g}$, there is a configuration for UC such that the i-th distinguished universal gate of UC computes the same function as the i-th gate of $\mathcal{C}$ for $1 \leq i \leq \mathtt{g}_c$.*

Universal boolean circuits are proposed by [Val76,KS08a]. The situation is not the same for arithmetic circuits. While depth universal arithmetic circuits

which aim to maintain the depth of original circuit, are proposed by Raz [Raz08], to our best knowledge, there exist no size optimized universal circuit construction for arithmetic circuits.

The nature of universal boolean circuit construction is a switching network where the basic building blocks are switches with two inputs and outputs. For ease of presentation we follow the notation similar to previous works [KS08a] and demonstrate the switches as follows. We denote the switches with two input and two output by $\mathcal{X}$. They accept a selection bit as a control signal and either act as buffer or swapper. We also denote the switches with two inputs and one output (similar to multiplexer with two inputs), by $\mathcal{Y}$. They also accept a selection bit, which determines the input that is transfered to output. Fig. 2 demonstrates the gate level implementation of $\mathcal{X}$ and $\mathcal{Y}$ switches which require 4 and 3 gates respectively. As can be observed each require only one AND gate. We sometime use the term gate level implementation when only binary gates are used in the specified implementation opposed to graph representation consisting of vertices/nodes.

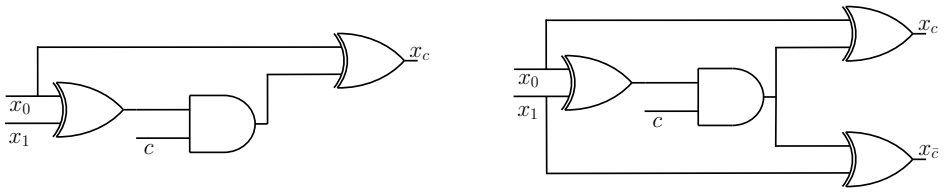

Fig. 2. Circuit implementation of $\mathcal{Y}$ Switch and $\mathcal{X}$ Switch

Edge-Universal Graphs. Valiant observed that Universal Circuits are closely related to the notion of edge-universal graphs as defined bellow.

Definition 2 (Edge-Universal Graphs). *Let* $\mathsf{DG}_k(\mathfrak{s})$ *(resp.,* $\mathsf{DAG}_k(\mathfrak{s})$*) be the set of directed (resp., directed acyclic) graphs with* $\mathfrak{s}$ *nodes and fan-in and fan-out* k*. 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* $\mathsf{DAG}_k(\mathfrak{s})$ *if it has distinguished poles* $p_1, \ldots, p_{\mathfrak{s}}$ *such that every* $G \in \mathsf{DAG}_{k'}(\mathfrak{s}')$*, with* $k' \leq k$ *and* $\mathfrak{s}' \leq \mathfrak{s}$*, 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*.*

We use $\mathsf{DAG}_k(\mathfrak{s})$ to denote directed acyclic graphs of size $\mathfrak{s}$, and fan-in/fan-out k. We use $\mathsf{EUG}_k(\mathfrak{s})$ (EUG for $\mathsf{DAG}_k(\mathfrak{s})$) to denote an edge-universal graph with $\mathfrak{s}$ distinguished pole, in which we can edge-embed any graph $G \in \mathsf{DAG}_{k'}(\mathfrak{s}')$, with $k' \leq k$ and $\mathfrak{s}' \leq \mathfrak{s}$.

When talking about complexity, we use $\mathsf{EUG}_k(\mathfrak{s})$ to denote the number of vertices in $\mathsf{EUG}_k(\mathfrak{s})$, and use $\mathsf{CircuitEUG}_k(\mathfrak{s})$ to denote the size of its circuit

equivalent. Note that to go back and forth between graph and circuit implementation, we only need to substitute switches with graph nodes and the other way around. To compute the gate complexity, we need to count $\mathcal{Y}$ switches as 3 gates, and $\mathcal{X}$ switches as 4 gates.

Matching in Bipartite Graphs Matching in bipartite graphs is an essential part of Valiant's UC construction. For any set S of vertices in graph G, the neighborhood set of S in G is defined to be the set of all vertices adjacent to vertices in S and is denoted by $N_G(S)$. For a bipartite graph G with bipartition (X, Y), we may wish to find a matching that saturates every vertex in X. Hall's theorem gives necessary and sufficient conditions for the existence of such a matching:

Theorem 1 (Hall's Theorem). *(Theorem 5.2 of [Bon76]) Let G be a bipartite graph with bipartition (X, Y). Then G contains a matching that saturates every vertex in X if and only if:*
$|N(S)| \geq |S|$ *for all* $S \subseteq X$.

Theorem 2 (Corollary to Hall's Theorem). *(Corollary 5.2 of [Bon76]) If G is a k-regular bipartite graph with $k > 0$, then G has a perfect matching.*

3 Valiant's UC Construction: A Full Description

In this section, we explain Valiant's construction for Universal Circuits, providing a more modular and clear description. Our intention is to make the construction more accessible to researchers, and hence facilitate efficiency improvements and software implementations for this important combinatorial object.

The construction can be naturally described in three, modular, top-bottom steps: (i) construct a Universal Circuit given an *edge-universal graph* (EUG) for graphs of fan-in/out two (a notion introduced by Valiant), (ii) construct an EUG for graphs of fan-in/out two given an EUG for graphs of fan-in/out one. (iii) construct an EUG for graphs of size $\mathfrak{s}$ and fan-in/out one given an EUG for graphs of size $\mathfrak{s}/k$ and fan-in/out one for some constant k. The last step, can be recursively repeated to build EUGs for arbitrary size graphs. These steps are shown in Fig.3.

Fig. 3. Top-down demonstration of Valiant universal circuit construction

We use the example circuit of figure 4 to describe how all the steps come together to build a complete universal circuit.

Next we discuss each step separately.

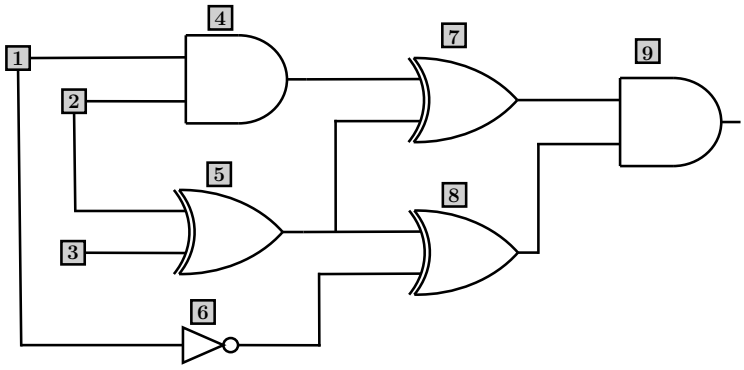

Fig. 4. Running example

3.1 From Edge-Universal Graphs to UC Circuits

Our goal is to design a Universal Circuit $\mathsf{UC}_{\mathsf{g}}^{\mathfrak{n},\mathfrak{m}}$. Valiant's universal circuit construction can be used for circuits with fan-in/fan-out limited to two which we denote by $\mathsf{UC}_{\mathsf{g},2}^{\mathfrak{n},\mathfrak{m}}$. It is possible to convert any circuit with arbitrary fan-in/out to a circuit with reduced fan-out. We first prove the following simple lemma. As a result of this lemma, we can focus on designing an EUG for $\mathsf{DAG}_2(\mathfrak{s})$.

Lemma 1. *Given an* $\mathsf{EUG}_2(\mathfrak{s})$ *for* $\mathsf{DAG}_2(\mathfrak{s})$ *where* $\mathfrak{s} = \mathfrak{n} + \mathsf{g}$ *with* $\mathfrak{s}$ *distinguished poles and* L *nodes, we can construct a universal circuit* $\mathsf{UC}_{\mathsf{g},2}^{\mathfrak{n},\mathfrak{m}}$ *of size at most* $9\mathfrak{s} + 4L$*, for circuits with fan-in/fan-out 2.*

Proof. We model the circuit as a graph $G_{\mathcal{C}}$ were 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 $\mathsf{DAG}_2(\mathfrak{s})$ with $\mathfrak{s} = \mathfrak{n} + \mathsf{g}$. From the definition of EUG, it is possible to embed any $\mathsf{DAG}_2(\mathfrak{s})$ such as $G_{\mathcal{C}}$ into the given $\mathsf{EUG}_2(\mathfrak{s})$, such that there are edge-disjoint paths between any two distinguished poles. We proceed by constructing a circuit equivalent of $\mathsf{EUG}_2(\mathfrak{s})$ as follows. Each distinguished pole can be implemented by a universal gate for 16 possible functions from $\{0,1\}^2 \to \{0,1\}$ i.e., $(x_1, x_2, c_1, c_2, c_3, c_4) \mapsto c_1x_1x_2 + c_2x_1\bar{x}_2 + c_3\bar{x}_1x_2 + c_4\bar{x}_1\bar{x}_2 = (x_1 \wedge (c_1 \wedge x_2 \oplus c_2 \wedge \bar{x}_2)) \oplus (\bar{x}_1 \wedge (c_3 \wedge x_2 \oplus c_4 \wedge \bar{x}_2))$. Each assignment to control signals $c_1c_2c_3c_4$ (concatenated for ease of presentation) corresponds to one of the possible functions, for example 1000 simulates the AND gate and 0110 simulates the XOR gate. The universal gate for the first $\mathfrak{n}$ distinguished poles are programmed to implement a constant binary value equal to the corresponding input to the circuit. Assigning 0000 and 1111 to $c_1c_2c_3c_4$ simulate zero and one. The $(\mathfrak{n} + i)$-th distinguished pole is programmed to implement the i-th topological gate of circuit being programmed. Circuit implementation of a universal gate requires 9 binary gates as shown in Fig.5. The cost of implementing all the universal gates is $9(\mathfrak{n} + \mathsf{g}) = 9\mathfrak{s}$ binary gates. Each of the other L nodes beside distinguished poles can be implemented by a $\mathcal{X}$ or $\mathcal{Y}$ switch which requires a maximum of $4L$ gates in total. The resulting circuit is a universal circuit for

circuits with $\mathsf{DAG}_2(\mathfrak{s})$ graph representation as one can find the edge-embedding corresponding to any $\mathsf{DAG}_2(\mathfrak{s})$, in $\mathsf{EUG}_2(\mathfrak{s})$, and configure the switches to implement the edge-disjoint paths and therefore simulate the topology of the given circuit. Universal gates ensure that each distinguished pole implements the functionality of the corresponding input/gate. □

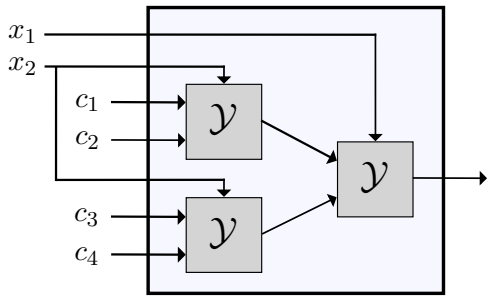

Fig. 5. Implementation of universal gate

Getting back to our example, we need to start by reducing the fan-out of gates in the circuit to 2. For this example the circuit already meets our criteria. Each number denotes a node in the $\mathsf{DAG}_2(\mathfrak{s})$. We need to find an EUG for this graph.

3.2 Edge-Universal Graphs: From EUG for $\mathsf{DAG}_1(\mathfrak{s})$ to EUG for $\mathsf{DAG}_k(\mathfrak{s})$

Next we show how to construct EUG for $\mathsf{DAG}_k(\mathfrak{s})$ from EUG for $\mathsf{DAG}_1(\mathfrak{s})$. This naturally enables us to use EUGs for $\mathsf{DAG}_1(\mathfrak{s})$, to construct an EUG for $\mathsf{DAG}_2(\mathfrak{s})$, which is what we need in lemma 1 to construct UC for fan-in/fan-out 2 circuits.

Lemma 2 (Lemma 2.1 of [Val76]). *For any* $\mathsf{DAG}_k(\mathfrak{s}) = (V, E)$, $E = \cup_{i=1}^{k} E_i$ *is the union of* k *disjoint set* E_i*, such that* $(V, E_i) = \mathsf{DAG}_1(\mathfrak{s})$.

Proof. We represent the $\mathsf{DAG}_k(\mathfrak{s}) = (V, E)$, where $V = \{v_1, \ldots, v_{\mathfrak{s}}\}$ by a bipartite graph $\mathsf{BG}_k = (V', E')$, where $V' = \{v_1, \cdots, v_{\mathfrak{s}}, v'_1, \cdots, v'_{\mathfrak{s}}\}$, and for each $(v_i, v_j) \in E$, we have $(v_i, v'_j) \in E'$. If required we add dummy edges to E' to make BG_k a k-regular bipartite graph. Given $\mathsf{BG}_k(\mathfrak{s})$, as a result of corollary to Hall's theorem given in theorem 2, one can find a perfect matching E_1, which covers all vertices with fan-out k or fan-in k (with the dummy edges removed from E_1). We then remove the edges in the matching E_1 from $\mathsf{DAG}_k(\mathfrak{s})$ to get $\mathsf{DAG}_{k-1}(\mathfrak{s})$. By induction we can find $\{E_1, \cdots, E_k\}$ whose union gives E. □

Lemma 3. *An* EUG *for* $\mathsf{DAG}_2(\mathfrak{s}) = (V, E)$*, can be constructed from two instances of* EUG *for* $\mathsf{DAG}_1(\mathfrak{s})$.

Proof. Using Lemma 2 we find E_1 and E_2 whose union produces E. Each (V, E_i) is a $\mathsf{DAG}_1(\mathfrak{s})$, therefore we can embed each in a separate EUG for $\mathsf{DAG}_1(\mathfrak{s})$. We merge the distinguished poles of the two $\mathsf{EUG}_1(\mathfrak{s})$, inside a single distinguished pole with each $\mathsf{EUG}_1(\mathfrak{s})$, providing one input to distinguished pole. This complete the $\mathsf{EUG}_2(\mathfrak{s})$ construction. □

Lemma 3 enables us to use two instances of $\mathsf{EUG}_1(\mathfrak{s})$ to construct $\mathsf{EUG}_2(\mathfrak{s})$ for our example. We first need to apply Lemma 2 to reduce $\mathsf{DAG}_2(9)$ representation of our example circuit to two $\mathsf{DAG}_1(9)$ as shown in figure 6. Then by merging the distinguished poles of the two $\mathsf{EUG}_1(9)$, one per each of the two graphs, and have each EUG provide one of the inputs to each pole, we get the desired $\mathsf{EUG}_2(9)$.

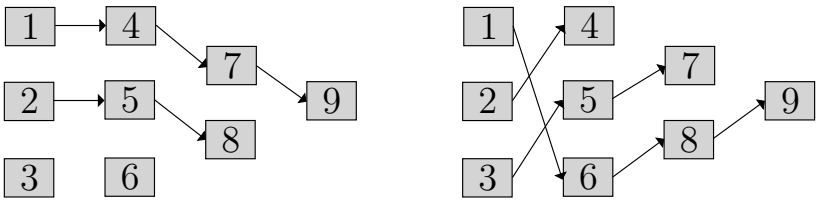

Fig. 6. Application of Lemma 2.1 to the running example

3.3 Edge-Universal Graphs for Fan-in/out One DAGs

So far, we have showed how to reduce construction of $\mathsf{UC}^{\mathtt{n},\mathtt{m}}_{\mathtt{g},2}$ to designing EUG for $\mathsf{DAG}_1(\mathfrak{s})$. We now show how to recursively reduce the problem to smaller sizes, until we get to the size for which we have an efficient EUG construction. We show each step of recursion by reducing EUG for $\mathsf{DAG}_1(\mathfrak{s})$ to EUG for $\mathsf{DAG}_1(\mathfrak{s}/k)$. For this reduction we need an essential building block called *Supernode* (see Definition 3). Informally, a supernode is a graph that enables us to look at certain number of nodes as one. Naturally, if we pack k nodes from a $\mathsf{DAG}_1(\mathfrak{s})$, we end up with a supernode with fan-in/out of k. Notice that since the nodes that are packed together are independent, there needs to be edge-disjoint paths between the inputs to the corresponding distinguished nodes inside the supernode. The same thing goes for the output of distinguished nodes. It might be the case that the input to distinguished nodes is from a previous distinguished node instead of the supernode inputs. There shoudl be edge-disjoint paths between any input, or output from previous distinguished nodes to any specific distinguished node. Figure 7 the graph for packing two nodes into a supernode. The two distinguished nodes (internal nodes) are the nodes that we intend to group together. There is a path from any of the two inputs to the first distinguished node through the vertex v_1. The input to the second distinguished node is selected from the input that was not routed to the first distinguished node and the output of

first distinguished node through v_2. Finally, the v_3 in the output enables any permutation of distinguished nodes' to be connected to the supernode outputs through edge-disjoint paths.

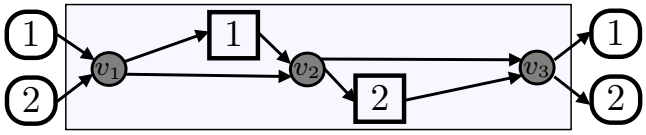

Fig. 7. 2-way split supernode construction ($k = 2$).

Definition 3 (Supernode). *A k-way split supernode $\mathcal{SN}(k)$ is an edge-universal graph with k nodes $\{in_1, \cdots, in_k\}$ marked as input, k nodes $\{out_1, \cdots, out_k\}$ marked as output, and k distinguished nodes $D = \{D_1, \cdots, D_k\}$, such that any graph $G = (V, E) \in DAG_1(3k)$, with $V = \{in_1, \cdots, in_k\} \cup \{D_1, \cdots, D_k\} \cup \{out_1, \cdots, out_k\}$, and for any edge $e = (v_1, v_2) \in E$:*

- *If $v_1 \in \{in_1, \cdots, in_k\}$ then $v_2 \in D$*
- *If $v_2 \in \{out_1, \cdots, out_k\}$ then $v_1 \in D$*
- *$v_1 \notin \{out_1, \cdots, out_k\}$*
- *$v_2 \notin \{in_1, \cdots, in_k\}$*

Can be edge embeded into $\mathcal{SN}(k)$.

We denote the number of binary gates required to implement the k-way split supernode by $\mathsf{Circuit}\mathcal{SN}(k)$.

Lemma 4. *Given an* EUG *for* $\mathsf{DAG}_1(\lceil \frac{\mathfrak{s}}{k} \rceil - 1)$ *with L nodes and a k-way supernode construction with size $\mathcal{SN}(k)$, we construct an* EUG *for* $\mathsf{DAG}_1(\mathfrak{s})$ *with at most* $\lceil \frac{\mathfrak{s}}{k} \rceil \mathcal{SN}(k) + kL$ *nodes.*

Proof. We start from the first node in topological order, and group each set of consecutive k nodes into a supernode using the given k-way split supernode. The resulting graph with each supernode seen as a single node, is $\mathsf{DAG}_k(\lceil \frac{\mathfrak{s}}{k} \rceil)$ as the maximum fan-in/out increases to k. Using Lemma 2, we break $\mathsf{DAG}_k(\lceil \frac{\mathfrak{s}}{k} \rceil)$ to k instances of $\mathsf{DAG}_1(\lceil \frac{\mathfrak{s}}{k} \rceil)$. In order to construct EUG on top of supernodes, we need to deal with two distinguished nodes per each supernode, one for the input and one for the output. This requires EUG for $\mathsf{DAG}_1(2\lceil \frac{\mathfrak{s}}{k} \rceil)$. Instead we only need an EUG for $\mathsf{DAG}_1(\lceil \frac{\mathfrak{s}}{k} \rceil - 1)$ to implement this, if we have the distinguished nodes corresponding to the output of i-th supernode and the input of $(i+1)$-th supernode be merged. Figure 11 demonstrates an example of how EUG for $\mathsf{DAG}_2(9)$ can be reduced to EUG for $\mathsf{DAG}_1(4)$.

As a result of this optimization it is required to perform some adjustments to convert $\mathsf{DAG}_1(\lceil \frac{\mathfrak{s}}{k} \rceil) = (V, E)$ to $\mathsf{DAG}_1(\lceil \frac{\mathfrak{s}}{k} \rceil - 1) = (V', E')$. For any edge

$(u, v) \in E$, if $v > u+1$ we add $(u, v-1) \in E'$. If $v = u+1$, then no edge is added, though the output of u-th supernode is passed through the u-th distinguished node to the input of v-th supernode. For the case $u = v$, no edge is required to be added as it corresponds to a connection inside the supernode and does not need to be part of $\mathsf{DAG}_k(\lceil \frac{\mathfrak{s}}{k} \rceil)$. □

In our example, we use Lemma 4 to go from $\mathsf{DAG}_1(9)$ to two instances of $\mathsf{DAG}_1(4)$. As demonstrated in figure 8, this involves packing every two consecutive node inside a 2-way split supernode of figure 14 to find $\mathsf{DAG}_2(5)$.

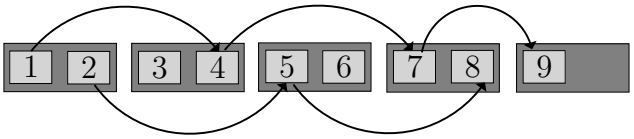

Fig. 8. From $\mathsf{DAG}_1(9)$ to $\mathsf{DAG}_2(5)$

We then apply a matching algorithm to reduce $\mathsf{DAG}_2(5)$ to two instances of $\mathsf{DAG}_1(5) = (V, E)$ as shown in figure 9. Finally, to find $\mathsf{DAG}_1(4)$ of next

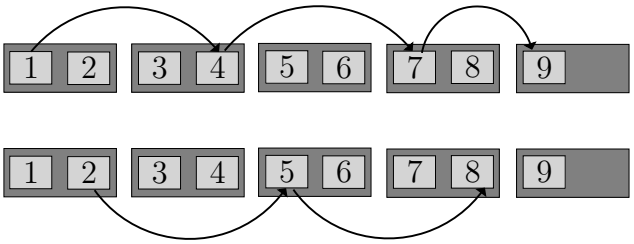

Fig. 9. From $\mathsf{DAG}_2(5)$ to two instances of $\mathsf{DAG}_1(5)$

recursion as shown in figure 10, for any edge $(u, v) \in E$, if $v > u + 1$ we add $(u, v - 1) \in E'$. Using the construction shown in figure 11, and have reduced $\mathsf{EUG}_1(9)$ to $\mathsf{EUG}_1(4)$.

3.4 Complexity of Valiant's Universal Circuit

We present the complexity analysis of Valiant's universal circuit. From Lemma 1, we need two copies of $\mathsf{EUG}_1(\mathfrak{s})$, where $\mathfrak{s} = \mathfrak{n}+\mathsf{g}$ to implement the UC. We denote the size of Valiant's UC with $\mathfrak{s}$ distinguished poles by $\mathsf{UC}(\mathfrak{s})$. Note that $\mathsf{UC}(\mathfrak{s})$ can handle circuits with $\mathfrak{n} + \mathsf{g} \leq \mathfrak{s}$ and maximum fan-in/out two.

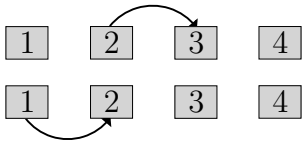

Fig. 10. From $\mathsf{DAG}_1(5)$ to $\mathsf{DAG}_1(4)$

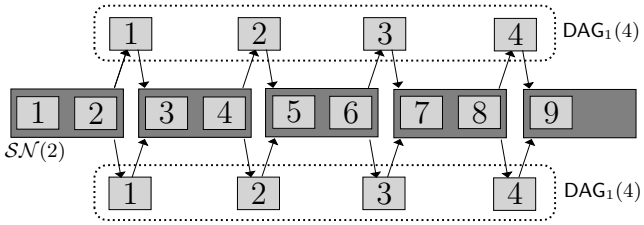

Fig. 11. Reduction from $\mathsf{EUG}_1(9)$ to $\mathsf{EUG}_1(4)$

$$\mathsf{UC}(\mathfrak{s}) = \mathsf{CircuitEUG}_2(\mathfrak{s})$$
$$\mathsf{EUG}_2(\mathfrak{s}) = 2\mathsf{EUG}_1(\mathfrak{s})$$

At each step, we reduce the problem of $\mathsf{EUG}(\mathfrak{s})$ to k instances of $\mathsf{EUG}\left(\left\lceil \frac{\mathfrak{s}}{k} \right\rceil - 1\right)$ using Lemma 4. This reduction requires using a k-way split supernode per each k consecutive nodes, except for the last supernode, which does not need the output distinguished nodes. The reduction cost would be $\left\lceil \frac{\mathfrak{s}}{k} \right\rceil \mathcal{SN}(k) - k$. Following is the recursive cost function.

$$\mathsf{EUG}(\mathfrak{s}) = k\mathsf{EUG}\left(\left\lceil \frac{\mathfrak{s}}{k} \right\rceil - 1\right) + \left\lceil \frac{\mathfrak{s}}{k} \right\rceil \mathcal{SN}(k) - k$$

The worst case complexity happens when in every recursion $\mathfrak{s} = k\mathfrak{s}_1 + 1 \approx k\mathfrak{s}_1$.

$$\mathsf{EUG}(\mathfrak{s}) \leq k\mathsf{EUG}\left(\frac{\mathfrak{s}}{k}\right) + \frac{\mathfrak{s}}{k}\mathcal{SN}(k)$$

$$\mathsf{EUG}(\mathfrak{s}) \leq \frac{\mathcal{SN}(k)}{k}\mathfrak{s}\log_k \mathfrak{s} \rightarrow \mathsf{EUG}_2(\mathfrak{s}) \leq \frac{2\mathcal{SN}(k)\mathfrak{s}\log\mathfrak{s}}{k\log k}$$

$$\mathsf{UC}(\mathfrak{s}) = \mathsf{CircuitEUG}_2(\mathfrak{s}) \leq \frac{2\mathsf{Circuit}\mathcal{SN}(k)\mathfrak{s}\log\mathfrak{s}}{k\log k}$$

4-way split supernode is the most efficient supernode suggested in [Val76], with 19 nodes:

$$\mathsf{EUG}_2(\mathfrak{s}) \leq \frac{2 \times 19\mathfrak{s}\log\mathfrak{s}}{4\log 4} = (19/4)\mathfrak{s}\log\mathfrak{s}$$

Each graph node can be implemented using one $\mathcal{X}$ switch:

$$\mathsf{UC}(\mathfrak{s}) \leq 4\mathsf{EUG}_2(\mathfrak{s}) = 19\mathfrak{s}\log\mathfrak{s}$$

4 Extension to Universal Arithmetic Circuits

We show how the graph abstraction of Valiant's UC, enables us to use the same EUG construction to construct universal arithmetic circuits (UAC). We observe that Lemma 1 is the (only) part that links the graph construction to its circuit equivalent. Thus, we only need to provide arithmetic circuit equivalents of Lemma 1.

Theorem 3. *Given an* $\mathsf{EUG}_2(\mathfrak{s})$ *for* $\mathsf{DAG}_2(\mathfrak{s})$ *where* $\mathfrak{s} = \mathfrak{n}+\mathfrak{g}$ *with* $\mathfrak{s}$ *distinguished poles and* L *nodes, we can construct a universal arithmetic circuit* $\mathsf{UAC}^{\mathfrak{n},\mathfrak{m}}_{\mathfrak{g},2}$ *of size at most* $O(1)\mathfrak{s} + 5L$*, for circuits with fan-in/fan-out 2.*

Proof. The proof is similar to Lemma 1, we only need to implement $\mathsf{EUG}_2(\mathfrak{s})$ with arithmetic operations instead of binary gates. The implementation of poles is slightly different, as we also need to deal with arbitrary inputs for input gates. We need 7 different functions: $(x_0, x_1, -x_0, -x_1, x_0 + x_1, x_0.x_1, c)$, where c is the arbitrary input. To select one of the seven functions we need 3 selection bits (c_2, c_1, c_0). Our implementation which requires $\mathcal{O}(1)$ arithmetic operations, is as follows:

$$(c_0, c_1, c_2, c, x_0, x_1) \rightarrow (1 - c_0)\left[(1 - c_1)\left(x_0 + x_1\right) + c_1\left((1 - c_2)x_0 + c_2 x_1\right)\right]$$

$$+c_0\left[(1 - c_1)((1 - c_2)(x_0 x_1) + c_2(c)) + c_1((1 - c_2)(-x_0) + c_2(-x_1))\right]$$

Each of the other L nodes beside distinguished poles can be implemented by a $\mathcal{X}$ or $\mathcal{Y}$ switch. The $\mathcal{X}$ switch functionality is defined based on arithmetic operations $(\cdot, +, -)$ as follows:

$$(c, x_0, x_1) \rightarrow ((1 - c)x_0 + cx_1, cx_0 + (1 - c)x_1) \ ,$$

Assuming negation can be done for free, this can be done using 5 arithmetic gates, by computing cx_0, cx_1, $cx_0 - cx_1$ and two additions. □

The above construction is an efficient UAC with the same asymptotic complexity as Valiant's UC for boolean circuits giving the best size optimized UAC to our knowledge.

5 A General Design for Supernodes

From the complexity analysis of section 3.4 it can be observed that supernode complexity dictates the concrete efficiency of Valiant's construction. Thus any improvement to the size of supernode circuits, directly affects the constant factors of the construction. Motivated by the importance of more compact supernodes, we study a general design for k-way split supernodes. Based on this general design, we derive improved k-way split supernodes when $k = 2$ and $k = 4$.

5.1 k-way Split Supernode From $\mathsf{EUG}_1(k)$

We show how to build k-way split supernodes given an EUG for $\mathsf{DAG}_1(k)$. From the definition of a supernode, we need to map any permutation of inputs to the distinguished nodes, and similarly map any permutation of distinguished nodes to output. The most efficient circuit for such purpose is a permutation network[Wak68]. An input to each distinguished node can also be the output of any previous distinguished node. This can be directly solved using an $\mathsf{EUG}_1(k)$. We need an extra $\mathcal{Y}$ switch per each distinguished node (except for the first one), to select between the input from EUG and the input from supernode inputs. Figure 12 shows this construction.

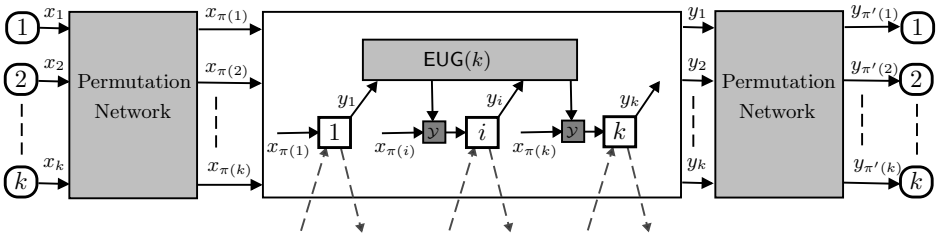

Fig. 12. k-way split supernode.

Complexity of k-way split supernode consists of $\mathsf{EUG}_1(k)$, two permutation networks with k inputs, k distinguished nodes, and $k-1$, $\mathcal{Y}$ switches to select from $x_{\pi(i)}$ and the input from EUG. Following is the supernode complexity in terms of number of nodes ($\mathcal{SN}$) and number of gates ($\mathcal{SN}_g$):

$$\mathcal{SN}(k) = (2(k\log k - k + 1) + k)\,\mathcal{X} + (k-1)\mathcal{Y} + \mathsf{EUG}_1(k) = 2k\log k + 1 + \mathsf{EUG}_1(k)$$

$$\mathsf{Circuit}\mathcal{SN}(k) = 8k\log k - k + 5 + \mathsf{CircuitEUG}_1(k)$$

This implies any supernode construction based on the above design cannot lead to UC construction with constant factor better than *16* due to the two permutation networks.

$$\mathsf{Circuit}\mathcal{SN}(k) > 8k\log k \rightarrow \mathsf{UC}(\mathfrak{s}) > \frac{2 \times 8k\log k \times \mathfrak{s}\log\mathfrak{s}}{k\log k} \rightarrow \mathsf{UC}(\mathfrak{s}) > 16\mathfrak{s}\log\mathfrak{s}$$

5.2 Design of $\mathsf{EUG}_1(k)$

One may try to design EUG for different sizes by hand. While we show improvements to supernode construction for small values of k ($k = 2$ and $k = 4$), we found it to be challenging for larger values of k to design efficient EUG by hand.

With this intuition, we investigate the efficiency of instantiating supernodes with a general construction. One may consider the recursive use of Valiant EUG

construction for smaller sizes inside supernode construction. If we assume the constant factor of construction to be $t(\approx 9)$, we have:

$$\mathsf{Circuit}\mathcal{SN}(k) = 8k\log k - k + 5 + \frac{t}{2}k\log k$$

Substituting this in the UC complexity gives the following, which is obviously less efficient since $t < t + 16$:

$$\mathsf{Circuit}\mathcal{SN}(k) \approx \left(8 + \frac{t}{2}\right) k\log k$$

$$\rightarrow \mathsf{UC}(\mathfrak{s}) = \frac{(16+t)k\log k \times \mathfrak{s}\log\mathfrak{s}}{k\log k} = (16+t)\mathfrak{s}\log\mathfrak{s}$$

Our second choice is from observing EUG circuit to be essentially a universal circuit, which is restricted to circuits with DAG_1 graph representation. With this intuition we turn back to UC construction of [KS08a]. This construction has been claimed to produce more compact UCs for smaller circuits comparing to Valiant. This gave hope for possible improvements by instantiating the EUG circuit inside the supernodes by UC construction of [KS08a]. The main building block in [KS08a] construction is what they call universal block, and can be seen as a universal circuit with inputs and outputs organized such that inputs with index $2i-1, 2i$ are the inputs to i-th universal gate, and the i-th output comes from i-th universal gate.

This match our requirement for EUG, except that we only require fan-in/out one in contrast to [KS08a] construction which can handle fan-in two and arbitrary fan-out. We design a modified and more efficient version of their universal block for our purpose, DAG_1. Figure 13 demonstrates the modified version of their recursive universal block construction with similar components. We outline the differences: In contrast to their construction, we only deal with one input per each universal gate(distinguished node for our purpose) which reduces the input size of $U_{k/2}$, from k to $k/2$. Furthermore, we avoid the use of selection block as we do not need to deal with replication or deletion of elements. Permutation networks ($P_{k/2}$) are used instead. Similarly, $\mathcal{M}_{k/2}$ consists of $k/2$ parallel $\mathcal{Y}$ switches.

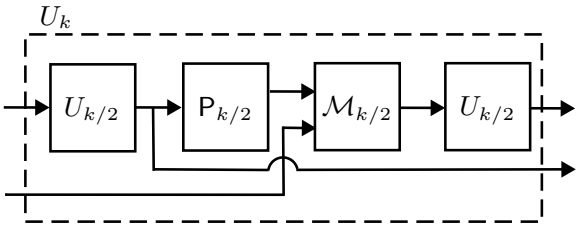

Fig. 13. EUG construction based on U_k object of [KS08a]. All lines carry $k/2$ wires.

Next, we analyze the complexity of our modified universal block. $\mathsf{U}(k)$ denotes the size of universal block for k distinguished nodes.

$$\mathsf{U}(k) = 2\mathsf{U}\left(\frac{k}{2}\right) + \mathsf{Perm}\left(\frac{k}{2}\right) + \mathcal{M}\left(\frac{k}{2}\right) = \sum_{i=0}^{\log k-1} 2^i \left(\mathsf{Perm}(\frac{k}{2^{i+1}})\mathcal{X} + \frac{k}{2^{i+1}}\mathcal{Y}\right)$$

$$\mathsf{U}(k) = \sum_{i=0}^{\log k-1} 2^i \left(\left(\frac{k}{2^{i+1}}\log\frac{k}{2^{i+1}} - \frac{k}{2^{i+1}} + 1\right)\mathcal{X} + \left(\frac{k}{2^{i+1}}\right)\mathcal{Y}\right)$$

$$\mathsf{U}(k) = \sum_{i=0}^{\log k-1} \left(\left(\frac{k}{2}\log k - \frac{k}{2}(i+1) - \frac{k}{2} + 2^i\right)\mathcal{X} + \left(\frac{k}{2}\right)\mathcal{Y}\right)$$

$$\mathsf{U}(k) = \left(\frac{k}{2}\log^2 k - \frac{k}{2}\left(\frac{\log k(\log k+1)}{2}\right) - \frac{k\log k}{2} + (k-1)\right)\mathcal{X} + \left(\frac{k\log k}{2}\right)\mathcal{Y}$$

$$\mathsf{U}(k) = \left(\frac{k}{4}\log^2 k - \frac{3k\log k}{4} + (k-1)\right)\mathcal{X} + \left(\frac{k\log k}{2}\right)\mathcal{Y} = \frac{k}{4}\log^2 k - \frac{k\log k}{4} + k - 1$$

$$\begin{aligned}\mathsf{CircuitU}(k) &= 4\left(\frac{k}{4}\log^2 k - \frac{3k\log k}{4} + (k-1)\right) + 3\left(\frac{k\log k}{2}\right)\\ &= k\log^2 k - \frac{3k\log k}{2} + 4k - 4\end{aligned}$$

Note that $\mathcal{M}$ block take the same role as our set of $\mathcal{Y}$ switches in Fig.12. We have:

$$\mathcal{SN}(k) = (2\mathsf{Perm}(k) + k)\mathcal{X} + \mathsf{U}(k)$$

By substituting this in $\mathcal{SN}(k)$ we get:

$$\mathcal{SN}(k) = (2(k\log k - k + 1) + k)\mathcal{X} + \mathsf{U}(k) = \frac{k}{4}\log^2 k + \frac{7}{4}k\log k + 1$$

$$\mathsf{Circuit}\mathcal{SN}(k) = 8k\log k - 4k + 8 + \mathsf{CircuitU}(k) = k\log^2 k + \frac{13}{2}k\log k + 4$$

Having the complexity of supernode for arbitrary k, we can compute the complexity of UC as follows:

$$\begin{aligned}\mathsf{EUG}_2(\mathfrak{s}) &= 2\mathcal{SN}(k)\frac{\mathfrak{s}\log\mathfrak{s}}{k\log k}\\ &= 2\left(\frac{k}{4}\log^2 k + \frac{7}{4}k\log k + 1\right)\frac{\mathfrak{s}\log\mathfrak{s}}{k\log k}\\ &= \left(\frac{\log k}{2} + 3.5 + \frac{2}{k\log k}\right)\mathfrak{s}\log\mathfrak{s}\end{aligned}$$

$$\begin{aligned}\mathsf{UC}(\mathfrak{s}) &= 2\mathsf{Circuit}\mathcal{SN}(k)\frac{\mathfrak{s}\log\mathfrak{s}}{k\log k}\\ &= 2\left(k\log^2 k+\frac{13}{2}k\log k+4\right)\frac{\mathfrak{s}\log\mathfrak{s}}{k\log k}\\ &= \left(2\log k+13+\frac{8}{k\log k}\right)\mathfrak{s}\log\mathfrak{s}\end{aligned}$$

Now we need to find the k value that minimizes the last two equations. Both $\mathsf{EUG}_2(\mathfrak{s})$ and $\mathsf{UC}(\mathfrak{s})$ are minimized when $k \approx 3.143$ with $\mathsf{EUG}_2(\mathfrak{s}) = 4.71\mathfrak{s}\log\mathfrak{s}$ and $\mathsf{UC}(\mathfrak{s}) = 17.84\mathfrak{s}\log\mathfrak{s}$.

5.3 Improved Supernode Constructions

Next, we present our improved supernodes with smaller gate count to replace the supernode in Fig. 2 in [Val76]. The main idea behind the improvement is taking advantage of the arbitrary fan-out for the actual UC circuit, and also observing that $\mathcal{Y}$ switches require one less gate compared to $\mathcal{X}$ switches. Fig. 14 demonstrates the construction. Notice the first distinguished node's extra fan-out with no additional cost compared to Fig. 7, which enables us to change v_2's implementation from $\mathcal{X}$ switch to $\mathcal{Y}$ switch. Our improved 2-way split supernode consists of 5 graph nodes, and requires a total of 19 gates for circuit implementation, leading to:

$$\mathsf{EUG}_2(\mathfrak{s}) = 2\mathsf{EUG}_1(\mathfrak{s}) = \frac{2\times 5\mathfrak{s}\log\mathfrak{s}}{2\times\log 2} = 5\mathfrak{s}\log\mathfrak{s}$$

$$\mathsf{UC}(\mathfrak{s}) = \frac{2\times 19\mathfrak{s}\log\mathfrak{s}}{2\times\log 2} = 19\mathfrak{s}\log\mathfrak{s}$$

Similarly the 4-way split supernode (Fig. 1) consists of 19 graph nodes, and requires the total of 72 gates for circuit implementation, leading to:

$$\mathsf{EUG}_2(\mathfrak{s}) = 2\mathsf{EUG}_1(\mathfrak{s}) = \frac{2\times 19\mathfrak{s}\log\mathfrak{s}}{4\times\log 4} = 4.75\mathfrak{s}\log\mathfrak{s}$$

$$\mathsf{UC}(\mathfrak{s}) = \frac{2\times 72\mathfrak{s}\log\mathfrak{s}}{4\times\log 4} = 18\mathfrak{s}\log\mathfrak{s}$$

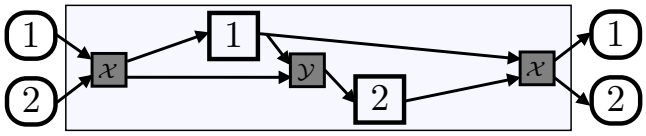

Fig. 14. 2-way split supernode construction ($k = 2$).

We observe that our supernode designs, are equivalent to what can be constructed using the modified [KS08a] given in previous section. In fact, for $k = 4$ we get $\mathsf{UC}(\mathfrak{s}) = 18\mathfrak{s}\log\mathfrak{s}$ and for $k = 2$ we get $\mathsf{UC}(\mathfrak{s}) = 19\mathfrak{s}\log\mathfrak{s}$ which exactly matches above supernodes.

6 Implementation

A main motivation for studying Valiant's UC was the lack of implementations for it in the literature. We provide the first implementation (of our improved variant) to our knowledge. Our implementation accepts any circuit in the format used in [TS15] for benchmark circuits, converts it to a circuit with reduced(=2) fan-out, and outputs the corresponding universal circuit as well as the set of control signal wires. We also provide the programming of UC, that is given a circuit $\mathcal{C}$ of certain size acceptable by a UC, we derive binary values that when assigned to control signal wires enables the universal circuit to emulate the functionality of $\mathcal{C}$. We also provide testing functions which evaluate the original circuit, the reduced-fanout circuit as well as the programmed universal circuit on a provided test input. Our implementation is done in C++ under Windows, but with no dependencies, which enables it to be compiled on other platforms. We intend to publish the source code.

Our implementation employs our improved 2-way[4] split supernode to construct the universal graph recursively, and adds the circuit implementation of each graph node to the UC as the wire index to all its inputs and outputs are determined. Programming the UC was the more challenging part due to some subtleties that we describe next. One of the main steps in programming Valiant UC is finding the matching which splits DAG_k to k instances of DAG_1 (Lemma 2). We need to find the matching and remove it from DAG_k to get DAG_{k-1} and DAG_1. We need to continue this to find all k instances of DAG_1. To this end, at each step we employed the Hopcroft-Karp algorithm [HK73] for the maximum matching in unweighted bipartite graphs with complexity $\mathcal{O}(E\sqrt{V})$, which aims for maximum number of edges in the matching. There are algorithms with better theoretical complexity which are reported to be slower in practice. We noticed that there is a possibility of multiple maximum matchings with some of them not satisfying our criteria of covering all vertices with fan-in/out k. Our solution to this was to do exactly as we have discussed in the proof of Lemma 2 and to add dummy edges to create a k-regular bipartite graph, while keeping track of these dummy edges. This guarantees that the Hopcroft-Karp algorithm will find the perfect matching, thus covering all vertices with fan-in/out k. We note that except for the proof of possibility by referring to Hall's theorem, these details are not discussed in [Val76].

After finding DAG_1s, before assigning it as a graph for the next recursion we need to perform some adjustments. This is done according to the proof of Lemma 4. We change the edge $u \to v$ to $u \to v - 1$ for the graph of next

[4] The implementation using our 4-way split supernode is in progress and is expected to lead to even better results.

recursion. Essentially the i-th distinguished node in the universal graph, accepts inputs from the i-th supernode, and provides outputs to the $(i+1)$-th supernode in the previous recursion. If $u = v$, then we do not need to consider the edge during the matching as it is a mapping inside the supernode and does not need to go through to next layer.

6.1 Comparison with UC Construction of [KS08a]

The efficiency analysis in [KS08a] counts the cost of $\mathcal{Y}$ switches as one, and the cost of $\mathcal{X}$ switches as two. In this work, we are interested in the total number of binary gates. The detailed complexity analysis of [KS08a] to find the exact gate complexity is given in appendix A. The total cost is:

$$\mathsf{CircuitU}(\mathsf{g}) = 3\mathsf{g}\log^2 \mathsf{g} + \mathsf{g}\log \mathsf{g} + 5\mathsf{g} - 5$$

$$\begin{aligned}\mathsf{UC}_{\mathsf{KS08}}(\mathsf{g}) = \mathsf{CircuitU}(\mathsf{g}) \\ + \left(\frac{\mathfrak{n}+2\mathsf{g}}{2}\log n + \frac{5\mathsf{g}+\mathfrak{m}}{2}\log \mathsf{g} + \mathfrak{m}\log \mathfrak{m} - \mathfrak{n} - \mathfrak{m} - \mathsf{g} + 4\right)\mathcal{X} \\ + (3\mathsf{g}-1)\,\mathcal{Y}\end{aligned}$$

Note that an additional $3\mathsf{g}\mathcal{Y}$ cost corresponding to g universal gates needs to be added to the above equation. [KS08a] claims a smaller UC compared to Valiant's UC for circuits of size ≈ 5000 or smaller when they set $\mathfrak{n} = 0.5\mathsf{g}$ and $\mathfrak{m} = 0.1\mathsf{g}$. We use the same parameters and set $\mathfrak{n} = 0.5\mathsf{g}$ and $\mathfrak{m} = 0.1\mathsf{g}$ as input and output sizes of Valiant's UC, $\mathsf{UC}_{\mathfrak{s}}^{\mathfrak{n},\mathfrak{m}}$ ($\mathfrak{s} = \mathfrak{n} + 2\mathsf{g} + \mathfrak{m} = 2.6\mathsf{g}$) for our comparisons with [KS08a]. Note that setting $\mathfrak{s}$ as such, is an overestimate in practice. For a given circuit, the $\mathsf{g} + \mathfrak{m}$ overhead of reducing the fan-out, can be much lower and in fact can be zero in some cases. Instead of leaking the size of original circuit with arbitrary fan-out, we can first reduce the fan-out, and release the size of the reduced fan-out circuit.

We identify $\mathsf{g} \approx 3700$ to be the new break-point if we consider the theoretical complexity $18\mathfrak{s}\log\mathfrak{s}$ ($\mathfrak{s} = 2.6\mathsf{g}$) based on our improved 4-way split supernode. But, for input circuits with fan-in/fan-out two ($\mathfrak{s} = \mathfrak{n} + \mathsf{g} = 1.5\mathsf{g}$), Valiant's UC is strictly smaller for circuits with more than 25 gates.

The theoretical complexity is an upper bound, and we expect better complexity in an actual implementation. One reason for better complexity in implementation, originates from the saving in the last supernode. Note that in the complexity analysis we consider a full cost for all supernodes except for the last supernode with no outputs that we subtract the cost of output distinguished nodes. This is lower in implementation as for a k-way split, we have $\mathfrak{s} = k\mathfrak{s}' + k'$, where $1 \leq k' \leq k$. Recall, in our worst case analysis we had $k' = 1$. In implementation we only need to route the k inputs to k' distinguished nodes which requires a smaller circuit compared to a complete supernode. One way of implementing such a circuit is to modify Figure 12 as follows. We use the first permutation network (or a more optimized Truncated permutation of [KS08a] from k to k'),

then instead of $\mathsf{EUG}_1(k)$, we only use $\mathsf{EUG}_1(k')$. There is no need for the last permutation network as there are no outputs from the last supernode.

Our implementation is based on our improved 2-way split, with the theoretical estimate of $19\mathfrak{s}\log\mathfrak{s}$. For our experiments, we generate circuits of different sizes randomly, and provide each as an input to our implementation. Given an input circuit, our implementation outputs the value of $\mathfrak{s}$ and the size of the final UC (beside the actual UC). Using the test parameters of [KS08a] for Valiant's UC, implies $\mathfrak{s} = 2.6\mathfrak{g}$. For comparison with [KS08a], we compute $\mathfrak{g} = \mathfrak{s}/2.6$ from the $\mathfrak{s}$ recovered from implementation and use it as input to the [KS08a] complexity from above to compute the size of their UC. Interestingly, we find that the actual break-point is much lower at $\mathfrak{g} \approx 400$. Also note that this is in fact still an upper bound for the break point due to two reasons. First, we used the upper bound for number of extra gates added during the fan-out reduction. Second, we have used the less optimized 2-way split supernode for this experiments. The size of 4-way split is expected to be better, due to a better constant factor. Figure 15 compares the complexity of [KS08a] versus Valiant's UC for smaller circuits for the purpose of demonstrating the break-points.

6.2 Experimental Results

Figure 16 show the Valiant's UC sizes for larger circuits. Sizes for [KS08a] are provided for comparison. Table 1 shows the size of the universal circuit for some of the benchmark circuits provided in [TS15].

7 Future Directions

As stated earlier, by providing a detailed description and implementation, we hope to motivate the use of this construction in the cryptographic protocols as well as research for possible efficiency improvements. One of the main drawbacks of Valiant's construction is its limitation to circuits with fan-in/out two. We want to point out that a solution to this issue has been proposed in Theorem 3.2 of [Val76]. Furthermore, Theorem 3.3 [Val76] offers an improvement, which reduces the constant factor behind the construction's complexity by a constant $\approx 5/6$ (by reducing a factor of 2 to 5/3). Although the possibility is shown in the proofs, detailed steps and implementation remains as an interesting future work.

References

AMPR14. Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In PhongQ. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 387–404. Springer Berlin Heidelberg, 2014.

Bon76. J. A. Bondy. *Graph theory with applications.* American Elsevier Pub. Co, New York, 1976.

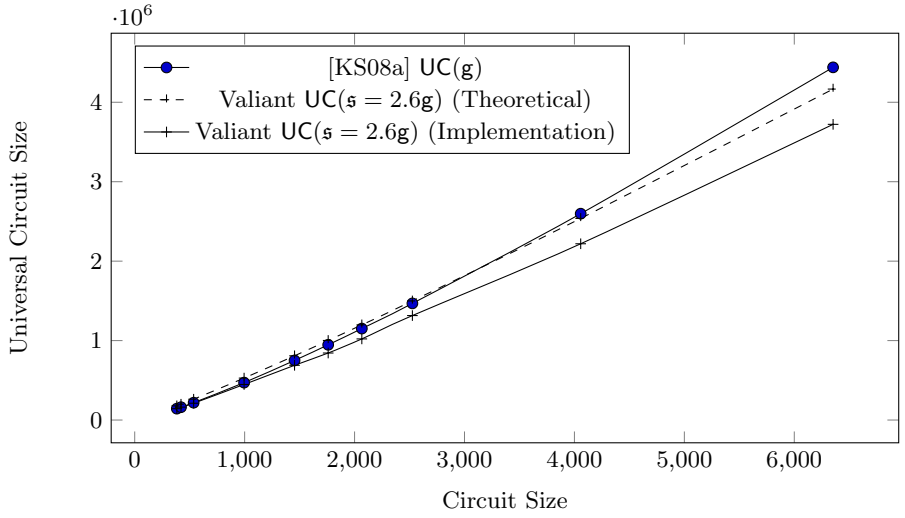

Fig. 15. Comparison of [KS08a] with Valiant's UC for circuits with arbitrary fan-out. Implementation: refers to sizes recovered from implementation (with our improved 2-way split supernode), Theoretical: refers to sizes computed from theoretical complexity analysis (with our improved 4-way split supernode).

GGH+13. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 40–49, Oct 2013.

HK73. John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

HKK+14. Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and AlexJ. Malozemoff. Amortizing garbled circuits. In JuanA. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014*, volume 8617 of *Lecture Notes in Computer Science*, pages 458–475. Springer Berlin Heidelberg, 2014.

KM11. Jonathan Katz and Lior Malka. Constant-round private function evaluation with linear complexity. In DongHoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 556–571. Springer Berlin Heidelberg, 2011.

KMR14. Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for xor gates that beats Free-XOR. In JuanA. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014*, volume 8617 of *Lecture Notes in Computer Science*, pages 440–457. Springer Berlin Heidelberg, 2014.

KS08a. Vladimir Kolesnikov and Thomas Schneider. Financial cryptography and data security. chapter A Practical Universal Circuit Construction and Secure Evaluation of Private Functions, pages 83–97. Springer-Verlag, Berlin, Heidelberg, 2008.

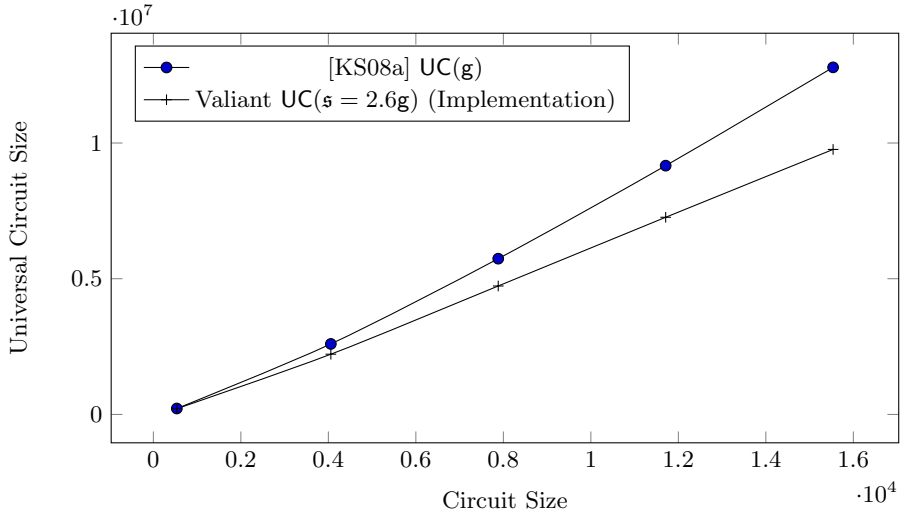

Fig. 16. Comparison of [KS08a] with Valiant's UC Implementation (with our improved 2-way split supernode) for larger circuits with arbitrary fan-out.

KS08b. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part II*, ICALP '08, pages 486–498. Springer-Verlag, 2008.

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, Denver, CO, USA, October 12-6, 2015*, pages 579–590, 2015.

MR13. Payman Mohassel and Ben Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In Ran Canetti and JuanA. Garay, editors, *Advances in Cryptology - CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 36–53. Springer Berlin Heidelberg, 2013.

MS13. Payman Mohassel and Saeed Sadeghian. How to hide circuits in MPC: an efficient framework for private function evaluation. In Thomas Johansson and PhongQ. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 557–574. Springer Berlin Heidelberg, 2013.

MSS14. Payman Mohassel, Saeed Sadeghian, and Nigel P. Smart. Actively secure private function evaluation. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014*, volume 8874 of *Lecture Notes in Computer Science*, pages 486–505. Springer Berlin Heidelberg, 2014.

Raz08. Ran Raz. Elusive functions and lower bounds for arithmetic circuits. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 711–720, New York, NY, USA, 2008. ACM.

TS15. Stefan Tillich and Nigel Smart. Circuits of basic functions suitable for MPC and FHE. `https://www.cs.bris.ac.uk/Research/`

`CryptographySecurity/MPC`, 2015. [Online; accessed 2-October-2015].

Val76. L.G. Valiant. Universal circuits (preliminary report). In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 196–203. ACM, 1976.

Wak68. Abraham Waksman. A permutation network. *J. ACM*, 15:159–163, January 1968.

Weg87. Ingo Wegener. *The Complexity of Boolean Functions*. B.G. Teubner J. Wiley, Stuttgart Chicester West Sussex New York, 1987.

Zim15. Joe Zimmerman. How to obfuscate programs directly. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9057 of *Lecture Notes in Computer Science*, pages 439–467. Springer Berlin Heidelberg, 2015.

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250. Springer Berlin Heidelberg, 2015.

A Circuit Complexity Analysis of [KS08a] UC Construction

The following is the circuit complexity analysis of [KS08a]. We first compute the cost of their building blocks, in terms of number of $\mathcal{X}$ and $\mathcal{Y}$ switches, and then use it to compute the circuit complexity. Computing the cost in terms of switches has two advantages for us. First, each switch can be seen as vertex in graph representation. Second, since each switch only requires one AND gate, this helps to learn the AND complexity, which directly translates to cryptographic cost as XOR gate can be evaluated for free in both the Yao's garbled circuit and GMW protocol. Next, we discuss the complexity analysis. We refer the reader to [KS08a] more details.

For a permutation network with u inputs and v outputs where $u \neq v$ (which they call expanded permutation if $u < v$ and truncated permutation if $u > v$), they propose a construction based on Waksman permutation network. We use the g subscript to refer to circuit complexity.

For $u < v$:

$$\mathsf{ExpandedPerm}(u, v) = \frac{u+v}{2} \log u - u + 1$$

$$\mathsf{ExpandedPerm}_{\mathsf{g}}(u, v) = 2(u+v) \log u - 4u + 4$$

For $u > v$, we need an extra $\mathcal{Y}$ switch to eliminate each extra input:

$$\mathsf{TruncPerm}(u, v) = \left(\frac{u+v}{2} \log v - v + 1\right) \mathcal{X} + (u-v)\mathcal{Y} = \frac{u+v}{2} \log v - 2v + u + 1$$

$$\mathsf{TruncPerm}_{\mathsf{g}}(u, v) = 2(u+v) \log v - 7v + 3u + 4$$

The cost of selection blocks is as follows:

$$u < v : \mathsf{Sel}(u, v) = \mathsf{ExpandedPerm}(u, v) + (v-1)\mathcal{Y} + \mathsf{Perm}(v)$$

$$\mathsf{Sel}(u,v) = \left(\frac{u+v}{2}\log u + v\log v - u - v + 2\right)\mathcal{X} + (v-1)\mathcal{Y}$$

$$u > v : \mathsf{Sel}(u,v) = \mathsf{TruncPerm}(u,v) + (v-1)\mathcal{Y} + \mathsf{Perm}(v)$$

$$\mathsf{Sel}(u,v) = \left(\frac{u+v}{2}\log u + v\log v - u - v + 2\right)\mathcal{X} + (u-1)\mathcal{Y}$$

They also give a custom construction for the $\mathsf{Sel}(v/2, v)$ with the following complexity:

$$\mathsf{Sel}(v/2,v) = \left(\frac{3v}{2}\log\frac{v}{2} - \frac{v}{2} + 2\right)\mathcal{X} + (v-1)\,\mathcal{Y}$$

using the above complexities, one can compute the cost of $\mathsf{U}(\mathtt{g})$ as follows:

$$\mathsf{U}(\mathtt{g}) = \left(\frac{3}{4}\mathtt{g}\log^2\mathtt{g} - \frac{5}{4}\mathtt{g}\log\mathtt{g} + 2\mathtt{g} - 2\right)\mathcal{X} + (2\mathtt{g}\log\mathtt{g} - \mathtt{g} + 1)\,\mathcal{Y}$$

$$\mathsf{U}(\mathtt{g}) = \frac{3}{4}\mathtt{g}\log^2\mathtt{g} + \frac{3}{4}\mathtt{g}\log\mathtt{g} + \mathtt{g} - 1$$

$$\mathsf{CircuitU}(\mathtt{g}) = 3\mathtt{g}\log^2\mathtt{g} + \mathtt{g}\log\mathtt{g} + 5\mathtt{g} - 5$$

The total cost is:

$$\mathsf{UC}_{\mathsf{KS08}}(\mathtt{g}) = \mathsf{Sel}(\mathtt{n}, 2\mathtt{g}) + \mathsf{U}(\mathtt{g}) + \mathsf{Sel}(\mathtt{g}, \mathtt{m})$$

$$\begin{aligned}\mathsf{UC}_{\mathsf{KS08}}(\mathtt{g}) = {} & \mathsf{CircuitU}(\mathtt{g}) \\ & + \left(\frac{\mathtt{n}+2\mathtt{g}}{2}\log n + \frac{5\mathtt{g}+\mathtt{m}}{2}\log\mathtt{g} + \mathtt{m}\log\mathtt{m} - \mathtt{n} - \mathtt{m} - \mathtt{g} + 4\right)\mathcal{X} \\ & + (3\mathtt{g}-1)\,\mathcal{Y}\end{aligned}$$

Note that there is an additional $3\mathtt{g}\mathcal{Y}$ cost corresponding to $\mathtt{g}$ universal gates not included in the following.