TECHNISCHE
UNIVERSITÄT
DARMSTADT

Master Thesis

Private Function Evaluation for Multi-Input Gates

Arthur Wigandt

October 15, 2021

ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING

Cryptography and Privacy Engineering Group
Department of Computer Science
Technische Universität Darmstadt

Supervisors: M.Sc. Daniel Günther
Prof. Dr. Yann Disser
Prof. Dr.-Ing. Thomas Schneider

Erklärung zur Abschlussarbeit gemäß §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Arthur Wigandt, die vorliegende Master Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Thesis Statement pursuant to §23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Arthur Wigandt, have written the submitted Master Thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are identical in content.

Darmstadt, October 15, 2021

Arthur Wigandt

Abstract

Private Function Evaluation (PFE) allows two parties to compute a Boolean function, where one party holds the function and one party holds the input, while keeping the input and function private. Only the output of the function shall be revealed. PFE can be reduced to Secure Two-Party Computation (STPC) by evaluating a Universal Circuit (UC), which is a Boolean circuit that can be programmed to compute any other circuit up to a given size as a public function with an STPC protocol like Yao's garbled circuits (FOCS'86) or the Goldreich-Micali-Wigderson protocol (STOC'87). Most UC implementations are restricted to binary gates and based on Edge-Universal Graphs (EUGs). In this thesis, we extend UCs to support gates with arbitrary many inputs via a black-box reduction from EUGs. Then, we implement the state-of-the-art UC of Liu et al. (CRYPTO'21) and the original UC of Valiant (STOC'76) and extend both to support multi-input gates. Furthermore, we compare our new construction with the original idea of Valiant to allow higher input gates by merging sufficiently many EUGs. However, this leads to a linear UC size increase in the maximum supported gate input size. Our multi-input gate construction circumvents this problem by specifying the maximum number of inputs per gate. This is a trade-off between efficiency and privacy since our approach leaks more information than standard PFE schemes. We show that both multi-input constructions always reduce the UC size compared to the standard binary construction.

Acknowledgments

Before we begin with the actual thesis, I would like to thank Prof. Dr. Thomas Schneider and Daniel Günther for providing this interesting topic, their excellent support, and their valuable feedback. My thanks goes to Prof. Dr. Yann Disser, who made this interdisciplinary thesis between computer science and mathematics possible. Furthermore, I am also very much obliged to Hossein Yalame, who supported me with synthesizing the Boolean circuits used for the benchmarks.

Contents

1 Introduction

Assume two parties A and B want to jointly evaluate a Boolean function f on input x, where party A possesses the function f and party B provides the input x. The goal is that both parties learn the output of the function $f(x)$ while keeping the input and function private. This task is called Private Function Evaluation (PFE) and can be achieved by constructing so-called Universal Circuits (UCs). On a high level, a UC is a Boolean circuit which allows to compute any Boolean circuit up to a given size by specifying so-called programming bits.

We can reduce PFE to Secure Two-Party Computation (STPC) [SYY99; Pin02; KS08a] by evaluating these UCs with an STPC protocol like Yao's garbled circuits [Yao86] or the Goldreich-Micali-Wigderson protocol [GMW87]. These protocols allow to evaluate a public Boolean function, represented as a Boolean circuit, privately, e.g., the inputs of each party remain secret, and only the output of the function is given. Using the programming bits of the UC as the private input of party A and the actual input of party B as his input, we can compute $f(x)$ while keeping the function f and input x secret. This approach directly leads to the challenge of constructing UCs of minimum size. The idea of UCs was originally introduced in [Val76] by Valiant, who also provided an asymptotically optimal construction with size $\Theta(n \log n)$, where n is the number of inputs, gates, and outputs. He used methods from graph theory and defined so-called Edge-Universal Graphs (EUGs), which build the core of the UC construction. Valiant provided two constructions that both depend on the same recursive structure with sizes $\sim 5n \log_2 n$ for the 2-Way construction and $\sim 4.75n \log_2 n$ for the 4-Way construction.

Using standard Universal Circuits for binary gates, a multi-input gate circuit must first be converted into a circuit with binary gates. In general, this transformation results in exponentially more gates [Weg87, Theorem 2.1].

The first UCs for multi-input gates were given in [SS08] (cf. *Related Work*). Also, Valiant proposed an idea for circuits with higher gate input sizes by merging sufficiently many EUGs. However, this approach allows all gates to have this many inputs. In particular, Valiant's construction grows linearly in the number of the maximum supported gate input size. This can result in a massive overhead as we will demonstrate. In this thesis, we present a construction that does not grow linearly with the maximum gate input size. Our construction is based on the construction algorithms for standard EUGs and uses them in a black-box manner. Therefore, each improvement in the size of EUGs also improves our construction. We pre- and post-process the circuits to be embedded and use standard EUG constructions as a black box. In our construction, we have to decide how many inputs each gate should support. This information, in addition to the number of inputs, gates, and outputs, is leaked. Therefore, we can only guarantee the anonymity of the circuit up to these restrictions. Thus,

our construction is a trade-off between efficiency and privacy, and it can also be seen as a Semi-Private Function Evaluation (SPFE) construction. SPFE, as proposed in [PSS09], is similar to PFE with the difference that the anonymity of the function f is only guaranteed within a class of functions $\mathcal{F}$, i.e., the information $f \in \mathcal{F}$ is leaked, but the concrete function stays private. The concrete anonymity set of our construction is defined in Section 4.2. A typical use case for SPFE are functions that can be split into a private and a public part. For example, Günther et al. showed in [GKSS19] how a car insurance tariff can be computed privately. A general framework for SPFE with concrete applications is given in [PSS09].

Applications of PFE

Private Function Evaluation can be used in situations, where the input contains sensible data and the function contains information like internal business data, knowledge, or research results. A direct example for this is privacy-preserving credit checking [FAZ05], where the concrete financial situation of the loanee can be kept private, while the loaner can make sure that their credit-granting policies are enforced and kept secret. In [OI07], it was shown how PFE can be used to privately search for information (e.g., keywords or their combinations) on streaming data without revealing the search criteria. PFE can also be used for privacy-preserving remote diagnostics as shown in [BPSW07; BFK+09], where the diagnostics routine, represented by a binary decision tree or a linear branching program, as well as the user's information is kept secret. In [SS08], Sadeghi and Schneider used their multi-input gate UC construction to securely evaluate Neural Networks. Further applications of PFE are private queries in database management systems [PKV+14; FVK+15] or privacy-preserving intrusion detection systems [MS13; NSMS14].

Related Work

The first UC constructions for multi-input gates were presented in [SS08] by Sadeghi and Schneider. They proposed three different constructions. The first one is an iterative construction that uses Choice Blocks which select the desired input wires to the multi-input gates out of the circuit inputs and previous gate outputs. Sadeghi and Schneider also proposed three possible constructions for instantiating a Choice Block. Using their proposed sub-linear Choice Block, their construction has size $\mathcal{O}(\log d\ k^2)$, where k is the number of gates and d the maximum gate input size. To keep the notation compact, we refrain from giving the complexity also for the number of inputs and outputs since they are, in general, much smaller than k. The second UC construction with size $\mathcal{O}(dk \log^2 k)$ is a generalization of the modular UC of [KS08a]. The third proposed UC construction in [SS08, §4.3] is a generalization of Valiant's original construction [Val76]. It has the asymptotic optimal size of $\mathcal{O}(dn \log n)$. All gates in above constructions support up to d inputs. Therefore, they do not leak the concrete gate sizes, in contrast to our construction.

There were several improvements to the original UC constructions of Valiant [KS16; LMS16; GKS17; ZYZL19; AGKS20]. The best known construction using Valiant's framework was given in [ZYZL19] with size $\sim 4.5n \log_2 n$. This construction was further improved in [AGKS20] using only $\mathcal{O}(n)$ memory during the UC compilation. Recently, a new even more efficient

construction, deviating more from the original idea of [Val76], was proposed in [LYZ+21] and reduced the size to $\sim 3n \log_2 n$. Most notably, this goes beyond the lower bound of $\sim 3.64n \log_2 n$ [ZYZL19] that holds for constructions using Valiant's framework. In [KS08a], Kolesnikov and Schneider proposed a modular UC construction of size $\sim 1.5k \log_2^2 k$ which is more efficient than Valiant's construction for small circuit sizes.

Mohassel and Sadeghian proposed two PFE schemes in [MS13]. The first scheme uses homomorphic encryption and has linear complexity, while the second scheme uses Oblivious Switching Networks and can be realized by mainly symmetric cryptography. This construction has a complexity of $\mathcal{O}(n \log n)$. In [BBKL19], Bingöl et al. further improved the communication effort of the latter construction by 40% using the half gates garbling technique of [ZRE15].

Katz and Malka showed in [KM11] that PFE can also be realized in linear complexity at the cost of using homomorphic encryption. In [MSS14], Mohassel and Sadeghian extended their protocol of [MS13] such that it is secure against active adversaries while retaining the linear complexity. In [BBK18], Biçer et al. proposed a PFE scheme based on homomorphic encryption with a reusability feature. The scheme is split into an initial execution and a resumption protocol for subsequent evaluations of the function, where the latter can reuse tokens of the initial execution to be more efficient. Recently, in [HKRS20], Holz et al. showed the practicality of homomorphic encryption based PFE schemes by improving the protocol of Katz and Malka [KM11] and using state-of-the-art homomorphic encryption schemes.

Felsen et al. showed in [FKSW19] that PFE can be realized very efficiently by using a Trusted Execution Environment (TEE).

Our Contributions

We provide a rigorous mathematical framework to analyze and prove the correctness of Valiant's EUG construction [Val76] and the-state-of-the-art EUG construction of [LYZ+21] (Chapter 3).

Then, we describe our multi-input gate construction, which is not growing linearly with the maximum gate input size, and prove its correctness (Chapter 4). To the best of our knowledge, there is no practical implementation of the state-of-the-art scheme of [LYZ+21] as their implementation only verifies correctness and size of the construction (cf. [LYZ+21, §4.2]). We provide the first practical implementation of Universal Circuits using the construction of Liu et al. [LYZ+21] and extend it with our multi-input gate construction and the (generalized) Valiant's multi-input gate construction of [Val76][SS08, §4.3]. We also implement a hybrid construction, which is a mix of our construction and Valiant's generalized multi-input gate construction (Section 5.4).

Last but not least, we compare the different multi-input gate constructions with the standard binary construction (Chapter 5). We show that all three multi-input constructions yield smaller UCs than the standard binary construction.

2 Preliminaries

In this chapter, we cover the needed graph theoretic results (Section 2.1), introduce the concept of an Edge-Universal Graph (Section 2.2), and show how we can use EUGs to enable Private Function Evaluation (Section 2.3).

We consider directed graphs $G = (V, E)$ throughout the rest of this work, if not explicitly stated otherwise, with V being the set of vertices and E being the set (or possibly multi set) of edges. $\mathbb{N}$ denotes the set $\{1, 2, ...\}$. $[k]$ denotes the set $\{1, 2, ..., k\}$ for $k \in \mathbb{N}$.

2.1 Graph Theory

Definition 1 (Basic Definitions)**.** *Let $G = (V, E)$ be a directed graph and $v \in V$.*

- *$\delta_G^+(v)$ denotes the set of all incoming edges to v, i.e., $\delta_G^+(v) := \{(u, v) \in E : u \in V\}$. For $U \subset V$, we define $\delta_G^+(U) := \{(u, v) \in E : u \in V \setminus U, v \in U\}$. $\delta_G^-(v)$ and $\delta_G^-(U)$ are defined analogously for outgoing edges.*
- *$\Gamma_G^+(v)$ denotes the set of incoming neighbors of v and is defined by $\Gamma_G^+(v) := \{u \in V : (u, v) \in E\}$. For $U \subset V$, we define $\Gamma_G^+(U) := \bigcup_{u \in U} \Gamma_G^+(u) \setminus U$. The sets $\Gamma_G^-(v)$ and $\Gamma_G^-(U)$ are defined analogously for neighbors connected by an outgoing edge.*
- *The indegree (resp. outdegree) of v is defined by $deg_G^+(v) := |\delta_G^+(v)|$ (resp. $deg_G^-(v) := |\delta_G^-(v)|$). G has fanin ρ if $deg_G^+(v) \leq \rho \ \forall \ v \in V$. G has fanout ρ if $\delta_G^-(v) \leq \rho \ \forall \ v \in V$.*
- *For $U \subset V$, $G[U] := \{U, \{e = (u, v) \in E : u, v \in U\}\}$ denotes the subgraph induced by U.*

For undirected graphs $G = (V, E)$, we define $\delta_G(v) := \{e = \{u, v\} \in E : u \in V\}$ for $v \in V$ and $G[U] := \{U, \{e = \{u, v\} \in E : u, v \in U\}\}$ for $U \subset V$. Because we often have to deal with multiple graphs in the remaining work, we will also say $v \in G$ (resp. $e \in G$) instead of $v \in V$ (resp. $e \in E$) for $G = (V, E)$. If the graph G is clear from the context, we use above notations without the index G, e.g., we write δ instead of δ_G.

Definition 2 (Topological order)**.** *Let $G = (V, E)$ be a directed acyclic graph. A topological order for G is a map $\eta_G : V \to \{1, ..., |V|\}$ such that $\forall \ (u, v) \in E : \eta_G(u) < \eta_G(v)$.*

Note that the topological order of a graph is, in general, not unique. However, we will waive the specification of the concrete topological order to keep the notation simple and always assume a fixed η_G for each graph G.

Definition 3 ($\Gamma_\rho(n)$). *The set $\Gamma_\rho(n)$ denotes all directed acyclic graphs with at most n vertices and fanin/fanout ρ for $\rho, n \in \mathbb{N}$.*

Definition 4 (Edge coloring). *An edge coloring of an undirected graph $G = (V, E)$ is a map $c : E \to [k]$ for $k \in \mathbb{N}$ with the property $c(e) \neq c(e') \; \forall \; e, e' \in \delta(v)$ with $e \neq e' \; \forall \; v \in V$. A minimum edge coloring is an edge coloring with the smallest k. This k is called the chromatic number of G and denoted by $\chi(G)$.*

On a high level, an edge coloring colors the edges of a graph such that all "neighboring" edges (i.e., edges that are connected by a common vertex) have different colors.

Definition 5 (Bipartite graphs). *An undirected graph $G = (V, E)$ is called bipartite if there are disjoint $V', V'' \subset V$ such that $V = V' \cup V''$, and there are no edges within V' or V'', i.e., $G[V'] = (V', \emptyset)$ and $G[V''] = (V'', \emptyset)$.*

One important characterization of bipartite graphs is that they do not contain cycles of odd length. A proof for this can be found in [Die10, Prop. 1.6.1].

Theorem 1 (Kőnig's theorem ([Die10, Prop. 5.3.1])). *The chromatic number of an undirected bipartite graph equals the maximum degree of a vertex in this graph.*

Proof (by [Die10, Prop. 5.3.1]). Let $G = (V, E)$ be an undirected bipartite graph. Let Δ be the maximum degree of a vertex in G. Observe that $\chi(G) \geq \Delta$ since there is a vertex with Δ adjacent edges that all need a different color. For the direction $\chi(G) \leq \Delta$, we use induction on $|E|$.

Base case: $|E| = 0$
Then $\Delta = \chi(G) = 0$ because there is no edge to color.

Induction step: $|E| = m + 1$
Choose an arbitrary edge $e = (u, v) \in E$ and remove it. Let $\tilde{G} = (V, \tilde{E})$ denote the resulting graph. By induction hypothesis, we can find a Δ-coloring of $\tilde{G}$. Now consider the vertices u and v of the removed edge. They have at most $\Delta - 1$ neighbors in $\tilde{G}$, and therefore, u and v have at least one color that is not used by an adjacent edge.

Case 1: There is a color that is not used by adjacent edges to u and v.
Then, we can color e with this color and obtain a Δ-coloring of G.

Case 2: There is no color that is not used by adjacent edges to u and v.
Then, we can choose a color β that is used by an adjacent edge $\tilde{e}$ to u but not by an adjacent

edge to v and a color α that is not used by an adjacent edge to u. Now consider the longest walk that starts with $\tilde{e}$ and continues with edges alternatingly colored to α and β. Such a walk must be a path. Therefore, a longest walk must exist:

Note that every vertex on this path, except for the endpoints, must have adjacent edges with colors α and β. If there was a cycle starting at one of those vertices on the path, either α or β would be used twice for adjacent edges to this vertex. Also there is no cycle possible that uses the starting point since by assumption the start vertex u has no α-colored edge.

We also know that this walk can not contain v because v has no adjacent β-colored edge and would have to be on this walk by an (incoming) α-colored edge. Thus, the length of the path until v would be even, and adding e to this path would introduce a cycle of odd length in a bipartite graph, which is impossible.

Now we can swap the colors of each edge on this path. Because we chose the longest such walk and u has no adjacent α-colored edge, this results in a valid coloring where u has no β-colored adjacent edge. Now we can color e with β to get a Δ-coloring for G. □

The constructive proof directly yields the following algorithm to find a minimum coloring:

Algorithm 1: EDGECOLORING(G)

Input : bipartite undirected graph $G = (V, E)$
Output : minimum edge coloring c

$\Delta \leftarrow \max_{v \in V}\{deg_G(v)\}$
foreach *uncolored* $e = (u, v) \in E$ **:**
  if $\exists\ i \in [\Delta]$ *with* $c(\bar{e}) \neq i\ \forall\ \bar{e} \in \delta_G(u) \cup \delta_G(v)$ **:**
    $c(e) \leftarrow i$
  else
    choose α, β *and* $\bar{e}$ *like in the proof*
    swap the colors of the longest β*-*α*-alternating path starting with* $\bar{e}$
    $c(e) \leftarrow \beta$
return c

Corollary 1. *Algorithm 1 returns a minimum edge coloring for an undirected bipartite graph* $G = (V, E)$ *in time* $\mathcal{O}(|V||E|)$.

Proof. Correctness directly follows from Theorem 1. The maximum degree Δ of a vertex in G can be found in time $\mathcal{O}(|V| + |E|)$ by Depth-First-Search. We can determine α, β, and $\bar{e}$, or find a free color for each $e = (u, v)$ in time $\mathcal{O}(|V|)$ by iterating through the adjacent edges of u and v (each at most $|V|$), and then remove colors from the candidate set that are used by any of these edges. If there is no color left, there is no free color, and we do the same procedure with two candidate sets to find α and β. Each path in which the colors are possibly

swapped has at most length $|V|$. Therefore, each iteration of the algorithm takes time $\mathcal{O}(|V|)$, resulting in a total of $\mathcal{O}(|V||E|)$ time for the whole algorithm. □

At this point, we want to say that there are faster algorithms like [Alo03; COS01] than the one derived from Kőnigs theorem. The fastest algorithm so far running in time $\mathcal{O}(|E|\log\Delta)$ was proposed in [COS01]. Next, we show how we can use an edge coloring to partition the edge set of a graph with fanin/fanout ρ into ρ edge sets that can be embedded more easily as we will see in Proposition 1.

Corollary 2 ([AGKS20, Theorem 1 & 2]). *Let $G = (V, E) \in \Gamma_\rho(n)$. Then, there exists a (disjoint) partition $E_1, E_2, ..., E_\rho$ of E such that $G_i = (V, E_i) \in \Gamma_1(n)\ \forall\ i \in [k]$.*

Proof (cf. [AGKS20, Theorem 1 & 2]). Construct a bipartite graph $\bar{G} = (\bar{V}, \bar{E})$ with

$$\bar{V} \coloneqq V \cup \tilde{V} \coloneqq V \cup \{\tilde{v} : v \in V\},$$
$$\bar{E} \coloneqq \{\{u, \tilde{v}\} : (u, v) \in E\}.$$

Since each $\bar{v} \in \bar{V}$ has at most ρ neighbors, there exists a ρ-coloring of $\bar{G}$ by Kőnig's theorem (Theorem 1). Now construct an edge set

$$E_i \coloneqq \{(u, v) \in E : \{u, \tilde{v}\} \in \bar{E} \text{ with } c(\{u, \tilde{v}\}) = i\} \text{ for each color } i \in [\rho].$$

Note that an undirected edge $\{u, \tilde{v}\} \in \bar{E}$ corresponds to exactly one directed edge $(u, v) \in E$ since G is acyclic. Consider E_i for $i \in [\rho]$ and let $v \in V$. By the coloring property, there is at most one edge containing v and one edge containing $\tilde{v}$ colored with color i. Thus, there is at most one incoming and one outgoing edge in E_i for each vertex v. Therefore, $G_i \in \Gamma_1(n)$. □

2.2 Edge-Universal Graphs

Definition 6 (Edge-Embedding). *Let $G = (V, E, P)$ and $G' = (P, E')$ be directed graphs with $P \subset V$ and G' acyclic. An edge-embedding from G' into G is a map $\psi : E' \to \mathcal{P}_G$, where $\mathcal{P}_G$ denotes the set of all paths in G with the following properties:*

- *$\psi(e')$ is a u-v-path (in G) for $e' = (u, v) \in E'$,*
- *$\psi(e')$ and $\psi(\tilde{e}')$ are edge-disjoint paths for all $e', \tilde{e}' \in E'$ with $e' \neq \tilde{e}'$.*

Definition 7 (Edge-Universal Graph). *A directed graph $G = (V, E, P)$ with ordered pole set $P \coloneqq \{p_1, ..., p_n\} \subset V$ is called an Edge-Universal Graph for $\Gamma_\rho(n)$ if:*

- *Every acyclic $G' = (P, E') \in \Gamma_\rho(n)$ that is order preserving, i.e., $\forall\ e = (p_i, p_j) \in E' \Rightarrow i < j$, can be edge-embedded into G.*

We also write $\mathcal{U}_\rho(n)$ for an EUG for $\Gamma_\rho(n)$.

Note that there are some technical differences between our definition of an EUG and the original definition of an EUG in [Val76, §2]. In our definition, only graphs that consist of the pole set P of the EUG can be embedded. This together with the restriction to order preserving graphs simplifies notation and improves the readability of the proofs. In the original definition of an edge-embedding, there is an additional map $\varphi : V' \to P$ that maps the vertices of the graph to be embedded to the poles of the EUG. The advantage of Valiant's definition is that the vertex set of the graph to be embedded must not be the pole set. However, since we are using nested EUGs, we have to treat some of the nodes of the EUG as poles of its nested EUG and solve the task of edge-embedding in this nested EUG. In particular, we will edge-embed edges between those vertices in the nested EUG, and this requires us that this vertex is a pole of the nested EUG. If we used a map φ like in the original definition, this would not be guaranteed.

This is solely a technical requirement, and our modified definition has no practical restrictions: Assume we want to edge-embed $G' = (V', E') \in \Gamma_\rho(n)$ with $V' \neq P$. Then, we can easily build a map $\varphi : V' \to P$ that maps the vertices of G' to the poles of G such that $\tilde{G}' = (P, \tilde{E}')$ (with $\tilde{E}' := \{(\varphi(u), \varphi(v)) : e = (u, v) \in E'\}$) is order preserving. To do that, sort the vertices of G' topologically (such that $V' = \{v'_1, ..., v'_m\}$ for $m \leq n$). Then, set $\varphi(v'_i) = p_i \ \forall \ v_i \in V$. If $|V'| < n$, we can add dummy vertices to V' with no incoming or outgoing edges. Now, $\tilde{G}'$ can be edge-embedded into G.

The other difference is that we do not require the EUG to be acyclic, although an EUG must be acyclic to be transformed into a Universal Circuit. This relaxation of the definition allows us to elegantly prove the correctness of the EUG construction by Liu et al. [LYZ+21], which deviates from the original construction of Valiant [Val76].

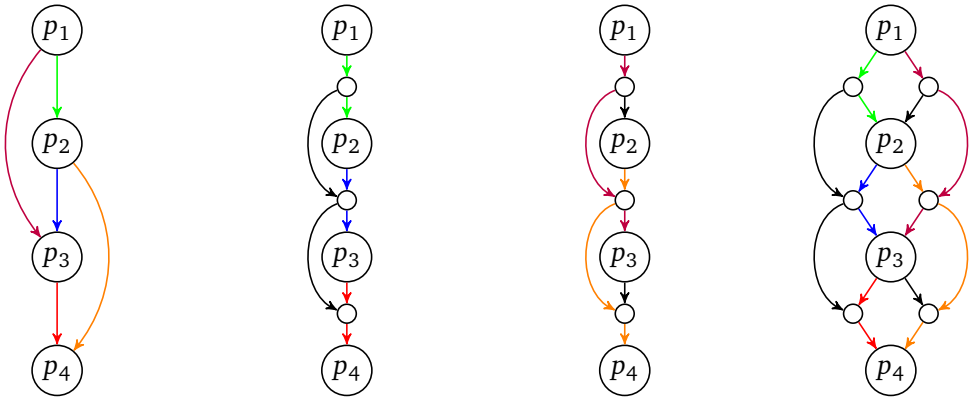
We summarize the concrete EUG constructions by Valiant and Liu et al. in Section 3.1 and Section 3.2. For now, assume that we have an EUG for $\Gamma_1(n)$. Then, we can easily construct an EUG for $\Gamma_\rho(n)$ by merging ρ instances of our $\Gamma_1(n)$ EUG. This idea was originally given in [Val76, Corollary 2.2].

Definition 8 (Merging of EUGs)**.** *Let $G = (V, E, P)$ and $\bar{G} = (\bar{V}, \bar{E}, P)$ be two EUGs for $\Gamma_\rho(n)$ and $\Gamma_{\bar{\rho}}(n)$ with the same same pole order and $V \cap \bar{V} = P$. Then $\hat{G} = (V \cup \bar{V}, E \cup \bar{E}, P)$ is called the merging of G and $\bar{G}$.*

An illustration of the coloring, edge-embedding, and merging process is given in Figure 2.1. Note that if $E \cap \bar{E} \neq \emptyset$, the edge set of the merged EUG will be a multi set. For example, this can happen if there is an edge from a pole to a pole.

Proposition 1. *The merging of a $\Gamma_\rho(n)$ EUG and a $\Gamma_{\bar{\rho}}(n)$ EUG is a $\Gamma_{\rho+\bar{\rho}}(n)$ EUG.*

Proof. Let $\hat{G} = (\hat{V}, \hat{E}, P)$ be the merging of the $\Gamma_\rho(n)$ EUG $G = (V, E, P)$ and the $\Gamma_{\bar{\rho}}(n)$ EUG $\bar{G} = (\bar{V}, \bar{E}, P)$. Let $G' = (P, E') \in \Gamma_{\rho+\bar{\rho}}(n)$ be the order preserving graph to be edge-embedded into $\hat{G}$. By Corollary 2, we can partition E' into disjoint sets $E^*_1, E^*_2, ..., E^*_{\rho+\bar{\rho}}$ such that $G^*_i =$



(a) $\Gamma_2(4)$ graph (b) $\mathcal{U}_1(4)$ (c) $\mathcal{U}_1(4)$ (d) merged $\mathcal{U}_2(4)$

Figure 2.1: (a) shows the $\Gamma_2(4)$ graph with already partitioned edge sets E_1 and E_2, (b) shows the EUG in which the edge set E_1 is embedded, (c) shows the EUG in which the edge set E_2 is embedded, (d) shows the merged EUG with all edges embedded.

$(P, E_i^*) \in \Gamma_1(n)\ \forall\ i \in [\rho + \bar{\rho}]$. Set $E_1' = \bigcup_{i=1}^{\rho} E_i^*$ and $E_2' = \bigcup_{i=\rho+1}^{\rho+\bar{\rho}} E_i^*$. Then, $G_1' := (P, E_1') \in \Gamma_\rho(n)$ and $G_2' := (P, E_2') \in \Gamma_{\bar{\rho}}(n)$. Now, G_1' and G_2' can be edge-embedded into G and $\bar{G}$ respectively. Define $\psi_{\hat{G}} : E' \to \mathcal{P}_{\hat{G}}$ as follows:

$$\psi_{\hat{G}}(e') \mapsto \begin{cases} \psi_G(e'), & \text{if } e' \in E_1', \\ \psi_{\bar{G}}(e'), & \text{if } e' \in E_2'. \end{cases}$$

Since $E' = E_1' \cup E_2'$ and all edges in E_1' and E_2' were edge-embedded into G and $\bar{G}$, each edge $(u, v) \in E'$ is mapped to a u-v-path. Furthermore, these paths are edge disjoint because E and $\bar{E}$, in which E_1' and E_2' were edge-embedded, are disjoint. □

Corollary 3 ([Val76, Corollary 2.2]). *An EUG for $\Gamma_\rho(n)$ can be constructed by merging ρ EUGs for $\Gamma_1(n)$.*

Proof. Let $G = (V, E, P)$ be a $\Gamma_1(n)$ EUG. Create $\rho - 1$ copies of G with the same pole set and merge these graphs successively. Correctness follows directly by applying Proposition 1 ρ times. □

With this observation in mind, we can restrict ourselves on constructing $\Gamma_1(n)$ EUGs.

2.3 Using Edge-Universal Graphs for Private Function Evaluation

Now that we understand EUGs, we can come back to our main goal which was to enable PFE. Recall that the setting was the following:

Let Alice and Bob be two parties, where Alice holds a Boolean function f, Bob holds some private information x, and Alice, Bob, or both, want to learn $f(x)$ without revealing f or x to the other party.
Note that we do not discuss the technical details about the underlying protocols or the concrete security definitions as PFE via Universal Circuits is a standalone application for STPC. We present the general and most common approach of reducing PFE to STPC [SYY99; Pin02; KS08a; KS16]. The high level idea of the reduction is:

1. Alice creates a Boolean circuit representing the Boolean function f of size n with n_i inputs, n_g gates, and n_o outputs.
2. Alice and/or Bob create a Universal Circuit of size $\geq n$ with at least n_i inputs, n_g gates, and n_o outputs.
3. Alice creates a programming p^f such that the UC computes f.
4. Bob and Alice evaluate the (public) Universal Circuit with private inputs p^f and x using an STPC protocol.

Universal Circuits

A Boolean Circuit can be seen as a directed acyclic graph whose vertices consist of Boolean inputs, gates, or outputs. The number of inputs, gates, and outputs is denoted by n_i,n_g and n_o. A directed edge is also called a wire. A Boolean gate is a function $z\colon \{0,1\}^k \to \{0,1\}$ for $k \in \mathbb{N}$. The inputs to this gate are the values of the incoming wires to the corresponding node. The outgoing wires of a node have the value of the evaluated gate. Note that every Boolean function with k inputs can be represented by a lookup table with 2^k entries. We further have distinguished nodes for inputs and outputs. However, we can always divide a k-input gate into $\mathcal{O}(2^k)$ binary gates. But in general, we can not avoid an exponential increase of the number of gates by this transformation [Weg87, Theorem 2.1]. The two most prominent minimization methods for Boolean formulas (resp. circuits) are given in [Qui52; Kar53]. Since the EUG constructions that we use (cf. Sections 3.1 and 3.2) were made for $\Gamma_\rho(n)$ graphs, we possibly need to reduce the outdegree of the gates. This can be done by using so-called copy gates which just copy their inputs [Val76, Corollary 3.1].

Definition 9 (Universal Circuit ([Val76; AGKS20])). *A Universal Circuit $\mathcal{U}$ for n_i inputs, n_g gates, and n_o outputs is a Boolean circuit that can be programmed to compute any Boolean circuit C with n_i inputs, n_g gates and n_o outputs by defining a set of programming bits p^f such that $\mathcal{U}(x, p^f) = C(x)$ for all possible input values $x \in \{0,1\}^{n_i}$.*

Note that a Universal Circuit can also compute circuits with less than the specified number of inputs, gates, and outputs by using dummy inputs, gates, and outputs with no functionality.

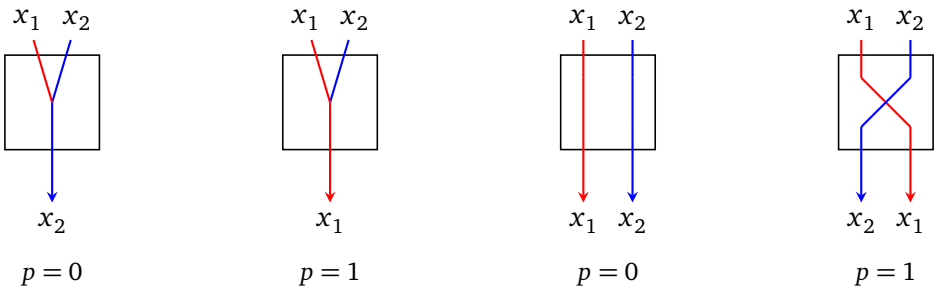
From Edge-Universal Graphs to Universal Circuits

Recall that the goal of the EUG constructions was to construct a Universal Circuit. Assume that we have a merged EUG with the construction of Valiant [Val76] or Liu et al. [LYZ+21] and a corresponding edge-embedding (cf. Definition 7). We are looking for a way to route the output of one vertex to another vertex. Since we already have an edge-embedding that yields a path, we can encode this path information in the programming bits for the Universal Circuit. We will define three additional types of nodes:

- Y-Switch (cf. Figure 2.2a): A Y-Switch has two incoming wires and only outputs one wire according to the programming bit. If the programming bit for this node is 0, the output will be the right wire. If the programming bit is 1, the output will be the left wire.
- X-Switch (cf. Figure 2.2b): An X-Switch has two incoming wires and two outgoing wires. If the programming bit of this node is 0, the wires will be output in the same order. If the programming bit is 1, the left input wire will be the right output wire and vice-versa.
- Universal Gate (for 2 inputs): A Universal Gates for a (binary) pole has two input wires and two output wires. For each Boolean gate, there are programming bits such that this node computes the desired Boolean gate.

The concrete instantiation of the X- and Y-Switches as well as the Universal Gates for two inputs with Boolean gates can be found in [Val76; KS08b]. The Universal Gate construction for more than two inputs is described in Section 4.3. Using these node types (in addition to input/output nodes), we can transform an EUG $G = (V, E, P)$ into a UC as follows. For each node $u \in V$:

- u is an input node, i.e., $\deg^+(u) = 0$:
  - Then u will also be an input node in the UC.
- u is an output node, i.e., $\deg^-(u) = 0$:
  - Then u will also be an output node in the UC.
- u just forwards one wire and is no pole, i.e., $\deg^+(u) = 1 \wedge \deg^-(u) = 2 \wedge u \notin P$:
  - Then we replace u by two wires since u does not yield another path option.
- u has two inputs, one output, and is no pole, i.e., $\deg^+(u) = 2 \wedge \deg^-(u) = 1 \wedge u \notin P$:
  - Then u becomes a Y-Switch.
- u has two inputs, two outputs, and is no pole, i.e., $\deg^+(u) = 2 \wedge \deg^-(u) = 2 \wedge u \notin P$:
  - Then u becomes an X-Switch.
- u is a pole, i.e., $u \in P$:
  - Then u becomes a Universal Gate.



(a) Y-Switch (b) X-Switch

Figure 2.2: The used switching nodes depending on the programming bit p.

This will cover all possible nodes in the constructions of Valiant and Liu et al. [Val76; LYZ+21] as we will see in Sections 3.1 and 3.2.

Translating The Edge-Embedding to Programming Bits

Assume we have an edge-embedding $\psi : P \to V$ and want the corresponding programming bits. We define for each switching node a left and right incoming (resp. outgoing) wire. For each u-v-path $\psi((u,v))$, we set the programming bits of the switching nodes along the path accordingly. For example, assume there is an X-Switch along the path, and the path enters this switch by the left wire and needs to leave it by the right wire. Then, we set the programming bit of this X-Switch to 1. Since we have edge-disjoint paths and the programming bits can simulate every possible choice for each switch, this is always possible.

In practice, we directly set the control bits of the switches and do not build an explicit edge-embedding map ψ. A modular algorithm to acquire the programming bits supporting Valiant's original construction (amongst others) is given in [GKS17]. With Algorithm 4 in Section 3.1, we provide a variant of this algorithm for the constructions of Valiant [Val76] and Liu et. al. [LYZ+21] that returns an explicit edge-embedding ψ to prove the correctness of the EUG constructions.

3 Edge-Universal Graph Constructions

In this chapter, we present the original idea of Valiant [Val76] (Section 3.1), describe the state-of-the-art construction of Liu et al. [LYZ+21] (Section 3.2), and prove the correctness of both constructions.

3.1 Valiant's Edge-Universal Graph Construction

The trick of Valiant's EUG construction is to divide the problem of finding a $\Gamma_1(n)$ EUG into the problem of finding a $\Gamma_1(\lceil \frac{n}{k} \rceil - 1)$ EUG via a divide and conquer approach that divides the EUG in k sub EUGs. The two main ideas of Valiant are the recursive structure of the construction and the use of so-called Superpoles. A Superpole can be thought of a node that has inputs, poles, and outputs, and guarantees an edge-disjoint routing between the inputs and poles of the Superpole and between the poles and outputs. Imagine having the poles in topological order in one line, and then grouping each k successive poles into a Superpole. Now we can use the inputs and outputs of each Superpole as the poles for another EUG. If we want to edge-embed an edge $e = (u, v)$, we first do proper routing inside the Superpoles that contain u and v. This means that we set the concrete output node at which u leaves his Superpole and the input node at which u enters the Superpole of v and route accordingly inside the Superpoles. Then, we can edge-embed the edge between the previously declared output node and input node in our nested sub EUG since these nodes are just the poles of this sub EUG.

We begin by defining the properties needed for our Superpoles. To capture this, we define so-called Augmented k-Way Valiant Blocks (cf. Augmented DAG in [LYZ+21]). An Augmented k-Way Valiant Block can be seen as a map that dictates which inputs will be directed to which poles and which poles will be directed to which other poles or outputs. However, any description of this mapping would suffice to capture the idea of a Superpole. For example, [AGKS20] used an input and output vector for each Superpole to describe the needed routing inside the Superpole.

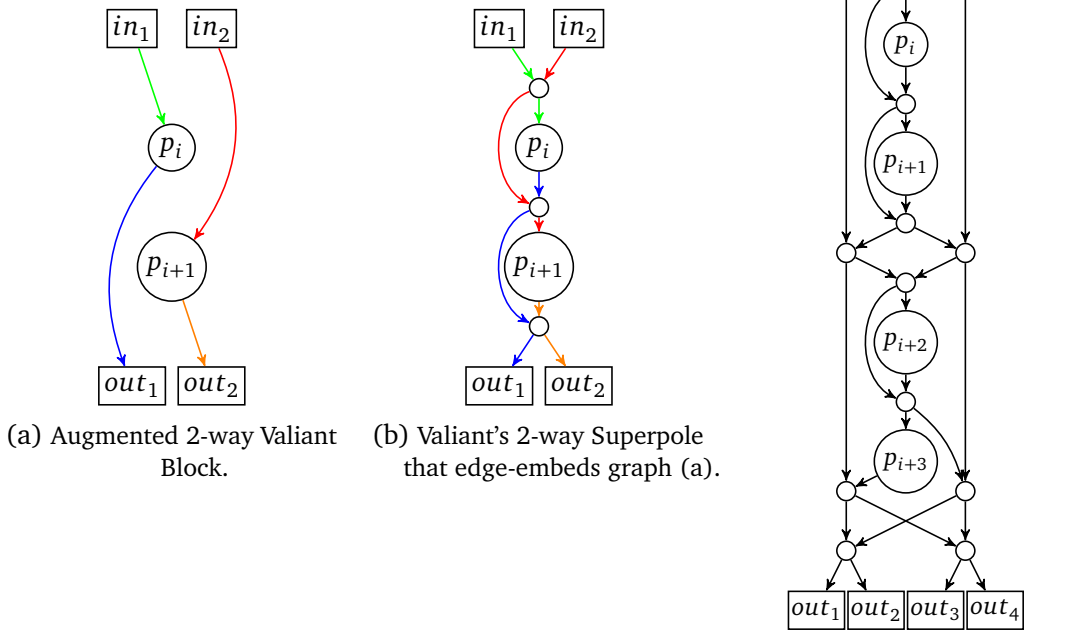
Definition 3.1.1 (Augmented k-Way Valiant Block)**.** *An Augmented k-Way Valiant Block $G = (V, E)$ for P, I, O is a directed graph such that*

- *$V = P \cup I \cup O$, $P \cap I = P \cap O = \emptyset$ and $|I| = |O| = k$,*
- *$G[P] := (P, E^P)$ has fanin and fanout 1 (cf. Definition 1),*

- $E = E^P \cup E^{io}$ *with* E^{io} *satisfying:*
  - *(Soundness) Every* $e \in E^{io}$ *satisfies either* $e = (in, p)$ *or* $e = (p, out)$ *for* $p \in P, in \in I, out \in O$,
  - *(Completeness) For every source (resp. sink)* $p \in P$*, there exists at most one* $in \in I$ *(resp.* $out \in O$*) such that* $(in, p) \in E^{io}$ *(resp.* $(p, out) \in E^{io}$*).*

The set of all Augmented k-Way Valiant Blocks for P, I, O *is denoted by* $\mathcal{B}_k(P, I, O)$.

The sets I and O represent the inputs and outputs of a Superpole. Note that $I \cap O \neq \emptyset$ is allowed by above definition. This is needed because the Superpoles in the construction of Liu et al. [LYZ+21] uses "merged nodes" that function as input and output at the same time. Since a Superpole shall guarantee the routing inside itself, we define the Superpole as follows.



(a) Augmented 2-way Valiant Block.

(b) Valiant's 2-way Superpole that edge-embeds graph (a).

(c) Valiant's 4-way Superpole.

Figure 3.1: Augmented 2-Way Valiant Block and Valiant's Superpole constructions.

Definition 3.1.2 (k-Way Superpole)**.** *A k-Way Superpole is a graph $G = (V, E, P, \mathcal{P}, \mathcal{I}, \mathcal{O})$, where the following conditions hold:*

- *$P = \mathcal{P} \cup \mathcal{I} \cup \mathcal{O}$ with $|\mathcal{I}| = |\mathcal{O}| = k$ and $\mathcal{P} \cap \mathcal{I} = \mathcal{P} \cap \mathcal{O} = \emptyset$.*
- *G can edge-embed every $G' \in \mathcal{B}_k(\mathcal{P}, \mathcal{I}, \mathcal{O})$.*

We denote the input recursion points $\mathcal{I}$ of a k-Way Superpole as $\{in_1, in_2, ..., in_k\}$ and the output recursion points $\mathcal{O}$ as $\{out_1, out_2, ..., out_k\}$. These nodes serve as as the inputs and outputs to the Superpole and will be the poles of the next recursion, i.e., of the next sub EUG. We neither require the sets $\mathcal{I}$ and $\mathcal{O}$ to be disjoint nor that the recursion points of different Superpoles must be disjoint. Thus, it is possible that two different Superpoles share the same or partly the same recursion points. Both constructions of Valiant and Liu et al. heavily use this possibility, which is also called "merged nodes" since one node functions as different recursion points for possibly different Superpoles. In case of Valiant's construction, the output recursion points of the i-th Superpole are merged with the input recursion points of the $(i+1)$-th Superpole.

Before we begin describing the construction, we need to make one technical but natural assumption. We assume that the edge-embeddings of the Superpoles fulfill the restriction that there is no path in the edge-embedding that uses a pole or a recursion point as an intermediary node. The reason for this pole restriction is that when translating the EUG into a UC, we need to instantiate the poles with Boolean gates and can not just pass the input value. The restrictions on the recursion points is needed for the EUG construction of Liu et al. (cf. proof of Theorem 3). Note that it does not make any sense to use a recursion point or a pole as an intermediary node in the Superpole constructions we will see in this work or in the Superpoles used in practice [Val76; KS16; LMS16; GKS17; ZYZL19; AGKS20; LYZ+21]. However, pathological instances violating these restrictions could be build and we have to exclude them for above reasons.

Algorithm 2: VALIANT(P,k)

Input : Poles $P := \{p_1,...,p_n\}$, split parameter k
Output : $\Gamma_1(n)$ EUG $G = (V,E,P,G^*,G^1,...,G^k)$

$V \leftarrow \emptyset$, $E \leftarrow \emptyset$, $G^* \leftarrow \emptyset$
$\mathcal{O}^{s^0} \leftarrow$ *create k dummy nodes*
for $i \leftarrow 1$ **to** $\lceil \frac{n}{k} \rceil$ **:**
    $\mathcal{P}^{s^i} \leftarrow \{p_{k(i-1)+1},...,p_{ki}\}$
    $s^i = (V^{s^i}, E^{s^i}, P^{s^i}, \mathcal{P}^{s^i}, \mathcal{I}^{s^i}, \mathcal{O}^{s^i}) \leftarrow$ CREATESUPERPOLE$(\mathcal{P}^{s^i}, \mathcal{O}^{s^{i-1}}, k)$ `// Use` $\mathcal{O}^{s^{i-1}}$ `as input recursion points to this Superpole (cf. Figure 3.1)`
    $G^* \leftarrow G^* \cup \{s^i\}$
    $V \leftarrow V \cup V^i, E \leftarrow E \cup E^i$
for $i \leftarrow 1$ **to** k **:**
    if $n \leq k$ **:**
        $G^i \leftarrow (\emptyset,...,\emptyset)$ `// Recursion base`
    else
        `// Take the` i`-th output recursion point of each Superpole (but the last) as the poles for the next sub EUG`
        $P^i \leftarrow \{\mathcal{O}^{s^1}[i], \mathcal{O}^{s^2}[i], ..., \mathcal{O}^{s^{\lceil \frac{n}{k} \rceil - 1}}[i]\}$
        $G^i = (V^i, E^i, ...) \leftarrow$ VALIANT(P^i, k)
        $V \leftarrow V \cup V^i, E \leftarrow E \cup E^i$
return $G = (V,E,P,G^*,G^1,...,G^k)$

Definition 3.1.3 (Valiant EUG)**.** *A Valiant EUG $G = (V,E,P,G^*,G^1,...,G^k)$ is a graph that is created by Algorithm 2 (*VALIANT*). We also use the notation* VALIANT$_k(n)$ *for a Valiant EUG with n poles and split parameter k if the concrete poles are not relevant for the statement.*

A depiction of Valiant's EUG construction is given in Figure 3.2. The algorithm CREATESUPERPOLE$(P,\mathcal{O},k)$ creates a Superpole with poles P, input recursion points $\mathcal{O}$, and split parameter k. A depiction of the Superpoles used in Valiant's construction for $k \in \{2,4\}$ can be seen in Figure 3.1. We refrain from providing the proof that these Superpole constructions are indeed correct since the proof would be tedious and purely technical. As stated above these output recursion points of the $(i-1)$-th Superpole are also used as the input recursion points of the i-th Superpole. This results in reducing our problem to k EUGs of size $\lceil \frac{n}{k} \rceil - 1$. The creation of the first output recursion points $\mathcal{O}^{s^0}$ is a technical trick and not needed because these vertices will never be used. But, this simplifies the definition of the algorithm by avoiding a case distinction.

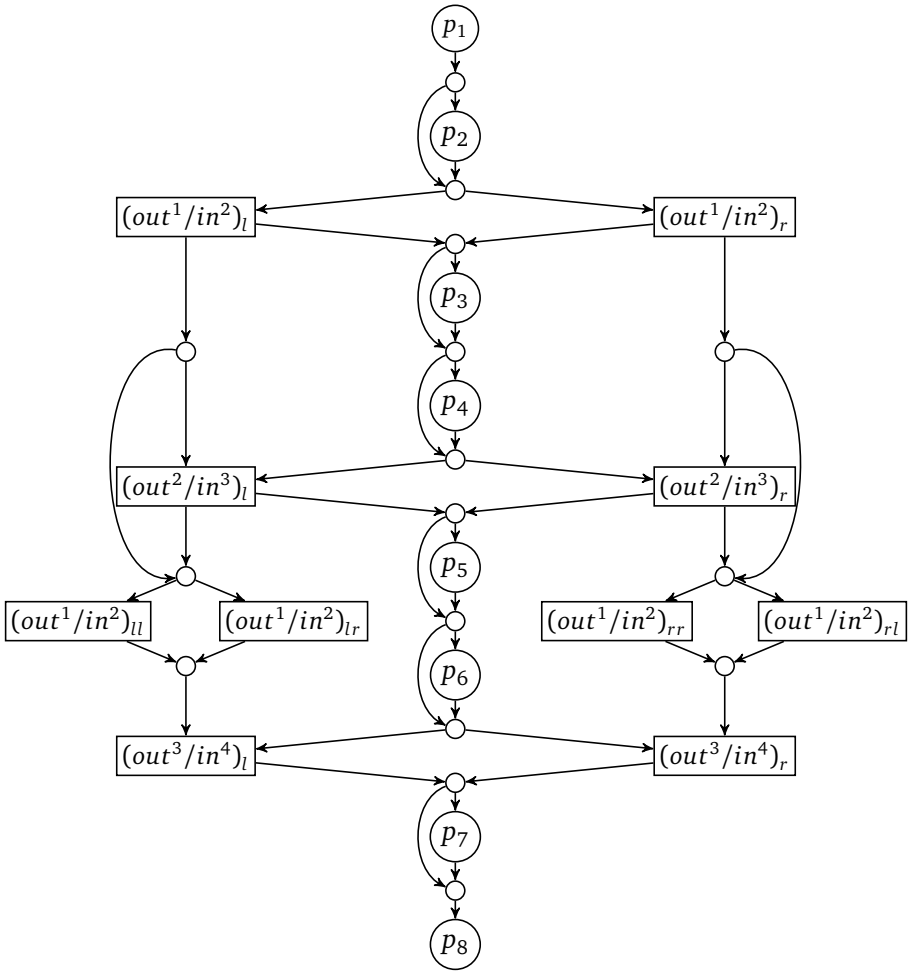


Figure 3.2: Valiant's 2-Way-Split construction for 8 poles. Note that unnecesserary head and tail nodes of each EUG and sub EUG were removed.

Proposition 2. *Let $G = (V, E, P, G^*, G^1, ..., G^k)$ be a Valiant EUG with $|P| = n$ sufficiently large, then*

$$|G| \leq \frac{|SP_k|}{k \log_2(k)} n \log_2(n) + \mathcal{O}(n),$$

where $|SP_k|$ denotes the size of a k-Way Superpole without recursion points and with k poles.

Proof. By definition of Algorithm 2, we add $\lceil \frac{n}{k} \rceil$ k-Way Superpoles (each Superpole has k poles except for the last with possibly less) and k $\textsc{Valiant}_k(\lceil \frac{n}{k} \rceil - 1)$ graphs for $n > k$. This yields

$$|\textsc{Valiant}_k(n)| \leq \lceil \frac{n}{k} \rceil |SP_k| + k \, |\textsc{Valiant}_k(\lceil \frac{n}{k} \rceil - 1)|$$

$$
\begin{aligned}
&\leq (\frac{n}{k}+1)|SP_k| + k\ |\textsc{Valiant}_k(\frac{n}{k})| \qquad (3.1)\\
&\overset{(3.1)}{\leq} (\frac{n}{k}+1)|SP_k| + k((\frac{n}{k^2}+1)|SP_k| + k\ |\textsc{Valiant}_k(\frac{n}{k^2})|)\\
&= (\frac{n}{k}+1)|SP_k| + (\frac{n}{k}+k)|SP_k| + k^2\ |\textsc{Valiant}_k(\frac{n}{k^2})|\\
&= \sum_{i=0}^{1}(\frac{n}{k}+k^i)|SP_k| + k^2\ |\textsc{Valiant}_k(\frac{n}{k^2})|.
\end{aligned}
$$

Iterating above inequality and using $|\textsc{Valiant}_k(m)| \leq |SP_k|$ for $m \leq k$ $(*)$ yields (with the smallest $x \in \mathbb{N}$ such that $\frac{n}{k^x} \leq k$)

$$
\begin{aligned}
|\textsc{Valiant}_k(n)| &\leq \sum_{i=0}^{x-1}(\frac{n}{k}+k^i)|SP_k| + k^x\ |\textsc{Valiant}_k(\frac{n}{k^x})|\\
&\overset{(*)}{\leq} \sum_{i=0}^{x-1}(\frac{n}{k}+k^i)|SP_k| + k^x\ |SP_k|\\
&= (x\frac{n}{k} + \sum_{i=0}^{x}k^i)|SP_k|\\
&\overset{\text{geometric series}}{=} (x\frac{n}{k} + \frac{k^{x+1}-1}{k-1})|SP_k|.
\end{aligned} \qquad (3.2)
$$

It follows $x = \lceil \log_k(n) - 1 \rceil$. Thus, (3.2) becomes

$$
\begin{aligned}
\lceil \log_k(n)-1 \rceil \frac{n}{k}|SP_k| + \frac{k^{\lceil \log_k(n)-1 \rceil+1}-1}{k-1}|SP_k| &\leq \log_k(n)\frac{n}{k}|SP_k| + \frac{k^{\log_k(n)+1}-1}{k-1}|SP_k|\\
&= \log_k(n)\frac{n}{k}|SP_k| + \frac{nk-1}{k-1}|SP_k|\\
&= \frac{|SP_k|}{k\log_2(k)}n\log_2(n) + \mathcal{O}(n).
\end{aligned}
$$

□

Using the Superpoles depicted in Figure 3.1 with $|SP_2| = 5$ and $|SP_4| = 19$, we get upper bounds for the leading coefficient in Proposition 2 of $2.5n\log_2 n$ and $2.375n\log_2 n$ for $\Gamma_1(n)$ EUGs. Merging two instances of these EUGs results in upper bounds of $5n\log_2 n$ and $4.75n\log_2 n$. In [ZYZL19], a new 4-Way Superpole with 18 nodes was proposed, which leads to an upper bound of $4.5n\log_2 n$.

To prove that Valiant's construction is indeed an EUG, we define the edge-embedding algorithm EdgeEmbedding (Algorithm 3) on page 19. This algorithm can also be used to get an edge-embedding for the weak version of the EUG construction of Liu et al. [LYZ+21] (cf. Section 3.2). In Theorem 4, we will see that this is sufficient to get an edge-embedding for the (optimized) construction of Liu et al. The algorithm consists of the following four steps:

Step 1: Create a $\Gamma_k(\lceil \frac{n}{k} \rceil - 1)$ graph that contains the edges between the input and output recursion points of the Superpoles that need to be embedded in the next recursion step. Then, split this graph into k $\Gamma_1(\lceil \frac{n}{k} \rceil - 1)$ graphs by Corollary 2. This determines which edges will be embedded into which sub EUGs and yields an Augmented k-Way Valiant Block for every Superpole. They contain the path information for the Superpoles. This is done by the algorithm PATHFINDING on page 20, which uses the idea of the edge-embedding algorithm in [GKS17].

Step 2: Embed the Augmented k-Way Valiant Blocks into the Superpoles. This is done by an edge-embedding algorithm SUPERPOLEEDGEEMBEDDING for the used Superpoles. As stated before, the definition of a concrete algorithm with a correctness proof would be too complex and time consuming. However, an open source implementation of the edge-embedding algorithms for the Superpoles described in [Val76; LMS16; ZYZL19; AGKS20] can be found in [AGKS20, §4.1 Block Edge-Embedding].

Step 3: Embed the k $\Gamma_1(\lceil \frac{n}{k} \rceil - 1)$ graphs into the k sub EUGs. This is done by (recursively) calling EDGEEMBEDDING (Algorithm 3) on the sub EUGs.

Step 4: Combine the obtained edge-embeddings from the Superpoles and sub EUGs. This is done by COMBINEEDGEEMBEDDINGS (Algorithm 5).

Algorithm 3: EDGEEMBEDDING(G, G')

Input : Valiant EUG [Val76] **or** weak Liu EUG [LYZ+21]
$G = (V, E, P, G^* = \{s^1, ..., s^{\lceil \frac{n}{k} \rceil}\}, G^1, ..., G^k)$, $G' = (P, E') \in \Gamma_1(n)$

Output : Edge-embedding ψ of G' into G

1 **if** $V = \emptyset$ **:**
2 **return** *empty map* ψ `// Recursion base`
3 $A^1, ..., A^{\lceil \frac{n}{k} \rceil}, R^1, ..., R^k \leftarrow$ PATHFINDING(G, G')
4 **for** $i \leftarrow 1$ **to** $\lceil \frac{n}{k} \rceil$ **:**
5 $\psi^{A^i} \leftarrow$ SUPERPOLEEDGEEMBEDDING(s^i, A^i)
6 **for** $j \leftarrow 1$ **to** k **:**
7 $\psi^{R^j} \leftarrow$ EDGEEMBEDDING(G^j, R^j)
8 $\psi \leftarrow$ COMBINEEDGEEMBEDDINGS $(G, G', \psi^{A^1}, ..., \psi^{A^{\lceil \frac{n}{k} \rceil}}, \psi^{R^1}, ..., \psi^{R^k})$
9 **return** ψ

Algorithm 4: PATHFINDING(G, G')

Input : Valiant EUG [Val76] **or** weak Liu EUG [LYZ+21]
$G = (V, E, P, G^* = \{s^1, ..., s^{\lceil \frac{n}{k} \rceil}\}, G^1, ..., G^k)$, $G' = (P, E') \in \Gamma_1(n)$
Output : Augmented k-Way Valiant Blocks $A^1, A^2, ..., A^{\lceil \frac{n}{k} \rceil}$,
$R^1, R^2, ..., R^k \in \Gamma_1(\lceil \frac{n}{k} \rceil - 1)$ for Valiant **or** $\Gamma_1(\lceil \frac{n}{k} \rceil)$ for weak Liu

for $i \leftarrow 1$ **to** $\lceil \frac{n}{k} \rceil$:
  Let $\mathcal{I}^i, \mathcal{O}^i, \mathcal{P}^i$ *be the input/output recursion points and the poles of the Superpole* s^i
  $A^i = (V^{A^i}, E^{A^i}) \leftarrow (\mathcal{I}^i \cup \mathcal{O}^i \cup \mathcal{P}^i, \emptyset)$
// R will be the graph that describes which edges between the recursion points of the Superpoles will embedded in the sub EUGs.
$\tilde{V}^R \leftarrow$ *create an input node* in_i *for each Superpole* s^i *but the first and an output node* out^i *for each Superpole but the last*
// In [Val76]: $in_{i-1} = out_i$. In [LYZ+21]: $in_i = out_i$.
$\tilde{E}^R \leftarrow \emptyset$
$\tilde{R} \leftarrow (\tilde{V}^R, \tilde{E}^R)$
foreach $e = (u, v) \in E'$:
  Let s^i *and* s^j *be the Superpoles in which* u *and* v *are a pole*
  if $i \neq j$:
    $\tilde{E}^R \leftarrow \tilde{E}^R \cup \{(out^i, in^j)\}$
$\tilde{R}^1 = (\tilde{V}^R, \tilde{E}^1_R), \tilde{R}^2 = (\tilde{V}^R, \tilde{E}^2_R), ..., \tilde{R}^k = (\tilde{V}^R, \tilde{E}^k_R) \leftarrow$ *k-coloring of* $\tilde{R}$ *(cf. Corollary 2)*
for $l \leftarrow 1$ **to** k :
  // We now 'un-merge' these 'merged' recursion points to get the concrete recursion points
  Let $\mathcal{IO}_l$ *be the poles of* G^l // These poles are exactly the l-th input and output recursion points in^l_i, out^l_i of each Superpole s^i.
  $R^l = (V^l_R, E^l_R) \leftarrow (\mathcal{IO}_l, \emptyset)$
  foreach $e = (out^i, in^j) \in \tilde{E}^l_R$:
    $E^l_R \leftarrow E^l_R \cup \{(out^i_l, in^j_l)\}$
foreach $e = (u, v) \in E'$:
  Let s^i *and* s^j *be the Superpoles in which* u *and* v *are a pole.*
  if $i = j$:
    $E^{A^i} \leftarrow E^{A^i} \cup \{(u, v)\}$
  else
    Choose $x \in [k]$ *such that* $(out^i_x, in^j_x) \in R^x$ *and not marked*
    mark (out^i_x, in^j_x) *in* R^x
    $E^{A^i} \leftarrow E^{A^i} \cup \{(u, out^i_x)\}$
    $E^{A^j} \leftarrow E^{A^j} \cup \{(in^j_x, v)\}$
return $A^1, A^2, ..., A^{\lceil \frac{n}{k} \rceil}, R^1, R^2, ..., R^k$

Algorithm 5: CombineEdgeEmbeddings$(G, G', \psi^{A^1}, ..., \psi^{A^{\lceil \frac{n}{k} \rceil}}, \psi^{R^1}, ..., \psi^{R^k})$

Input : Valiant EUG [Val76] **or** weak Liu EUG [LYZ+21]
$G = (V, E, P, G^* = \{s^1, ..., s^{\lceil \frac{n}{k} \rceil}\}, G^1, ..., G^k)$, $G' = (P, E') \in \Gamma_1(n)$,
edge-embeddings $\psi^{A^1}, ..., \psi^{A^{\lceil \frac{n}{k} \rceil}}$ of Augmented k-Way Valiant Blocks
$A^1, ..., A^{\lceil \frac{n}{k} \rceil}$ into Superpoles $s^1, ..., s^{\lceil \frac{n}{k} \rceil}$ and edge-embeddings $\psi^{R^1}, ..., \psi^{R^k}$ of
$R^1, ..., R^k \in \Gamma_1(\lceil \frac{n}{k} \rceil - 1)$ for Valiant **or** $\Gamma_1(\lceil \frac{n}{k} \rceil)$ for weak Liu into $G^1, ..., G^k$
given by PathFinding(G, G')

Output : Edge-embedding ψ of G' into G

1 **foreach** $(u, v) \in E'$ **:**
2 *Let s^i and s^j be the Superpoles in which u and v are a pole*
3 **if** $i = j$ **:**
4 $\psi((u, v)) \leftarrow \psi^{A^i}((u, v))$
5 **else**
  `// There is exactly one edge` $(u, out^i_m) \in A^i$ `for some` $m \in k$
6 $out^i_m \leftarrow$ *the output recursion point to which u is routed inside A^i*
  `// There is exactly one edge` $(v, in^j_m) \in A^j$
7 $in^j_m \leftarrow$ *the input recursion point which is routed to v inside A^j*
  `// '+' denotes the concatenation of paths`
8 $\psi((u, v)) \leftarrow \psi^{A^j}((u, out^i_m)) + \psi^{G^m}((out^i_m, in^j_m)) + \psi^{A^i}((in^j_m, v))$
9 **return** ψ

Using PathFinding to get the path information $R^1, ..., R^k$ and Superpole routings $A^1, ..., A^{\lceil \frac{n}{k} \rceil}$, the following Lemma guarantees that we can edge-embed $R^1, ..., R^k$ into the corresponding sub EUGs $G^1, ..., G^k$ and $A^1, ..., A^{\lceil \frac{n}{k} \rceil}$ into the Superpoles $s^1, ..., s^{\lceil \frac{n}{k} \rceil}$.

Lemma 1. *Let $G = (V, E, P, G^*, G^1, ..., G^k)$ be a Valiant EUG with n poles, $G' = (P, E') \in \Gamma_1(n)$ the order preserving graph to be embedded and $\mathcal{IO}_i$ the set of the i-th input recursion points of each Superpole $s^l \in G^*$ (except for $l = 1$) for $i \in [k]$. Let $\mathcal{I}^j, \mathcal{O}^j, \mathcal{P}^j$ be the input recursion points, output recursion points, and the poles of the Superpole $s^j \in G^*$ for $j \in [\lceil \frac{n}{k} \rceil]$. Then,* PathFinding$(G, G')$ *outputs $A^j \in \mathcal{B}_k(\mathcal{P}^j, \mathcal{I}^j, \mathcal{O}^j)\ \forall\ j \in [\lceil \frac{n}{k} \rceil]$ and $R^i = (\mathcal{IO}_i, E_i) \in \Gamma_1(\lceil \frac{n}{k} \rceil - 1)$ that are order preserving $\forall\ i \in [k]$.*

Proof. We begin by showing that $R^1, R^2, ..., R^k \in \Gamma_1(\lceil \frac{n}{k} \rceil - 1)$.

Consider $\tilde{R} = (\tilde{V}^R, \tilde{E}^R)$ (with $\tilde{E}^R$ being a multi set of edges), where V^R is the set that contains one node in^l per Superpole $s^l \in G^*$ (except for the first Superpole) for $l \in \{2, ..., \lceil \frac{n}{k} \rceil\}$. Each node represents an merged input recursion point and this graph will contain the edges which are embedded in the sub EUGs (lines 4-6). Note that the l-th input recursion points are the $(l-1)$-th output recursion points in Valiant's construction, i.e., $in^l = out^{l-1}\ \forall\ l \in \{2, ..., \lceil \frac{n}{k} \rceil\}$. Thus, $|\tilde{V}^R| = \lceil \frac{n}{k} \rceil - 1$.

An edge (out^l, in^m) for $l, m \in [\lceil \frac{n}{k} \rceil]$ is added to E^R if and only if there is an edge from a vertex $u \in s^l$ to $v \in s^m$ with $l \neq m$ (lines 7-10). Since a Superpole has at most k poles and G' has fanin/fanout 1, there can be at most k incoming and k outgoing edges into each merged input/output recursion point. Because G' is order preserving and the order of poles in G^* is maintained, every edge $(out^l, in^m) \in E^R$ implies $l < m$. In particular, R is acyclic and order preserving. Thus, $R \in \Gamma_k(\lceil \frac{n}{k} \rceil - 1)$.

We now use Corollary 2 to get $\tilde{R}^i \in \Gamma_1(\lceil \frac{n}{k} \rceil - 1) \; \forall \; i \in [k]$ (line 11) and replace the merged nodes $\tilde{V}^R$ of each $\tilde{R}^i$ by the i-th input (resp. $(i-1)$-th output) recursion points $\mathcal{IO}_i$ to get R^i (lines 12-16).

Now consider $A^j = (V^{A^j}, E^{A^j})$ for $j \in \lceil \frac{n}{k} \rceil$. By line 3 in PATHFINDING, we have $V^{A^j} = \mathcal{P}^j \cup \mathcal{I}^j \cup \mathcal{O}^j$. Assume there is a vertex $v \in V^{A^j}$ with $\deg^{-/+}_{A^j}(v) > 1$.

Case 1: $v \in \mathcal{P}^j$
Edges in A^j from/to v are added if and only if there are corresponding edges from/to v in G' (lines 17-25). But then $\deg^{-/+}_{A^j}(v) > 1$ would imply that v has in- or outdegree greater than 1 in G'. Thus, G' can not have fanin or fanout 1. ↯

Case 2: $v \in \mathcal{I}^j \cup \mathcal{O}^j$
In this case, edges from/to v are added if and only if there is a corresponding edge from/to v in R^x for some $x \in [k]$ (line 22). Since R^x has fanin and fanout 1, one can apply the same argument as above.

Therefore, $A^j \in \Gamma_1(|\mathcal{P}^j \cup \mathcal{I}^j \cup \mathcal{O}^j|)$. Define $E^{io}_{A^j}$ to be the set of all edges in E^{A^j} that connect poles and inputs or outputs. Let $E^P_{A^j}$ be the set of edges between poles. Since there are no edges between input and output recursion points, $E^{A^j} = E^P_{A^j} \cup E^{io}_{A^j}$ with $E^{io}_{A^j} \cap E^P_{A^j} = \emptyset$. Then, each $e \in E^{io}_{A^j}$ is of the form $e = (in, p)$ or $e = (p, out)$ for $p \in \mathcal{P}^j, in \in \mathcal{I}^j, out \in \mathcal{O}^j$ (Soundness). Note that A^j having fanin and fanout 1 implies (Completeness). Furthermore, $A^j[P]$ is order preserving since edges between poles in A^j directly correspond to edges in G'. Thus, $A^j \in \mathcal{B}(\mathcal{P}^j, \mathcal{I}^j, \mathcal{O}^j)$. □

Now we can prove that Valiant's construction indeed yields a Γ_1 EUG.

Theorem 2. *Let $G = (V, E, P, G^*, G^1, ..., G^k)$ be a Valiant EUG with n poles. Then G is an EUG for $\Gamma_1(n)$. In particular,* EDGEEMBEDDING*(G, G') yields an edge-embedding from G' into G for all order preserving $G' = (P, E') \in \Gamma_1(n)$.*

Proof. We prove the statement by induction over the number of poles n.

Induction base: $0 < n \leq k$
Note that by definition of VALIANT(P, k), G consists of only one Superpole with n poles. Now consider EDGEEMBEDDING(G, G'). Since G only has one Superpole and no sub EUGs, PATHFINDING returns $A^1 \in \mathcal{B}_k(P, \mathcal{I}^1, \mathcal{O}^1)$ and $R^1 = R^2 = ... = R^k = (\emptyset, \emptyset)$. Note that all poles

of G' are contained in A^1 which is to be embedded into s^1 (resp. G) in line 5. Let $(u,v) \in E'$. We show that $\psi((u,v))$ yields a u-v-path in G. The edge (u,v) is added to A^1 in line 20 of PATHFINDING. Since A^1 is edge-embedded into s^1, $\psi^{A^1}((u,v))$ is already a u-v-path in G. COMBINEEDGEEMBEDDINGS in line 8 then sets $\psi((u,v)) = \psi^{A^1}((u,v)) \; \forall \; (u,v) \in E'$. Since the paths of the image of ψ^{A^1} are disjoint, also the paths of the image of ψ are disjoint. Therefore, ψ is an edge-embedding from G' into G.

Induction step: $n-1 \rightsquigarrow n$

We begin by calling PATHFINDING(G, G') to get $A^j \in \mathcal{B}_k(\mathcal{P}^j, \mathcal{I}^j, \mathcal{O}^j)$ for each Superpole $s^j \in G^*, j \in [\lceil \frac{n}{k} \rceil]$ and order preserving $R^i = (\mathcal{IO}_i, E_i) \in \Gamma_1(\lceil \frac{n}{k} \rceil - 1) \; \forall \; i \in [k]$ (line 3, by Lemma 1). By definition of a Superpole, we can edge embed each A^j into $s^j \in G^* \; \forall \; j \in [\lceil \frac{n}{k} \rceil]$ (line 5). Since the sub EUGs $G^1, ..., G^k$ have $\lceil \frac{n}{k} \rceil - 1$ poles , we can edge embed each R^i, by induction hypothesis, into $G^i \; \forall \; i \in [k]$ (line 7). This yields corresponding maps ψ^{A^j} and ψ^{G^i} for $j \in [\lceil \frac{n}{k} \rceil]$ and $i \in [k]$. We now show how this gives rise to an edge-embedding ψ of G' into G. Let $(u,v) \in E'$.

Case 1: u and v are in the same Superpole $s^j \in G^*$

Then, we set $\psi((u,v)) = \psi^{A^j}((u,v))$. Since $(u,v) \in A^j$ (line 20 in PATHFINDING) and ψ^{A^j} is a correct edge-embedding, this yields a correct u-v-path in G.

Case 2: $u \in s^j$ and $v \in s^l$ for $j \neq l$

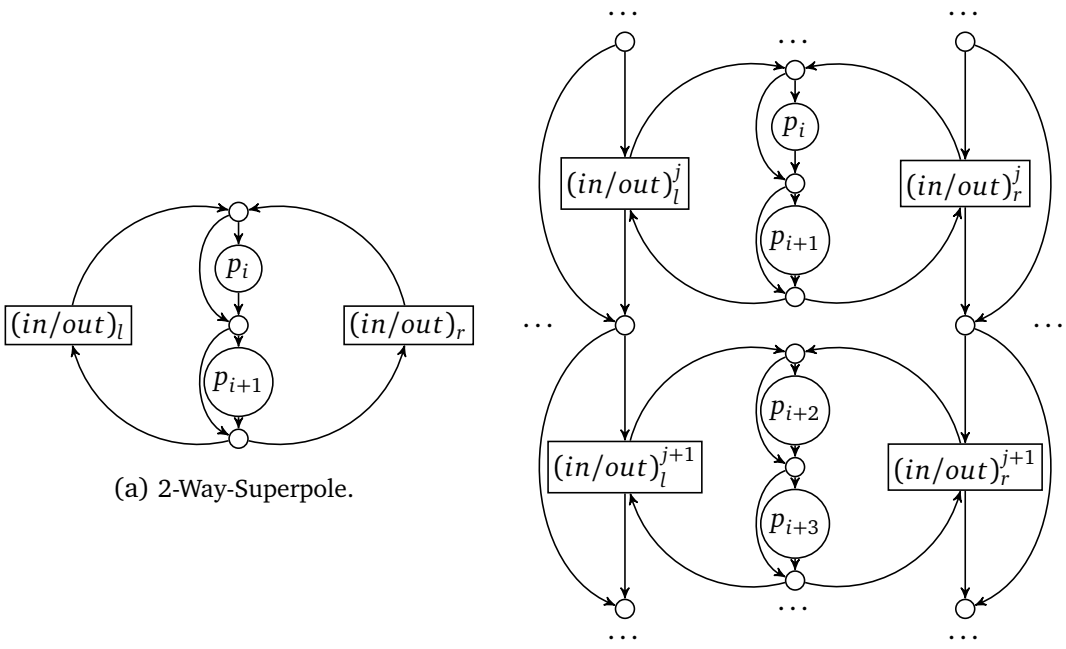
Then, there must be edges $(u, out^j_m) \in A^j, (out^j_m, in^l_m) \in R^m$, and $(in^l_m, v) \in A^l$ for some $m \in [k]$ (lines 9-10 and 21-25 in PATHFINDING). Set

$$\psi((u,v)) = \psi^{s^j}((u, out^j_m)) + \psi^{G^m}((out^j_m, in^l_m)) + \psi^{s^l}((in^l_m, v)),$$

where '+' denotes the concatenation of the paths. Note that s^j, G^m, and s^l are node-disjoint (except for the recursion points). Thus, $\psi((u,v))$ contains no cycles, and $\psi((u,v))$ is a u-v-path in G.

Furthermore, all paths in the image of ψ are edge-disjoint because all Superpoles and sub EUGs are edge-disjoint. The described creation of ψ is done by COMBINEEDGEEMBEDDINGS in line 8. □

3.2 Liu et al.'s Edge-Universal Graph Construction



(a) 2-Way-Superpole.

(b) Basic structure for the 2-Way split.

Figure 3.3: Superpole and basic structure of the 2-Way split construction of Liu et al.

As stated in the beginning, the EUG construction of Liu et al. [LYZ+21] relies on the same basic ideas as Valiant's EUG construction, but it does some smart optimizations to the construction to reduce the size of the (until then) smallest size of $\sim 4.5n \log_2 n$ [ZYZL19] by 33% to $\sim 3n \log_2 n$. Similar to the original paper, we begin by defining an intermediate construction, a so called weak version. This is an EUG that is neither more efficient than Valiant's original construction nor is it acyclic, but we can optimize out the redundancy in order to remove the cycles and reduce the size.

3.2.1 The Weak Version

The two main components of [LYZ+21] are still the recursive structure and the Superpoles. However, this time we do not merge the output recursion points of the i-th Superpole with the input recursion points of the $(i+1)$-th Superpole, but instead we merge the input and output recursion points of each Superpole (cf. Figure 3.3). This results in the aforementioned disadvantages of a recursion size of $\lceil \frac{n}{k} \rceil$ instead of $\lceil \frac{n}{k} \rceil - 1$ and cycles within the Superpoles. The Superpole itself (without input/output recursion points) stays unaffected. An example of a weak Liu EUG is given in Figure 3.4 on page 26.

Algorithm 6: WEAKLIU(P, k)

Input : Poles $P := \{p_1, ..., p_n\}$, split parameter k
Output : $\Gamma_1(n)$ EUG $G = (V, E, P, G^*, G^1, ..., G^k)$

$V \leftarrow \emptyset$, $E \leftarrow \emptyset$, $G^* \leftarrow \emptyset$
for $i \leftarrow 1$ **to** $\lceil \frac{n}{k} \rceil$ **:**
  $\mathcal{P}^{s^i} \leftarrow \{p_{k(i-1)+1}, ..., p_{ki}\}$
  $s^i = (V^{s^i}, E^{s^i}, P^{s^i}, \mathcal{P}^{s^i}, \mathcal{I}^{s^i}, \mathcal{O}^{s^i}) \leftarrow$ CREATESUPERPOLE$(\mathcal{P}^{s^i}, k)$
  $G^* \leftarrow G^* \cup \{s^i\}$
  $V \leftarrow V \cup V^i, E \leftarrow E \cup E^i$
for $i \leftarrow 1$ **to** k **:**
  if $n \leq k$ **:**
    $G^i \leftarrow (\emptyset, ..., \emptyset)$ `// Recursion base`
  else
    `// Take the i-th output recursion point of each Superpole as the poles for the next sub EUG`
    $P^i \leftarrow \{\mathcal{O}^{s^1}[i], \mathcal{O}^{s^2}[i], ..., \mathcal{O}^{s^{\lceil \frac{n}{k} \rceil - 1}}[i], \mathcal{O}^{s^{\lceil \frac{n}{k} \rceil}}[i]\}$
    $G^i = (V^i, E^i, ...) \leftarrow$ WEAKLIU(P^i, k)
    $V \leftarrow V \cup V^i, E \leftarrow E \cup E^i$
return $G = (V, E, P, G^*, G^1, ..., G^k)$

Definition 3.2.1 (Weak Liu EUG)**.** *A weak Liu EUG $G = (V, E, P, G^*, G^1, ..., G^k)$ is a graph that is created by Algorithm 6 (*WEAKLIU*) on page 25. We also use the notation* WEAKLIU$_k(n)$ *for a weak Liu EUG with n poles and split parameter k if the concrete poles are not relevant for the statement.*

The proof that each weak Liu EUG is an EUG is very similar to the proof for Valiant's construction. The only difference is that a recursion size of $\lceil \frac{n}{k} \rceil$ is used instead of $\lceil \frac{n}{k} \rceil - 1$. Hence, we also use the same edge-embedding algorithm EDGEEMBEDDING on page 19 as with Valiant's construction. The following proofs are analogous to the prior proofs for Valiant's construction.

Lemma 2. *Let $G = (V, E, P, G^*, G^1, ..., G^k)$ be a weak Liu EUG [LYZ+21] with n poles, $G' = (P, E') \in \Gamma_1(n)$ the order preserving graph to be embedded and $\mathcal{IO}_i$ the set of the i-th input recursion points of each Superpole $s^l \in G^*$ (except for $l = 1$) for $i \in [k]$. Let $\mathcal{I}^j, \mathcal{O}^j, \mathcal{P}^j$ be the input recursion points, output recursion points, and the poles of the Superpole $s^j \in G^*$ for $j \in [\lceil \frac{n}{k} \rceil]$. Then,* PATHFINDING$(G, G')$ *outputs $A^j \in \mathcal{B}_k(\mathcal{P}^j, \mathcal{I}^j, \mathcal{O}^j)\ \forall\ j \in [\lceil \frac{n}{k} \rceil]$ and $R^i = (\mathcal{IO}_i, E_i) \in \Gamma_1(\lceil \frac{n}{k} \rceil)$ that are order preserving $\forall\ i \in [k]$.*

Proof. We begin by showing that $R^1, R^2, ..., R^k \in \Gamma_1(\lceil \frac{n}{k} \rceil)$.

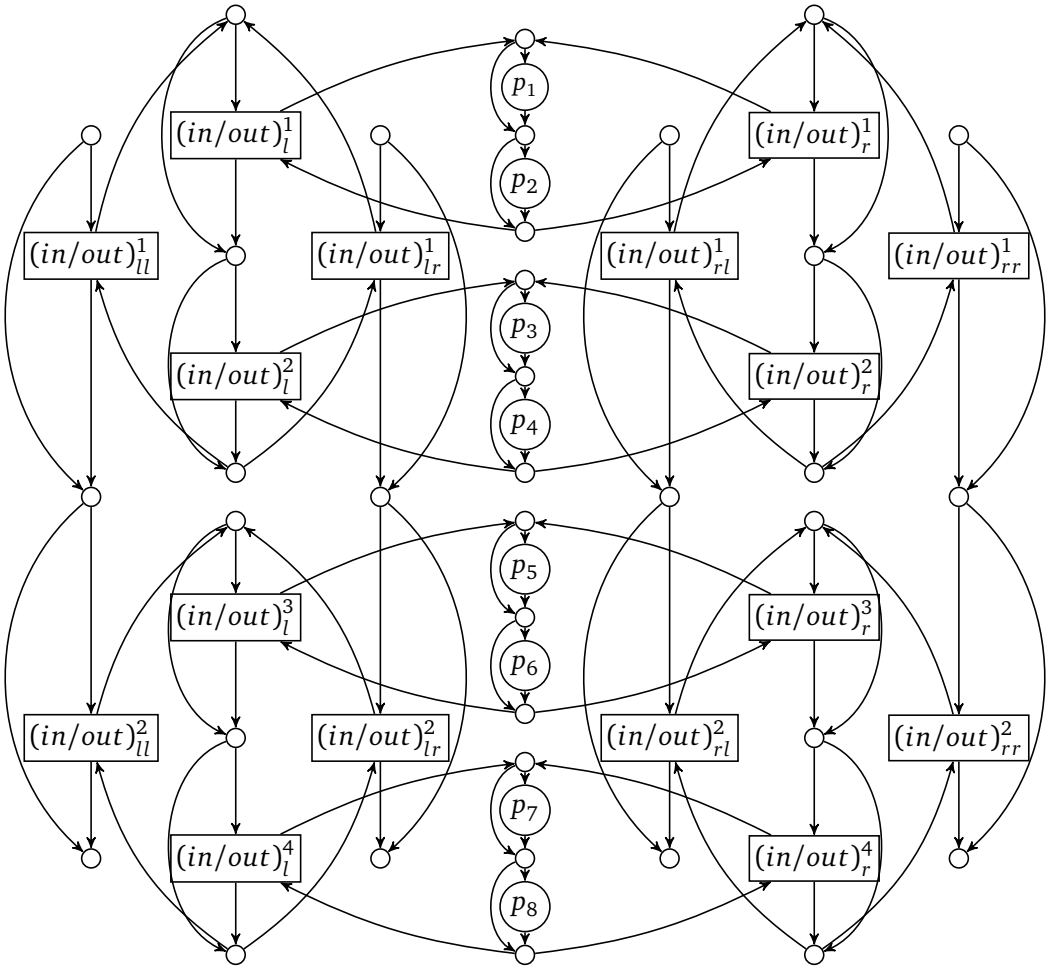


Figure 3.4: The complete $\Gamma_1(8)$ EUG with Liu's weak 2-Way-Split construction for 8 poles.

Consider $\tilde{R} = (\tilde{V}^R, \tilde{E}^R)$ (with $\tilde{E}^R$ being a multi set of edges), where V^R is the set that contains one node in^l per Superpole $s^l \in G^*$ for $l \in \{2, ..., \lceil \frac{n}{k} \rceil\}$. Each node represents an merged input recursion point and this graph will contain the edges which are embedded in the sub EUGs (lines 4-6). Note that the l-th input recursion points are the l-th output recursion points in the construction of Liu et al. [LYZ+21], i.e., $in^l = out^l \ \forall \ l \in \{1, ..., \lceil \frac{n}{k} \rceil\}$. Thus, $|\tilde{V}^R| = \lceil \frac{n}{k} \rceil$.

An edge (out^l, in^m) for $l, m \in [\lceil \frac{n}{k} \rceil]$ is added to E^R if and only if there is an edge from a vertex $u \in s^l$ to $v \in s^m$ with $l \neq m$ (lines 7-10). Since a Superpole has at most k poles and G' has fanin/fanout 1, there can be at most k incoming and k outgoing edges into each merged input/output recursion point. Because G' is order preserving and the order of poles in G^* is maintained, every edge $(out^l, in^m) \in E^R$ implies $l < m$. In particular, R is acyclic and order preserving. Thus, $R \in \Gamma_k(\lceil \frac{n}{k} \rceil)$.

We now use Corollary 2 to get $\tilde{R}^i \in \Gamma_1(\lceil \frac{n}{k} \rceil)\ \forall\ i \in [k]$ (line 11) and replace the merged nodes $\tilde{V}^R$ of each $\tilde{R}^i$ by the i-th input (resp. $(i-1)$-th output) recursion points $\mathcal{IO}_i$ to get R^i (lines 12-16).

Now consider $A^j = (V^{A^j}, E^{A^j})$ for $j \in \lceil \frac{n}{k} \rceil$. By line 3 in PATHFINDING, we have $V^{A^j} = \mathcal{P}^j \cup \mathcal{I}^j \cup \mathcal{O}^j$. Assume there is a vertex $v \in V^{A^j}$ with $\deg^{-/+}_{A^j}(v) > 1$.

Case 1: $v \in \mathcal{P}^j$
Edges in A^j from/to v are added if and only if there are corresponding edges from/to v in G' (lines 17-25). But then $\deg^{-/+}_{A^j}(v) > 1$ would imply that v has in- or outdegree greater than 1 in G'. Thus, G' can not have fanin or fanout 1. ↯

Case 2: $v \in \mathcal{I}^j \cup \mathcal{O}^j$
In this case, edges from/to v are added if and only if there is a corresponding edge from/to v in R^x for some $x \in [k]$ (line 22). Since R^x has fanin and fanout 1, one can apply the same argument as above.

Therefore, $A^j \in \Gamma_1(|\mathcal{P}^j \cup \mathcal{I}^j \cup \mathcal{O}^j|)$. Define $E^{io}_{A^j}$ to be the set of all edges in E^{A^j} that connect poles and inputs or outputs. Let $E^P_{A^j}$ be the set of edges between poles. Since there are no edges between input and output recursion points, $E^{A^j} = E^P_{A^j} \cup E^{io}_{A^j}$ with $E^{io}_{A^j} \cap E^P_{A^j} = \emptyset$. Then, each $e \in E^{io}_{A^j}$ is of the form $e = (in, p)$ or $e = (p, out)$ for $p \in \mathcal{P}^j, in\ \in \mathcal{I}^j, out \in \mathcal{O}^j$ (Soundness). Note that A^j having fanin and fanout 1 implies (Completeness). Furthermore, $A^j[P]$ is order preserving since edges between poles in A^j directly correspond to edges in G'. Thus, $A^j \in \mathcal{B}(\mathcal{P}^j, \mathcal{I}^j, \mathcal{O}^j)$. □

Theorem 3. *Let $G = (V, E, P, G^*, G^1, ..., G^k)$ be a weak Liu EUG with n poles. Then G is an EUG for $\Gamma_1(n)$. In particular,* EDGEEMBEDDING*(G, G') yields an edge-embedding from G' into G for all order preserving $G' = (P, E') \in \Gamma_1(n)$.*

Proof. We prove the statement by induction over the number of poles n.

Induction base: $0 < n \leq k$
Note that by definition of WEAKLIU(P, k), G consists of only one Superpole with n poles. Now consider EDGEEMBEDDING(G, G'). Since G only has one Superpole and no sub EUGs, PATHFINDING returns $A^1 \in \mathcal{B}_k(P, \mathcal{I}^1, \mathcal{O}^1)$ and $R^1 = R^2 = ... = R^k = (\emptyset, \emptyset)$. Note that all poles of G' are contained in A^1 which is to be embedded into s^1 (resp. G) in line 5. Let $(u, v) \in E'$. We show that $\psi((u, v))$ yields a u-v-path in G. The edge (u, v) is added to A^1 in line 20 of PATHFINDING. Since A^1 is edge-embedded into s^1, $\psi^{A^1}((u, v))$ is already a u-v-path in G. COMBINEEDGEEMBEDDINGS in line 8 sets $\psi((u, v)) = \psi^{A^1}((u, v))\ \forall\ (u, v) \in E'$. Since the paths of the image of ψ^{A^1} are disjoint, also the paths of the image of ψ are disjoint. Therefore, ψ is an edge-embedding from G' into G.

Induction step: $n - 1 \rightsquigarrow n$
We begin by calling PATHFINDING(G, G') to get $A^j \in \mathcal{B}_k(\mathcal{P}^j, \mathcal{I}^j, \mathcal{O}^j)$ for each Superpole $s^j \in$

$G^*, j \in [\lceil \frac{n}{k} \rceil]$ and order preserving $R^i = (\mathcal{IO}_i, E_i) \in \Gamma_1(\lceil \frac{n}{k} \rceil)\ \forall\ i \in [k]$ (line 3, by Lemma 1). By definition of a Superpole, we can edge embed each A^j into $s^j \in G^*\ \forall\ j \in [\lceil \frac{n}{k} \rceil]$ (line 5). Since the sub EUGs $G^1, ..., G^k$ have $\lceil \frac{n}{k} \rceil$ poles , we can edge embed each R^i, by induction hypothesis, into $G^i\ \forall\ i \in [k]$ (line 7). This yields corresponding maps ψ^{A^j} and ψ^{G^i} for $j \in [\lceil \frac{n}{k} \rceil]$ and $i \in [k]$. We now show how this gives rise to an edge-embedding ψ of G' into G. Let $(u, v) \in E'$.

Case 1: u and v are in the same Superpole $s^j \in G^*$

Then we set $\psi((u,v)) = \psi^{A^j}((u,v))$. Since $(u,v) \in A^i$ (line 20 in PATHFINDING) and ψ^{A^j} is a correct edge-embedding, this yields a correct u-v-path in G.

Case 2: $u \in s^j$ and $v \in s^l$ for $j \neq l$

Then there must be edges $(u, out^j_m) \in A^j, (out^j_m, in^l_m) \in R^m$ and $(in^l_m, v) \in A^l$ for some $m \in [k]$ (lines 9-10 and 21-25 in PATHFINDING). Set

$$\psi((u,v)) = \psi^{s^j}((u, out^j_m)) + \psi^{G^m}((out^j_m, in^l_m)) + \psi^{s^l}((in^l_m, v)),$$

where '+' denotes the concatenation of the paths. Note that s^j, G^m and s^l are node-disjoint (except for the recursion points). By assumption, we never use a Superpole edge-embedding that uses an input or output recursion point as an intermediary node. Since there are no paths between input and output recursion points in the Augmented k-Way Valiant Blocks (cf. (Soundness) in Definition 3.1.1), the only possible cycle in the EUG via the recursion points is not used. Thus, $\psi((u,v))$ contains no cycles, and $\psi((u,v))$ is a u-v-path in G.

Furthermore, all paths in the image of ψ are edge-disjoint because all Superpoles and sub EUGs are edge-disjoint. The described creation of ψ is done by COMBINEEDGEEMBEDDINGS in line 8. □

Next, we want to show that the weak construction of Liu et al. [LYZ+21] results in (asymptotically) the same size as Valiant's construction [Val76] with the same upper bound on the leading coefficient. To do that, we first need to calculate the number of Superpoles in each recursion step of the algorithm.

Lemma 3. *The number of poles in the i-th recursion step in Algorithm 6 (*WEAKLIU*) is given by*

$$\hat{n}_i = \begin{cases} \lceil \frac{n}{k^i} \rceil, & \text{if } \hat{n}_{i-1} > k, \\ 0, & \text{else} \end{cases} \qquad \forall\ i \in \{1, 2, \ldots\}.$$

Furthermore, we define $\hat{n}_0 = n$. Thus, the number of Superpoles in the i-th recursion step is given by $\bar{n}_i := \lceil \frac{\hat{n}_i}{k} \rceil\ \forall i \in \{1, 2, \ldots\}$.

Proof. We proof the statement by induction over the number of recursion steps.

Induction start: $i = 0$
Then, we obviously have $\hat{n}_0 = n$ poles.

Induction step: $i = l+1$

By induction hypothesis, it holds $\hat{n}_l = 0$ or $\hat{n}_l = \lceil \frac{n}{k^l} \rceil$. Trivially, in the first case $\hat{n}_{l+1} = 0$. So assume $\hat{n}_l = \lceil \frac{n}{k^l} \rceil$.
If $\hat{n}_l \leq k$, no recursion is called. Therefore, $\hat{n}_{l+1} = 0$. Otherwise, the number of poles in the $(l+1)$-th recursion is the number of Superpoles in the l-th recursion (line 11 in WeakLiu), i.e., $\lceil \frac{\hat{n}_l}{k} \rceil = \lceil \frac{\lceil \frac{n}{k^l} \rceil}{k} \rceil = \lceil \frac{n}{k^{l+1}} \rceil$.

The number of Superpoles $\bar{n}_i$ directly follows from the number of poles $\hat{n}_i$ since each Superpole contains k consecutive poles (except for the last Superpole). □

Proposition 3. *Let $G = (V, E, P, G^*, G^1, ..., G^k)$ be a weak Liu EUG with $|P| = n$ sufficiently large, then*

$$|G| \leq \frac{|SP_k|}{k \log_2(k)} n \log_2(n) + \mathcal{O}(n),$$

where $|SP_k|$ denotes the size of a k-Way Superpole without recursion points with k poles.

Proof. By definition of Algorithm 6(WeakLiu), we have

$$|\text{WeakLiu}_k(\bar{n}_{i-1})| \leq \begin{cases} |SP_k|, & \text{if } \bar{n}_{i-1} \leq k, \\ \bar{n}_i |SP_k| + k\ |\text{WeakLiu}_k(\bar{n}_i)|, & \text{else} \end{cases} \quad \forall\ i \in \{1, 2, \ldots\} \tag{3.3}$$

for the number of Superpoles in the i-th recursion step $\bar{n}_i$ like in Lemma 3. Iterating (3.3) yields (with the smallest $x \in \mathbb{N}$ such that $\lceil \frac{n}{k^x} \rceil \leq k$, i.e., $x = \lceil \log_k(n) - 1 \rceil$)

$$\begin{aligned}
|\text{WeakLiu}_k(n)| &\leq \bar{n}_0 |SP_k| + k\ |\text{WeakLiu}_k(\bar{n}_0)| \\
&\leq \bar{n}_0 |SP_k| + k(\bar{n}_1 |SP_k| + k\ |\text{WeakLiu}_k(\bar{n}_1)|) \\
&\leq \bar{n}_0 |SP_k| + k\bar{n}_1 |SP_k| + k^2(\bar{n}_2 |SP_k| + k\ |\text{WeakLiu}_k(\bar{n}_2)|) \\
&\leq \ldots \\
&\leq \sum_{i=0}^{x-1} k^i \bar{n}_i |SP_k| + k^x |SP_k| \\
&\overset{\text{Lemma 3}}{=} \sum_{i=0}^{x-1} k^i \lceil \frac{n}{k^{i+1}} \rceil |SP_k| + k^x |SP_k| \\
&\leq \sum_{i=0}^{x-1} k^i (\frac{n}{k^{i+1}} + 1) |SP_k| + k^x |SP_k| \\
&= (x \frac{n}{k} + \sum_{i=0}^{x} k^i) |SP_k| \\
&\overset{\text{geometric series}}{=} (x \frac{n}{k} + \frac{k^{x+1} - 1}{k - 1}) |SP_k|.
\end{aligned}$$

Thus,

$$\begin{aligned}
|\textsc{WeakLiu}(n)| &\leq (\lceil \log_k(n) - 1 \rceil \frac{n}{k} + \frac{k^{\lceil \log_k(n)-1 \rceil + 1} - 1}{k-1}) |SP_k| \\
&\leq (\log_k(n) \frac{n}{k} + \frac{k^{\log_k(n)+1} - 1}{k-1}) |SP_k| \\
&= (\log_k(n) \frac{n}{k} + \frac{kn-1}{k-1}) |SP_k| \\
&= \frac{|SP_k|}{k \log_2(k)} n \log_2(n) + \mathcal{O}(n).
\end{aligned}$$

□

Note that in the proof of the size of Valiant's construction (Proposition 2), we always bounded $\lceil \frac{n}{k} \rceil - 1$ from above by $\frac{n}{k}$. Thus, we get the same asymptotic bound on the size of the EUG. However, the concrete size of Valiant's EUG construction for large n is always smaller than the construction of Liu et al. [LYZ+21] since in the latter, every recursion step, compared to Valiant, has an additional pole.

3.2.2 The Optimized Version

We will now show how Liu et al. optimized their weak construction such that the EUG becomes acyclic and its size is bounded by $\sim 3n \log_2(n)$. For each edge (u, r_x) that connects a node u of a Superpole with one of its recursion points r_x (where x denotes the sub EUG G^x in which r_x is a pole), we replace this edge by an edge (u, w) with w being the unique successor of r_x in G^x. We proceed analogously with edges connecting recursion points to nodes in a Superpole and replace them by edges connecting the predecessor of the recursion point to the node. Because all recursion points are short-circuited, we can remove them. This saves us k nodes per k-Way Superpole. A depiction of this procedure can be seen in Figure 3.5a on page 32.

Note that after short-circuiting the weak Liu EUG G, there are no cycles left in any Superpole of G. Thus, this optimized EUG is acyclic.

Definition 3.2.2 (Liu EUG)**.** *A Liu EUG is a short-circuited weak Liu EUG, i.e., it is given by Algorithm 7 on page 31 (*ShortCircuit*) for a weak Liu EUG G and denoted by* $\textsc{Liu}(P, k)$ *for pole set P and split parameter k, resp.* $\textsc{Liu}(n)_k$ *for n poles if the concrete poles are not relevant for the statement.*

The following theorem shows that to edge-embed Γ_1 graphs into a Liu EUG, it is sufficient to edge-embed the graph into the weak Liu EUG (by Algorithm 3), and then, replace the short-circuited edges.

Algorithm 7: ShortCircuit(G)

Input : $\Gamma_1(n)$ weak Liu EUG $G = (V, E, P, G^*, G^1, ..., G^k)$
Output : $\Gamma_1(n)$ Liu EUG $G = (V, E, P, G^*, G^1, ..., G^k)$

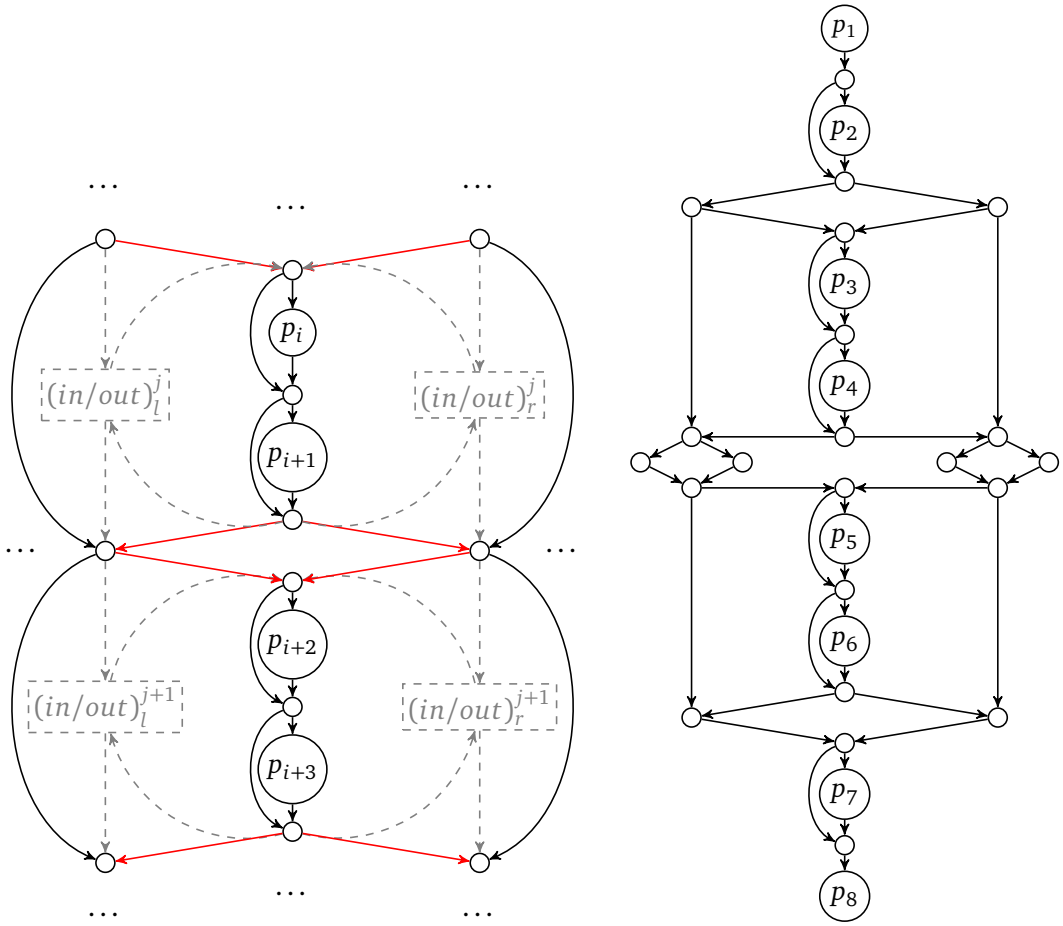
if $V = \emptyset$ **:**
  return $(\emptyset, \emptyset, ...)$
foreach $(u, v) \in E$ **:**
  if *$u \in s$ and v is recursion point for some Superpole $s \in G^*$* **:**
    $G^x \leftarrow$ *the EUG in which v is a pole*
    $E \leftarrow E \setminus \{(u, v)\}$
    $w \leftarrow \Gamma^-_{G^x}(v)$
    $E \leftarrow E \setminus \{(v, w)\}$
    $E \leftarrow E \cup \{(u, w)\}$
  else if *u is recursion point for some Superpole $s \in G^*$ and $v \in s$* **:**
    $G^x \leftarrow$ *the EUG in which u is a pole*
    $E \leftarrow E \setminus \{(u, v)\}$
    $w \leftarrow \Gamma^+_{G^x}(u)$
    $E \leftarrow E \setminus \{(w, u)\}$
    $E \leftarrow E \cup \{(w, v)\}$
  remove all recursion points from G
  for $i \leftarrow 1$ **to** k **:**
    $G^i \leftarrow$ ShortCircuit(G^i)
  return G

Theorem 4 (cf. [LYZ+21, Theorem 4]). *$\mathrm{Liu}_k(n)$ is an EUG for $\Gamma_1(n)$ with size bounded by*

$$\frac{|SP_k| - k}{k \log_2(k)} n \log_2(n) + \mathcal{O}(n).$$

Proof (cf. [LYZ+21, Theorem 4]). Let $G = (V, E, P, G^*, G^1, ..., G^k)$ be a weak Liu EUG for $\Gamma_1(n)$. Let $G' = (P', E')$ be the order preserving graph to be embedded and $\psi : E' \to \mathcal{P}_G$ the corresponding edge-embedding of G' into G by Theorem 3. We need to show that we can short-circuit each recursion point. So consider the u-v-path $\psi(e')$ in G for $e' \in E'$.

Assume that along the path a recursion point r for some Superpole $s^i \in G^*$ is used. Then, this recursion point is a pole in a sub EUG G^x for some $x \in [k]$. Because $\psi(e')$ can not start or end in a recursion point, there are edges (q, r) and (r, t) on the path for $q, t \in V$. Note that by construction, r has exactly two ingoing eges (q_1, r) with $q_1 \in s^i$ and (q_2, r) with $q_2 \in G^x$. Analogously, r has exactly two outgoing edges (r, t_1) with $t_1 \in s^i$ and (r, t_2) with $t_2 \in G^x$. We need to show that either $q = q_1 \wedge t = t_2$ or $q = q_1 \wedge t = t_2$ because these are the options left after short-circuiting G.



(a) Short-circuiting Liu et al.'s (weak) construction with 2-Way split. Edges in red are added while edges and nodes in gray are removed.

(b) Liu et al.'s 2-Way-Split construction for 8 poles without first and last node of each EUG and sub EUG .

Figure 3.5: Short circuiting Liu et.al.'s weak construction.

Case 1: r is used as an output recursion point, i.e., $q = q_1$
Then, $t = t_2$ because $t = t_1$ would imply that we use an recursion point as an intermediary node inside the Superpole s^i. This would contradict our assumption on the Superpole edge-embedding.

Case 2: r is used as an input recursion point, i.e., $q = q_2$
Then, $t = t_1$ because $t = t_2$ would imply that there exists a path in the corresponding edge-embedding ψ^{G^x} that uses a pole as an intermediary node because r is a pole in G^x. Again, this would contradict our assumption on the Superpole edge-embedding.

Thus, we can replace the edges (q, r) and (r, t) in $\psi(e')$ by (r, t). Note that this preserves the edge-disjointness of the paths. Doing this for every edge $e' \in E'$, we end up with a correct edge-embedding for ShortCircuit(G).

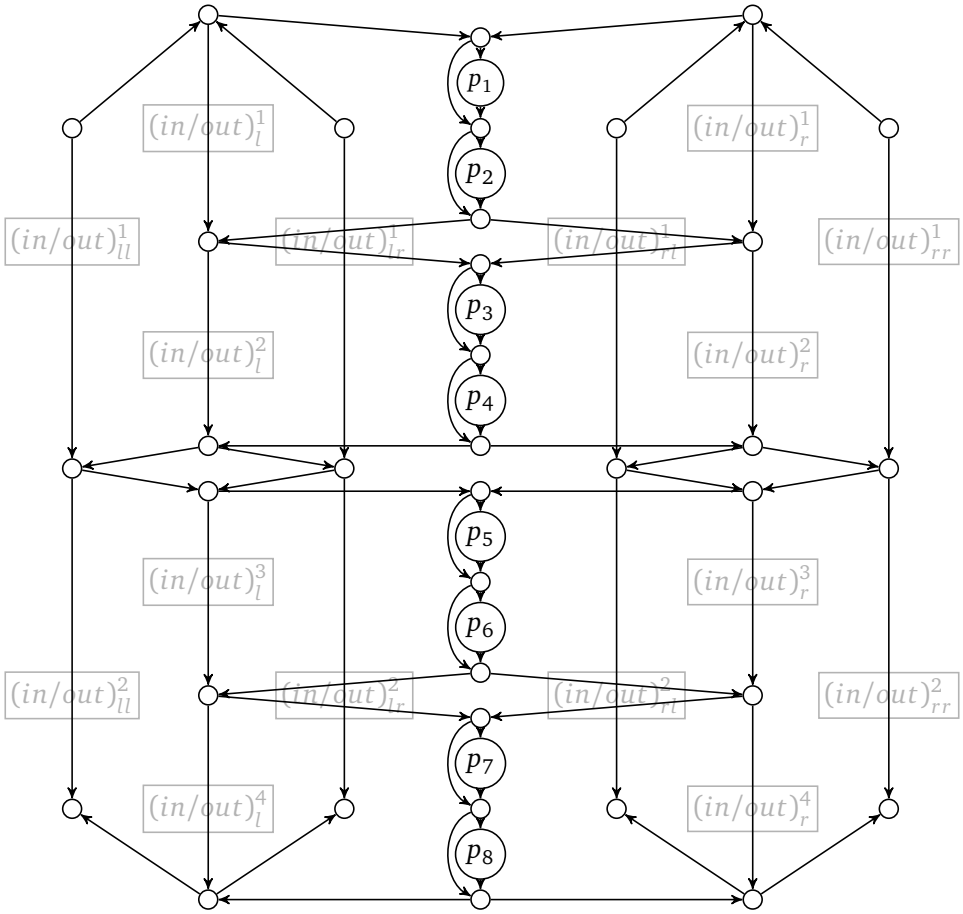


Figure 3.6: Liu's 2-Way-Split construction for 8 poles. The recursion points in gray are left for an easier comparison with the weak version. Note that the first and last node of each EUG and sub EUG are not needed.

By Proposition 3, it holds $|G| \leq \frac{|SP_k|}{k\log_2(k)} n\log_2(n) + \mathcal{O}(n)$. The number of Superpoles in each iteration i was given by $\bar{n}_i$ in Lemma 3. Thus, the total number of Superpoles in G is bounded from below by

$$\sum_{i=0}^{x-1} k^i \bar{n}_i + k^x = \sum_{i=0}^{x-1} k^i \lceil \frac{n}{k^{i+1}} \rceil + k^x \geq \sum_{i=0}^{x-1} k^i \frac{n}{k^{i+1}} = \sum_{i=0}^{x-1} \frac{n}{k} = x\frac{n}{k} \tag{3.4}$$

with $x = \lceil \log_k(n) - 1 \rceil$. Thus, (3.4) becomes

$$\lceil \log_k(n) - 1 \rceil \frac{n}{k} \geq \frac{n\log_2(n)}{k\log_2(k)} - 2\frac{n}{k} = \frac{n\log_2(n)}{k\log_2(k)} - \mathcal{O}(n).$$

Since we additionally remove k recursion points per Superpole, we get

$$|\textsc{ShortCircuit}(G)| = |G| - k\frac{n\log_2(n)}{k\log_2(k)} + \mathcal{O}(n) \leq \frac{|SP_k| - k}{k\log_2(k)} n\log_2(n) + \mathcal{O}(n).$$

□

In particular, if $k = 2$, we get a bound of $1.5n\log_2(n) + \mathcal{O}(n)$ for a $\Gamma_1(n)$ Liu EUG and $3n\log_2(n) + \mathcal{O}(n)$ for the merged $\Gamma_2(n)$ Liu EUG .

4 Support for Multi-Input Gates

In this chapter, we focus on extending the already seen constructions from Chapter 3 to support multi-input gates. The first approach is based on the idea of [Val76][SS08, §4.3] and described in Section 4.1. Our new proposed dynamic construction is presented in Section 4.2. In Section 4.3, we describe a generalized Universal Gate construction for arbitrary high input gate sizes with fanin 2. Note that this Universal Gate construction is used in both approaches for multi-input gates.

4.1 The Fixed Edge-Universal Graph Construction of [Val76][SS08, §4.3]

Assume we want to edge-embed a graph with fanin ρ. The idea of this construction is to merge ρ instances of Γ_1 EUGs into a Γ_ρ EUG. This idea was originally mentioned in [Val76] and practically deployed in [SS08, §4.3] along with two other multi-input gate UC constructions. The advantage of this construction is that each Universal Gate supports ρ inputs and ρ outgoing wires, which is why we call this the "fixed construction". It also has asymptotic optimal size and leaks less information than our new construction (cf. Section 4.2). We use the state-of-the-art construction of [LYZ+21] as our underlying EUG.

Corollary 4. *An EUG for $\Gamma_\rho(n)$ for $\rho \in \mathbb{N}_{\geq 2}$ can be constructed with size at most*

$$1.5\rho n \log_2(n) + \rho\mathcal{O}(n).$$

Proof. Construct ρ instances of $\text{LIU}(n)_2$ and merge them. By Corollary 3, this yields an EUG for $\Gamma_\rho(n)$ with size bounded by $1.5\rho n \log_2(n) + \rho\mathcal{O}(n)$. □

Note that the size of this construction is directly dependent on the fanin ρ. Our following construction is only dependent on the number of gate inputs inputs exceeding the limit of 2 inputs per gate. But, Section 4.2 shows that this comes with a trade-off in privacy.

4.2 Our Dynamic Edge-Universal Graph Construction

Because we want to use dynamic multi-input gates, we need to modify our definition of the graphs that can be edge-embedded. In particular, we have to define the maximum number of inputs for each gate before constructing the EUG (resp. the UC). Note that the concrete construction depends on these parameters together with the number of inputs, gates, and outputs. Therefore, the "anonymity" of the function is only guaranteed within this set of functions, i.e., all functions respecting above limitations can be embedded in the same UC (with different programming bits). Thus, they can not be distinguished when using a secure Two-Party Computation protocol. More formally, let f be a Boolean function and G' the graph that results from converting this Boolean function into a Boolean circuit. Sort G' topologically, convert it to a circuit with fanout 2 by using copygates, and let P^+ be the vector indicating how many inputs each gate (in topological order) can use. Then, our construction guarantees that f can not be distinguished from functions with the same number of inputs, number of gates, number of outputs, and same vector P^+ for the gate input sizes (cf. Definition 4.2.1). Note that the fixed construction [Val76][SS08, §4.3] guarantees this with the restriction of $\mathrm{P}^+ = \mathbb{1}\rho$, where $\mathbb{1}$ denotes the vector where each entry is 1, and ρ denotes the fanin of the circuit, i.e., only the number of inputs, gates, outputs, and the maximum used gate input size is leaked. This means that the fixed construction always leaks less information, unless $\mathrm{P}^+ = \mathbb{1}\rho$. Therefore, our construction can also be seen as a Semi-Private Function Evaluation scheme. This is sufficient in many applications as [PSS09; GKSS19] show, but keep in mind that there can be situations where this is not secure. The amount of leakage can be reduced to some extent by using a maximum gate input size higher than needed. However, this increases the size of the resulting UC.

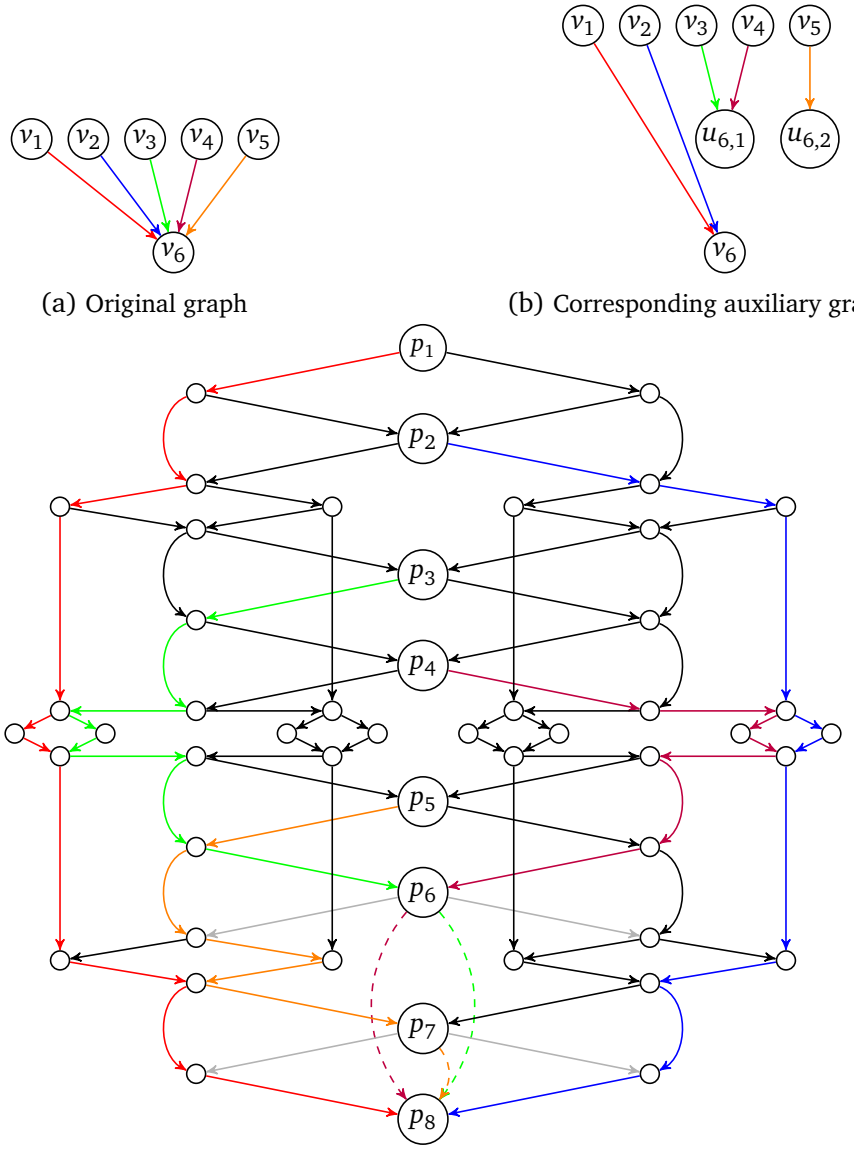
To formally capture the set of graphs that our new construction can edge-embed, we need the following definition.

Definition 4.2.1 ($\Gamma_{\mathrm{P}^+,\mathrm{P}^-}(n)$)**.** *Let $G = (V, E)$ be a directed acyclic graph with topologically ordered $V := \{v_1, v_2, ..., v_n\}$ and $\mathrm{P}^+, \mathrm{P}^- \in \mathbb{N}^n$. Then $G \in \Gamma_{\mathrm{P}^+,\mathrm{P}^-}(n)$ if:*

- $|V| \leq n$,
- $\delta^+(v_i) \leq \mathrm{P}_i^+ \wedge \delta^-(v_i) \leq \mathrm{P}_i^- \ \forall\ i \in [n]$.

If $\mathrm{P}^{+/-} = \mathbb{1}\rho$ for some $\rho \in \mathbb{N}$, we write ρ instead of $\mathbb{1}\rho$.

In this sense, Corollary 4 yields a $\Gamma_{\rho,\rho}(n)$ EUG. In the following, we describe our dynamic multi-input gate construction. An example of the whole EUG creation and embedding process is depicted in Figure 4.1 on page 37. The explicit creation of the used auxiliary graph is given by Algorithm 8 on page 38.



(a) Original graph

(b) Corresponding auxiliary graph

(c) Edge-embedding of the original graph. First, the edges from the auxiliary graph are embedded. Then, edges in gray are removed from the EUG, while dashed edges are added to the EUG, resp. to the edge-embedding. The result is an edge-embedding for the original graph. Then, we can replace the ingoing edges to p_6 by directed edges to the multi-input pole p_8. The auxiliary pole p_7 becomes a Y-Switch that only forwards the orange wire.

Figure 4.1: Example of our dynamic multi-input gate construction.

Algorithm 8: AUXILIARYGRAPH(G)

Input : $G = (V, E) \in \Gamma_{\mathrm{P}^+,2}(n)$

Output : $\bar{G} = (\bar{V}, \bar{E}) \in \Gamma_2(n + \Delta)$ with $\Delta = \sum_{i=0}^{n} \max\{\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil, 0\}$

```
1  Ḡ = (V̄, Ē) ← (V, ∅)
2  foreach v_i ∈ V :
3      j ← 0
4      foreach e = (w, v_i) ∈ E :
5          if j ≥ 2 :
6              if j ≡ 0 (mod 2) :
7                  V̄ ← V̄ ∪ {u_{i,j/2}}
8              Ē ← Ē ∪ {(w, u_{i,⌈j/2⌉})}
9          else
10             Ē ← Ē ∪ {e}
11         j ← j + 1
```

The idea of our construction is the following:
We begin by constructing an auxiliary graph $\bar{G}$. For each pole that has more than two incoming edges, we create an auxiliary pole for each two additional inputs. Then, we replace all edges to the original pole, up to the first 2 edges, by edges to the auxiliary poles. The purpose of the auxiliary poles is to forward their inputs to the original multi-input pole. The resulting EUG $\mathcal{U}$ then guarantees that there can be a path from any pole with lower order to the corresponding auxiliary poles. Note that because we merge 2 Γ_1 EUGs, each auxiliary pole always has two inputs. If there is a multi-input gate with an odd number of inputs, there will be one auxiliary pole in $\bar{G}$ with only one input. Nevertheless, the corresponding auxiliary pole in $\mathcal{U}$ will have two inputs, so we can not directly forward the inputs of this pole.

One option is to always forward both inputs to the multi-input gate, but this means that the multi-input pole has one input more than it actually needs. This can be solved by padding the function bits such that this input is ignored, but then we need a bigger universal gate. Since the number of inputs to each pole is public anyway and the size of the Universal Gates grows exponentially with its inputs (cf. Proposition 4), we circumvent this problem by a small trick.

If the auxiliary pole has 2 inputs in $\bar{G}$, we can just replace all edges to the auxiliary pole by direct edges to the corresponding multi-input pole in the EUG and remove the auxiliary pole. If the auxiliary pole has only one input in $\bar{G}$, we replace the auxiliary pole by a Y-Switch and set its control bit such that the desired input is forwarded.

Theorem 5. *Let* $\mathrm{P}^+ \in \mathbb{N}^n$. *Then, there exists an EUG for* $\Gamma_{\mathrm{P}^+,2}(n)$ *with size bounded by*

$$3(n + \Delta)\log_2(n + \Delta) + \mathcal{O}(n + \Delta),$$

where $\Delta := \sum_{i=0}^{n} \max\{\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil, 0\}$.

Proof. Let $G = (V, E) \in \Gamma_{\mathrm{P}^+,2}(n)$ be the graph to be embedded in an EUG. W.l.o.g. assume that $V := \{v_1, v_2, ..., v_n\}$ is in topological order with possibly dummy nodes if $|V| < n$.

Step 1: Construct a $\Gamma_2(n + \Delta)$ graph using auxiliary poles for nodes with indegree higher than 2:
Set $\bar{G} = (\bar{V}, \bar{E}) = \text{AUXILIARYGRAPH}(G) \in \Gamma_2(n+\Delta)$ (Algorithm 8). Note that the "relative topological order" is maintained, i.e., $\eta_{\bar{G}}(v_i) < \eta_{\bar{G}}(v_{i+1})\ \forall i \in [n]$. Furthermore, we can choose $\eta_{\bar{G}}$ such that the topological order of $\bar{V}$ has the form $(..., v_i, u_{i+1,1}, ..., u_{i+1,\lceil \frac{\deg^+(v_{i+1})-2}{2} \rceil}, v_{i+1}, ...)$ for all $i \in [n]$, i.e., the auxiliary poles $u_{i,j}$ for $j \in [\lceil \frac{\deg_G^+(v_i)-2}{2} \rceil]$ are directly before the original pole v_i if auxiliary poles are needed.

Step 2: Create a $\Gamma_2(n + \Delta)$ EUG $\mathcal{U} = (V^{\mathcal{U}}, E^{\mathcal{U}}, P, \mathcal{U}^*, \mathcal{U}_1, \mathcal{U}_2)$ with pole set $P = \bar{V}$ and edge-embed $\bar{G}$ into it:
We do this by creating a Liu EUG $\mathcal{U}$ with pole set $\bar{V}$ and split parameter 2. Then, we merge two instances of it. By Theorem 3 and Corollary 3, this yields a $\Gamma_2(n + \Delta)$ EUG of size at most $3(n + \Delta)\log_2(n + \Delta) + \mathcal{O}(n + \Delta)$. Edge-embedding $\bar{G}$ into $\mathcal{U}$ (by Theorem 4) yields $\psi : \bar{E} \to \mathcal{P}_{\mathcal{U}}$.

Step 3: Adjust $\mathcal{U}$ to get the final EUG $\bar{\mathcal{U}} = (V^{\bar{\mathcal{U}}}, E^{\bar{\mathcal{U}}}, V, \bar{\mathcal{U}}^*, \bar{\mathcal{U}}_1, \bar{\mathcal{U}}_2)$:
Iterate through all auxiliary poles of $\mathcal{U}$ and do the following: Let $u_{i,j}$ be an auxiliary pole for v_i for $i \in [n], j \in [\lceil \frac{\deg_G^+(v_i)-2}{2} \rceil]$. If the auxiliary pole forwards two inputs (in $\bar{G}$), remove all outgoing edges from the auxiliary pole and replace each of them with an edge connecting the auxiliary pole to the original multi-input pole, i.e., remove each $(u_{i,j}, w) \in E^{\mathcal{U}}$ for $w \in V^{\mathcal{U}}$ and replace it by $(u_{i,j}, v_i)$. This yields two edges $(u_{i,j}, v_i)$ per auxiliary pole $u_{i,j}$. Thus, $E^{\mathcal{U}}$ becomes a multi set.

If the auxiliary pole only forwards one input (in $\bar{G}$), remove all outgoing edges, and only add one edge connecting the auxiliary pole to the original multi-input pole, i.e., add the edge $(u_{i,j}, v_i)$ to $\mathcal{U}$. The graph that results from modifying $\mathcal{U}$ in the just described way is denoted by $\bar{\mathcal{U}}$.

To show that $\bar{\mathcal{U}}$ is a $\Gamma_{\mathrm{P}^+,2}(n)$ EUG, we need to define an edge-embedding $\bar{\psi}$ from G into $\bar{\mathcal{U}}$:
Note that for edges $e = (v_i, v_l) \in G \setminus \bar{G}$, i.e., edges whose endpoints are not auxiliary poles, ψ already yields edge-disjoint v_i-v_l-paths, and we can set $\bar{\psi}(e) = \psi(e)$ for those edges.

Now consider edges edges $e = (v_i, v_l) \in G \cap \bar{G}$, i.e., the endpoints of those edges are transformed into an auxiliary pole in $\bar{G}$. For each e, there is exactly one $\bar{e} = (v_i, u_{l,j}) \in \bar{G}$ for $j \in [\lceil \frac{\deg_G^+(v_l)-2}{2} \rceil]$ (line 8 in AUXILIARYGRAPH). Now set $\bar{\psi}(e) = \psi(\bar{e}) + (u_{l,j}, v_l)$ for one of the possibly two edges $(u_{l,j}, v_l)$ that were added to $\bar{\mathcal{U}}$ before. Obviously, this yields a v_i-v_l-path. Since there are at most two edges connecting to an auxiliary pole in $\bar{G}$, we can choose a unique last edge for each path. Because the paths in the image of ψ were already edge-disjoint, also the paths in the image of $\bar{\psi}$ are edge-disjoint. Thus, $\bar{\psi}$ is an edge-embedding of G into $\bar{\mathcal{U}}$. □

Once the EUG is created, we can remove, resp. replace, the auxiliary poles as described before because they are not needed anymore in the UC.

4.3 Transforming a Multi-Input Edge-Universal Graph into a Universal Circuit

Now that we can edge embed $\Gamma_{P^+,P^-}(n)$ graphs into our EUG, we have to deal with the translation of the EUG and the edge-embedding into a UC. One problem that arises with all constructions is that, by default, secure Two-Party Computation protocols like Yao's garbled circuits or Goldreich-Micali-Wigderson [Yao86; GMW87] only support gates with two inputs. Hence, we can not directly convert our poles with multi-input gates into a suitable circuit. Furthermore, using only 2-input AND gates preserves the ability to use the resulting UC in all standard garbled circuits STPC schemes like the state-of-the-art garbling scheme of [RR21] without further modification. The scheme of [RR21] has a communication complexity of $1.5\kappa+5$ bits per AND gate, where κ denotes the security parameter (usually $\kappa = 128$). It also uses the Free XOR technique [KS08b]. Therefore, XOR gates can be evaluated without any communication effort. Note that Yao's garbled circuits can be extended to support multi-input gates like in [MNPS04], which was already used in the multi-input gate UC construction of [SS08]. This approach would further reduce the needed communication, but there are only exactly n Universal Gates and $\mathcal{O}(n \log n)$ X- and Y-Switches such that this improvement would be marginal. Thus, we decided to use only 2-input gates which are supported by all STPC protocols.
Note that all other nodes, i.e. the X- and Y-Switches, are not affected by our construction and have the same number of inputs and outputs as in the original construction. We use a generalized Universal Gate construction supporting arbitrary many inputs. Note that any k-input Boolean gate can be described by a lookup table with 2^k entries. Hence, we can use a 2^k-to-1 multiplexer for each Universal Gate with k inputs to simulate an k-input Boolean gate. Then, the multiplexer selects the right lookup table entry depending on the input bits.

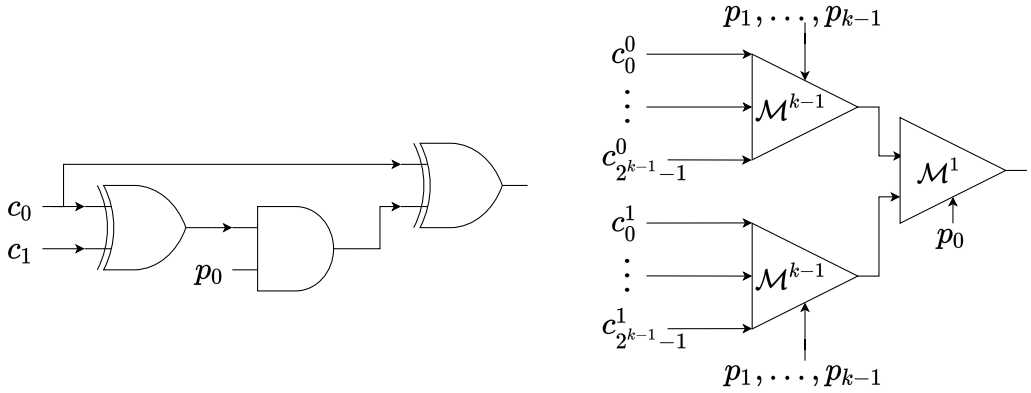
Definition 4.3.1 (2^k-to-1 multiplexer)**.** *Let $k \in \mathbb{N}$. A 2^k-to-1 multiplexer $\mathcal{M}$ is a Boolean circuit with 2^k function table bit inputs $c = (c_0, ..., c_{2^k-1}) \in \{0,1\}^{2^k}$ and k programming bit inputs $p = (p_0, ..., p_{k-1}) \in \{0,1\}^k$ such that*

$$\mathcal{M}(c,p) = c_{[p]} \;\forall\; c \in \{0,1\}^{2^k}, p \in \{0,1\}^k,$$

where $[p]$ denotes the decimal number of the binary number $p_0...p_{k-1}$, i.e.,

$$[p] = \sum_{i=0}^{k-1} p_i 2^{k-1-i}.$$

To build a 2^k-to-1 multiplexer $\mathcal{M}^k$, we use a tree of $2^k - 1$ 2-to-1 multiplexers. This is done recursively by using the output of two 2^{k-1}-to-1 multiplexers as the input to a 2-to-1 multiplexer. The construction is depicted in Figure 4.2.



(a) 2-to-1 multiplexer $\mathcal{M}^1$ (b) 2^k-to-1 multiplexer $\mathcal{M}^k$

Figure 4.2: The 2^k-to-1 multiplexer construction. The control bits c^0 and c^1 are the lookup table entries for inputs starting with 0, resp. with 1 (cf. proof of Proposition 4).

Proposition 4. *A 2^k-to-1 multiplexer can be realized by a Boolean circuit with $2^k - 1$ AND gates and $2(2^k - 1)$ XOR gates.*

Proof. Let $c = (c_0, ..., c_{2^k-1}) \in \{0,1\}^{2^k}$ be the function table bits and $p = (p_0, ..., p_{k-1}) \in \{0,1\}^k$ the programming bits for $k \in \mathbb{N}$. We use the construction of Figure 4.2 and prove its correctness by induction over k.

Induction base: $k = 1$
If $p_0 = 0$, then $\mathcal{M}^1(c_o, c_1, p_0) = ((c_0 \oplus c_1) \wedge 0) \oplus c_0 = 0 \oplus c_0 = c_0$. If $p_0 = 1$, then $\mathcal{M}^1(c_o, c_1, p_0) = ((c_0 \oplus c_1) \wedge 1) \oplus c_0 = c_0 \oplus c_1 \oplus c_0 = c_1$. Thus, $\mathcal{M}^1$ is a 2-to-1 multiplexer with one AND gate and two XOR gates (cf. Figure 4.2).

Induction step: $k - 1 \rightsquigarrow k$
Let $\mathcal{M}^{k-1}$ be the 2^{k-1}-to-1 multiplexer given by the induction hypothesis. Let $c^0 \in \{0,1\}^{2^{k-1}}$ be the function table bits of c, where the first input is 0, i.e., all the bits $c_{[0,x_1,x_2,...,x_{k-1}]}$ for $x_1, x_2, ..., x_{k-1} \in \{0,1\}$. Analogously, let $c^1 \in \{0,1\}^{2^{k-1}}$ be the function table bits, where the first input is 1. Then

$$\mathcal{M}^k(c,p) := \mathcal{M}^1(\mathcal{M}^{k-1}(c^0, p_1, ..., p_{k-1}), \mathcal{M}^{k-1}(c^1, p_1, ..., p_{k-1}), p_0)$$

is a 2^k-to-1 multiplexer: By induction hypothesis on $\mathcal{M}^{k-1}$, the two 2^{k-1}-to-1 multiplexers return $c_{[0,p_1,...,p_{k-1}]}$ and $c_{[1,p_1,...,p_{k-1}]}$. Since $\mathcal{M}^1$ is a 2-to-1 multiplexer, it outputs $c_{[p_0,p_1,...,p_{k-1}]}$. Therefore, $\mathcal{M}^k$ is a 2^k-to-1 multiplexer.

The circuit size of $\mathcal{M}^k$ is given by two times the circuit size for $\mathcal{M}^{k-1}$, which is each $2^{k-1}-1$ AND and $2(2^{k-1}-1)$ XOR gates plus the one AND gate and two XOR gates from $\mathcal{M}^1$. This results in $2(2^{k-1}-1)+1 = 2^k-1$ AND gates and $2 \cdot 2(2^{k-1}-1) = 2(2^k-1)$ XOR gates in total. □

5 Implementation & Experimental Evaluation

In this chapter, we describe our implementation (Section 5.1), the test setup, and the used circuits (Section 5.2). Then, we benchmark our multi-input gate UC construction (Section 5.3) and propose an improved hybrid construction along with benchmarks in Section 5.4.

We will benchmark our dynamic multi-input gates UC construction (cf. Section 4.2) and compare it with the usual approach of using circuits with at most binary gates and with the fixed multi-gate input approach (Section 4.1). All results in this chapter use the EUG construction of Liu et al. [LYZ+21] to construct the underlying Γ_1 EUGs. For the sake of completeness, there is also a smaller set of benchmarks that use Valiant's 2-Way construction [Val76] as the underlying EUG in Appendix A.1. Because these results are analogous to the following results with Liu et al.'s construction, only with larger size, we refrain from presenting them in the main part of this work. Compilation times of the UCs and their programmings can also be found in Appendix A.2.

5.1 About The Implementation

Since the 2-Way construction (split parameter $k = 2$) is the most efficient in the construction of Liu et. al [LYZ+21], we only implemented this variant of the EUGs. Our implementation supports arbitrary high gate input sizes, but keep in mind that the size of the Universal Gates grows exponentially in its number of inputs (cf. Proposition 4).

Our implementation is using C++ (version C++17) and is compiled with gcc version 9.3.0 (Gnu Compiler Collection) under Linux Mint 20.1 running kernel 5.11.0-051100-generic.

Let C denote the circuit to be embedded and ρ the maximum fanin of the circuit. The UC compilation works in the following way:

1. Parse the circuit:
The circuit is input in Secure Hardware Definition Language (SHDL) [MNPS04] or Bench format [1], possibly converted from Bench to SHDL, and parsed into the internal graph representation. If the fanout of the graph is higher than the allowed fanout (ρ for the fixed construction, 2 for the dynamic construction), the fanout is reduced by copy gates. The resulting graph is denoted by G. If we want to use dynamic multi-input gates, the auxiliary graph as in Theorem 5 is generated (cf. Algorithm 8). In this case, we will denote the auxiliary

[1] The Bench format was used as a standard benchmark circuit description for ISCAS'85, ISCAS'89, and ISCAS'99.

graph by G and the former graph with possibly reduced fanout by $\bar{G}$. This gives us a mapping of poles of $\bar{G}$ to poles of auxiliary graph G.

2. Split G into multiple Γ_1 graphs:
This is done by coloring the edges (cf. Corollary 2). If, we use the dynamic construction, we get 2 Γ_1 graphs. If we use the fixed construction, this yields ρ Γ_1 graphs.

3. Create an Γ_1 EUG for each Γ_1 graph:
For each Γ_1 graph from the previous step, we create a Γ_1 EUG. Possible EUGs are Valiant's EUG [Val76] and the EUG of Liu et al. [LYZ+21] with 2-Way split.

4. Edge-embed the Γ_1 graphs and merge them:
Each Γ_1 graph is edge-embedded into the corresponding Γ_1 EUG. This edge-embedding is coded directly into the control bits of the X- and Y-Switching nodes of the EUG. We do not construct an explicit edge-embedding map ψ because we only need the control bits to create the UC and the programming bits. The concrete algorithm (Algorithm 3) uses a slightly modified version of the edge-embedding algorithm in [GKS17] to also support the construction of Liu et al. Then, the Γ_1 EUGs are merged into a Γ_ρ EUG (for the fixed construction) or into a Γ_2 EUG (for the dynamic construction).

5. Do basic optimizations and check the correctness of the edge-embedding:
We remove edges connecting into an input pole since they will never be used and replace nodes just forwarding by wires. Then, we possibly remove isolated nodes or change X- to Y-Switching nodes if one edge was removed before. Switches that are not set yet are randomly set. At the end, we check the correctness of the edge-embedding by checking for each edge (u, v) in G whether there is a path leading from u to v (by backtracking according to the control bits).

6. Set the gates of the EUG:
If we use the dynamic construction, we replace the auxiliary poles by wires connecting directly to the actual pole or by a Y-Switching node if only one input is forwarded. Analogously to step 5, we check the correctness of the edge-embedding now with respect to $\bar{G}$. For each node in G, we set the function bits of the corresponding EUG pole. Since the function bits depend on the order of inputs, and there is no guarantee about the order, we determine the order of inputs and set the function bits accordingly. This also involves padding the function bits if the gate has more inputs than needed (e.g., in the fixed case, each gate supports the maximum fanin of the circuit). Note that these additional inputs are likely to occur since each Universal Gate outputs ρ (fixed construction) or 2 (dynamic construction) wires, independent of whether they are used in G or not. The function bits are padded such that the additional and undesired inputs are ignored.

7. Transform the EUG into a UC:
We topologically sort the EUG with the restriction that the circuit input occur before all other nodes. This allows us to enumerate all nodes and assign numbers in ascending order to the gate output wires. Each node (except for X-Switches) only has one output value, so all outgoing wires of a node receive the same number. The X-Switches get two wire numbers,

one for the left and one for the right output. Then, each node along with its ingoing and outgoing wires is written into a circuit file, while the programming bits are written into a separate programming bits file.

8. Check the correctness of the UC:
The correctness of the UC is checked by simulating the evaluation of the SHDL circuit and the UC files for random inputs. This can be repeated arbitrarily often (default is 1) or skipped. The check fails, if the outputs of both evaluations differ, otherwise the check passes.

5.2 Test Setup and Methodology

The used circuits can be found in Table 5.1. Their basic properties and sources can be found in Table 5.2. Originally, all these circuits have only gates with at most two inputs. To create equivalent circuits with higher gate input sizes, we used the circuit synthesis system abc [1] that allows Lookup Table (LUT)- or Field Programmable Gate Array (FPGA)-Mapping, i.e., it converts sets of binary gates to LUTs/multi-input gates. We used the included FPGA-Mapper by [CMCB07] using Priority Cuts, which can be used by the command "if -K <maximum LUT size>". Additionally, we used the option "-a" for area-oriented mapping, which aims to minimize the number of nodes and not the depth, and topologically sorted the resulting circuit. We did not use any special circuit optimization for STPC purposes, but above option yielded the best results in our cases. To remain consistent, we also used the FPGA-Mapper for a maximum LUT size of 2.

Category	Name	Description
Arithmetic	rca256	Ripple-Carry Adder with two 256-bit inputs.
	rcmul64	Ripple-Carry Multiplier with two 64-bit inputs.
	karatsuba64	Karatsuba multiplier with two 64-bit inputs.
	rcsqr64	Ripple-Carry Squarer with one 64-bit input.
Distance	md256	Manhattan distance between two two-dimensional 256-bit points.
	ed64	Euclidean distance between two two-dimensional 64-bit points.
Crypto	AES	128-bit AES encryption algorithm (with expanded input key).
	DES	64-bit DES encryption algorithm (with expanded input key).
	MD5	Compression function of the MD5 hash algorithm with 512-bit input.
	SHA1	Compression function of the SHA-1 hash algorithm with 512-bit input.

Table 5.1: The circuits used in our benchmark.

[1] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 1.01. http://www.eecs.berkeley.edu/~alanmi/abc/

Category	Name	Inputs	Gates	Outputs	Source
Arithmetic	rca256	512	1532	257	[DKS+17]
	rcmul64	128	28032	128	[DKS+17]
	karatsuba64	128	19813	128	[DKS+17]
	rcsqr64	64	13981	128	[DKS+17]
Distance	md256	1024	5627	257	[DKS+17]
	ed64	256	30633	129	[DKS+17]
Crypto	AES	1536	26085	128	[ST][1]
	DES	832	18589	64	[ST][1]
	MD5	512	14570	128	[ST][1]
	SHA1	512	30117	160	[ST][1]

Table 5.2: Basic properties of the used circuits.

Note that the circuits we embed must have fanout bounded by 2 (in the case of the binary or dynamic approach), resp. ρ, where ρ denotes the maximum gate input size (for the fixed multi-input gates approach). The sizes and gate distributions of all LUT-mapped circuits can be found in Figures 5.2, 5.6, and 5.10. Since we now have Universal Gates of different size, we can not just count the number of nodes of the EUG to compare the implementations. Therefore, we counted the number of AND gates that are necessary to instantiate the building blocks of the UC (cf. Table 5.3). Note that due to the Free XOR technique [KS08b] for Yao's garbled circuits [Yao86], XOR gates can be evaluated without communication effort.

Building block	AND gates	XOR gates
X-Switching block [KS08b]	1	3
Y-Switching block [KS08b]	1	2
Universal Gate with $k \geq 2$ inputs (Prop. 4)	$2^k - 1$	$2^{k+1} - 2$

Table 5.3: The needed AND and XOR gates per building block in our UC constructions.

[1] These circuits are transformed into SHDL format by the included Bristol-to-SHDL converter of the ENCRYPTO open source UC compiler https://github.com/encryptogroup/UC [AGKS20]

In addition to our real life circuits, we also use Algorithm 9 to create synthetic circuits.

Algorithm 9: CREATERANDOMCIRCUIT($n_I, n_G, n_O, \rho^+, \rho^-$)

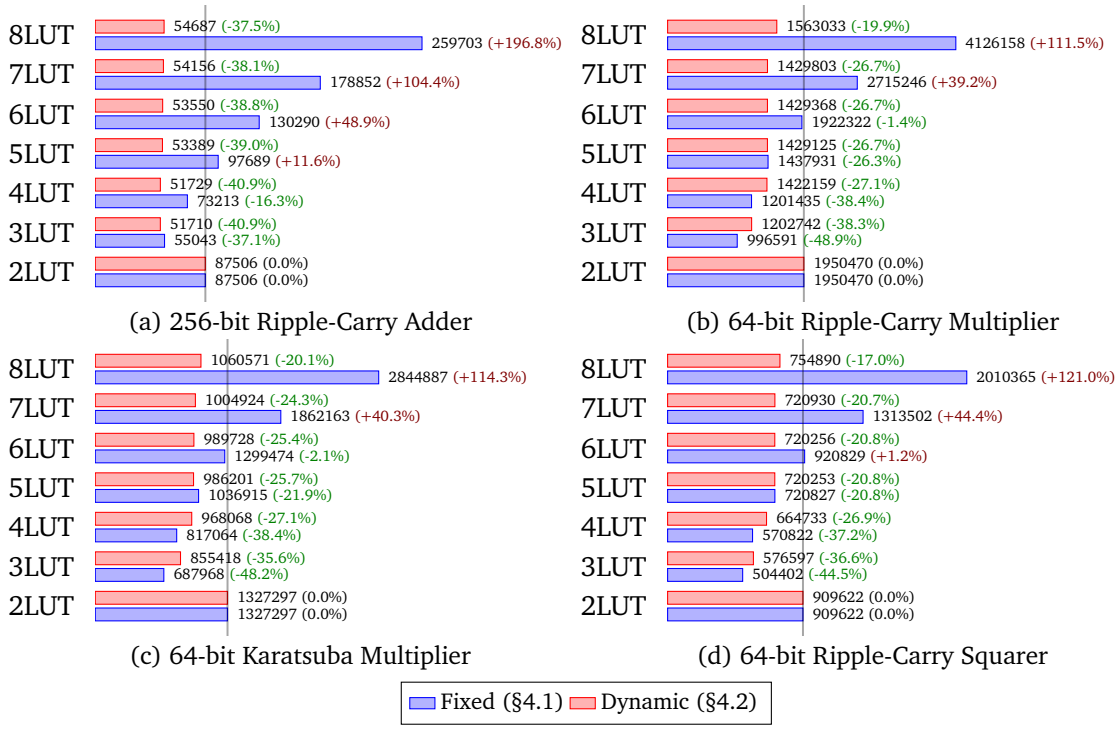
Input : Number of inputs, gates and outputs n_I, n_G, n_O, fanin ρ^+, fanout ρ^-

Output : Graph $G = (V, E)$ with n_I inputs, n_G gates, n_O outputs, fanin ρ^+ and fanout ρ^-

```
1  n ← n_I + n_G + n_O
2  create vertices v_1, ..., v_n
3  G = (V, E) ← ({v_1, ..., v_n}, ∅)
4  for i ← 1 to n_I + n_G :
5      for k ← 1 to ρ^- :
6          j ←$ {i + 1, ..., n}  // choose uniformly random
7          if δ^+_G(v_j) < ρ^+ :
8              E ← E ∪ {(v_i, v_j)}
9      if i > n_I :
10         set random function bits for v_i
11     else
12         mark v_i as input
13 for i ← n_I + n_G + 1 to n :
14     set random function bits for v_i
15     mark v_i as output
16 return G
```

The goal of these synthetic circuits is to get circuits with a high variance in gate input sizes. Thus, we do not need real random graphs in the sense of a uniformly random chosen circuit out of all valid circuits. Instead, for each node we try to connect it to ρ^- randomly chosen nodes with higher topological order. Only if those target nodes have less than ρ^+ inputs, we add this wire. This is not a uniformly random chosen circuit.

5.3 Universal Circuit Sizes



(a) 256-bit Ripple-Carry Adder

(b) 64-bit Ripple-Carry Multiplier

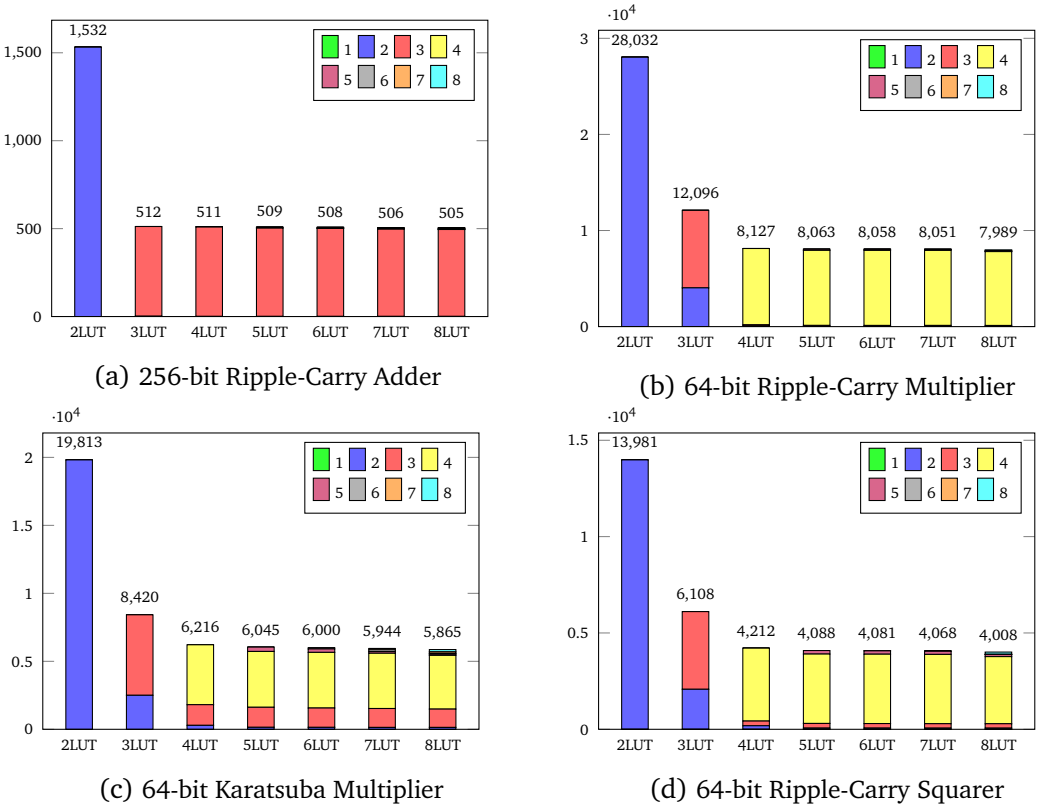
(c) 64-bit Karatsuba Multiplier

(d) 64-bit Ripple-Carry Squarer

Figure 5.1: **Arithmetic:** Comparison of the UC sizes (number of AND gates) needed to embed arithmetic operations w.r.t. different LUT sizes. The vertical lines show the results of the binary approach. The number in braces is the relative improvement over the binary construction.

The benchmarks for arithmetic operations in Figure 5.1 show that in every circuit, the size of the UC can be reduced by more than 40% by using multi-input gates. The sweet spot for the circuit fanin is between 3 and 4 for all cases and holds for the fixed as well as the dynamic approach. Especially the fixed multi-input approach is interesting in two ways. The positive observation is that, in contrast to the dynamic approach, we achieve the same level of anonymity (up to the maximum fanin of the circuit) as with the binary approach, while reducing the size of the UC by 45% on average for the arithmetic circuits (when using the optimal LUT size). Most notably, we could reduce the size of the 64-bit Ripple-Carry Multiplier by 49%. However, this only holds for small LUT sizes up to 5. Using higher LUT sizes drastically increases the number of nodes such that the resulting UC becomes larger than the UC constructed by the standard binary construction. On the other hand, the dynamic construction does not show this behavior and always improves on the usual binary construction on average by 38% when used with the best possible LUT size and by 25% over all used LUT sizes. This shows that our dynamic construction is more consistent, but we

expected the dynamic construction also to be more efficient than the fixed construction since it leaks more information.



(a) 256-bit Ripple-Carry Adder

(b) 64-bit Ripple-Carry Multiplier

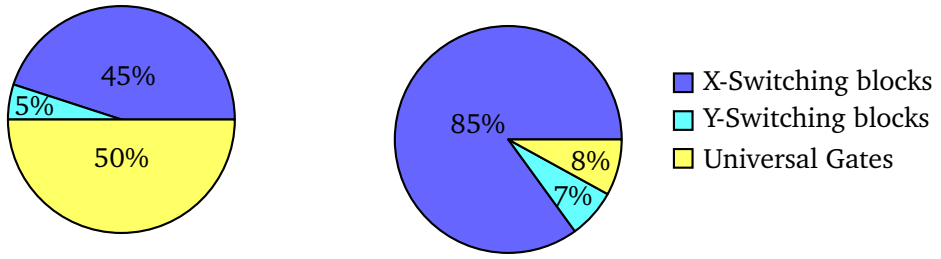
(c) 64-bit Karatsuba Multiplier

(d) 64-bit Ripple-Carry Squarer

Figure 5.2: **Arithmetic:** Distribution of the gate input sizes w.r.t. different LUT sizes. The different colors represent the used gate input sizes. Note that this figure only counts the number of gates and does not weight a multi-input gate more than a binary gate.

Figure 5.2 shows the distribution of the different LUT sizes for the arithmetic circuits. Surprisingly, there is no almost circuit size reduction by allowing more than 4 gate inputs. The possibility to use high LUT sizes was barely used by our FPGA-Mapper. Because this observation is extreme for the 256-bit Ripple-Carry Adder, we further analyzed this circuit and its corresponding UCs. There is no significant size reduction by using more than 3 gate inputs. On the other hand, because there are always some gates with the maximum LUT size, the fixed multi-input gate construction has to support this gate input size for all Universal Gates. This results in half of the UC consisting of the Universal Gates for the 8LUT case (cf. Figure 5.3 on page 50). But even without the huge share of the Universal Gates, the fixed multi input gate construction would be more than twice the size of the dynamic multi-input construction in this case due to the merging of 8 Γ_1 EUGs, where most of the wires are never used.

If the circuit size can further be reduced by using even higher LUT sizes, we suspect that the dynamic construction will outperform the fixed construction in those situations. This can be seen in the benchmarks with LUT sizes greater than 4 since in these cases the dynamic construction profits extremely from not being directly dependent on the fanin. However, the circuit synthesis suite used in our benchmark could not reduce the circuit sizes sufficiently with high LUT sizes. But, we will see such a case with the DES circuit and very high LUT sizes in Section 5.4.



(a) Fixed (§4.1) (b) Dynamic (§4.2)

Figure 5.3: **Ripple-Carry Adder:** Share of the building blocks in the UC size (number of AND gates) for the 256-bit Ripple-Carry Adder with LUT size 8.

Note that the fanout of the circuit to be embedded must be equal to the fanin of the circuit for the fixed multi-input construction, resp. 2 for the dynamic multi-input construction. Since the added copy gates are treated like usual gates by the EUG construction, this increases the resulting UC size significantly. This is more severe with the dynamic construction because we only merge two Γ_1 EUGs. Thus, each Universal Gate in the dynamic construction natively only supports two outgoing wires. The effect on the circuit size can be seen in Figure 5.4.

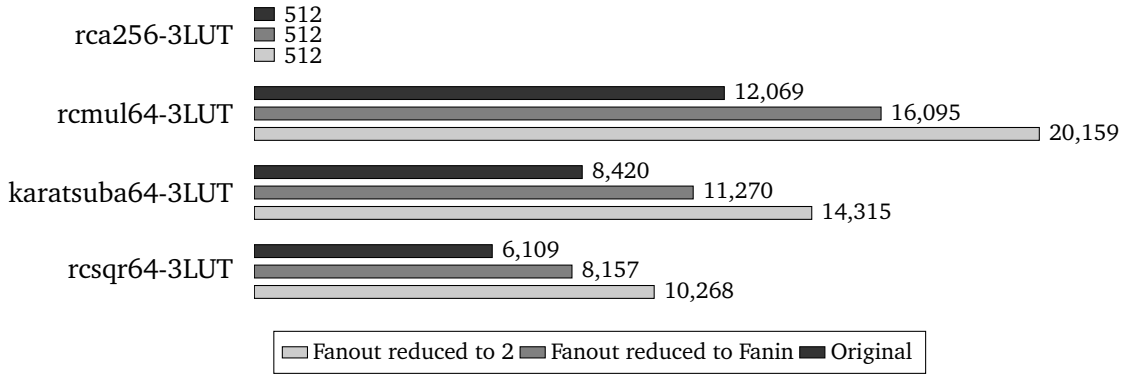
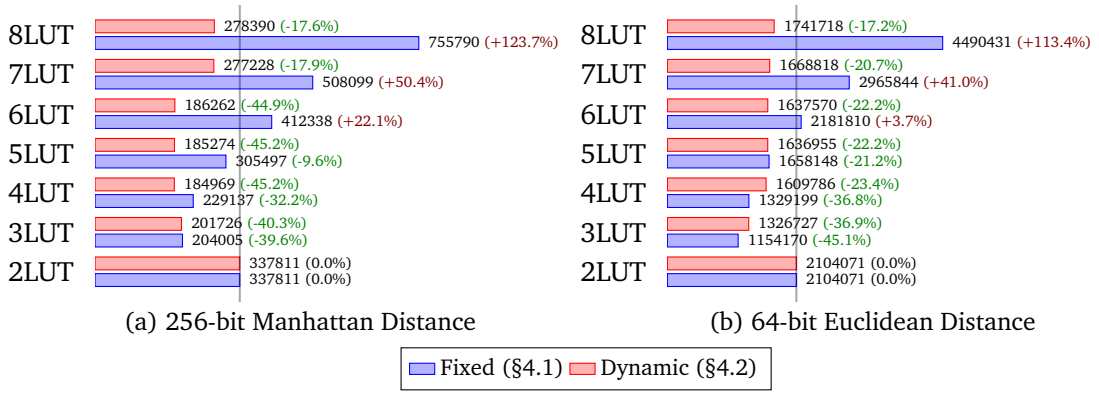


Figure 5.4: **Arithmetic:** Comparison between the original circuit sizes and the same circuits after reducing the fanout. The used versions of the circuits are the ones that yielded the smallest UC size using the fixed or dynamic approach. Note that these are the circuit sizes before replacing wires by auxiliary poles in the dynamic construction.

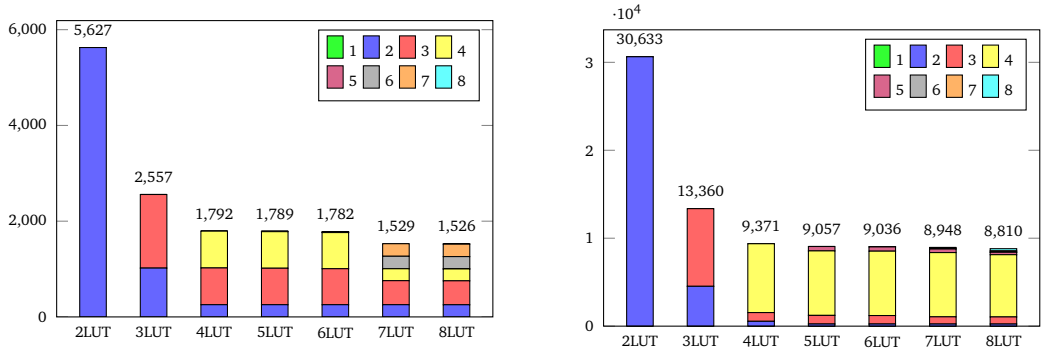


(a) 256-bit Manhattan Distance

(b) 64-bit Euclidean Distance

Figure 5.5: **Distance:** Comparison of the UC sizes (number of AND gates) needed to embed Manhattan and Euclidean Distance w.r.t. different LUT sizes. The vertical lines show the results of the binary approach. The number in braces is the relative improvement over the binary construction.

The benchmarks for the Universal Circuits computing a distance (cf. Figure 5.5) draw a similar picture as the prior ones. The best result for the Manhattan Distance is achieved by the dynamic construction with a 45% size reduction compared to the standard binary construction, while the UC for the Euclidean Distance can be reduced by 45% using the fixed multi-input gate construction.



(a) 256-bit Manhattan Distance

(b) 64-bit Euclidean Distance

Figure 5.6: **Distance:** Distribution of the gate input sizes w.r.t. different LUT sizes. The different colors represent the used gate input sizes. Note that this figure only counts the number of gates and does not weight a multi-input gate more than a binary gate.

Again, there is only a little reduction in circuit size by allowing higher LUT sizes (cf. Figure 5.6). In particular, the circuit size reduction from 6LUT to 7LUT is not sufficient for a UC size reduction, while the reduction from 3LUT to 4LUT is sufficient. Figure 5.7 shows that a solely

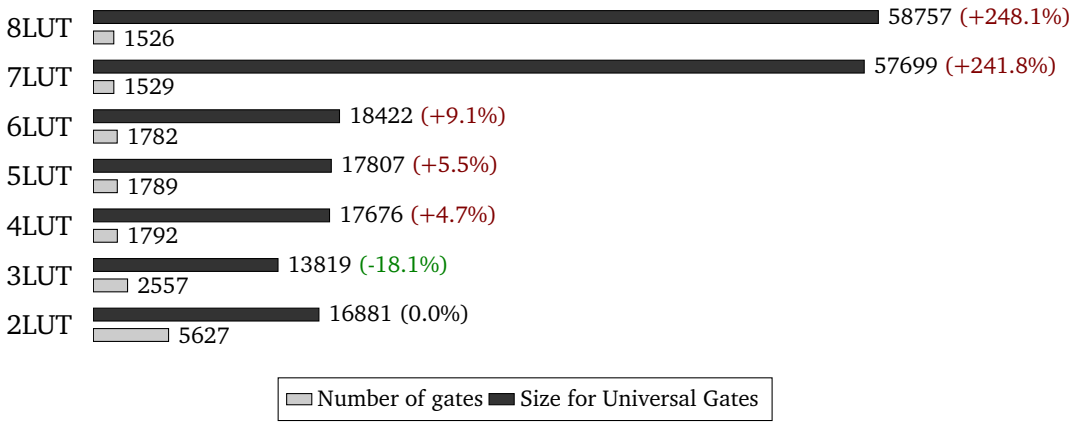


Figure 5.7: **Manhattan Distance:** Number of AND gates needed to instantiate the gates of the Manhattan circuit as Universal Gates w.r.t. different maximum LUT sizes. The number in braces is the relative difference to the number of AND gates needed for the Universal Gates with the binary construction (2LUT). Note that this is before fanout reduction of the circuit, which would possibly add copy gates.

circuit size reduction is not enough to reduce the UC size if the LUT sizes become too big. This is a problem because the size of the Universal Gates grows exponentially in its number of inputs (cf. Proposition 4), which can be seen in the difference from 6LUT to 7LUT in Figure 5.7. On the other side, the reduced number of gates in the 4LUT circuit is enough to compensate for the increased Universal Gate sizes compared to the 3LUT circuit. Note that the reduction of the circuit fanout will possibly further increase the number of gates, which was not considered by our observations so far. This effect is similar to the arithmetic circuits and can be seen in Figure 5.8.

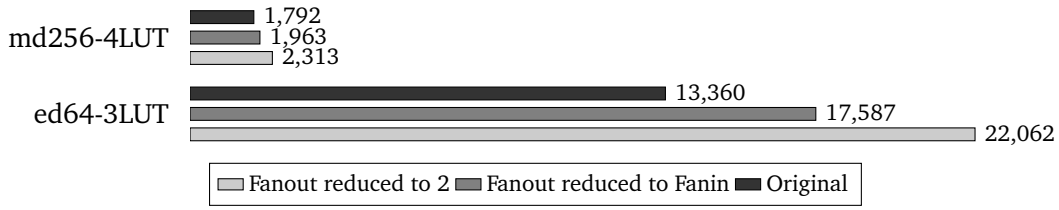


Figure 5.8: **Distance:** Comparison between the original circuit sizes and the same circuits after reducing the fanout. The used versions of the circuits are the ones that yielded the smallest UC size using the fixed or dynamic approach. Note that these are the circuit sizes before replacing wires by auxiliary poles in the dynamic construction.

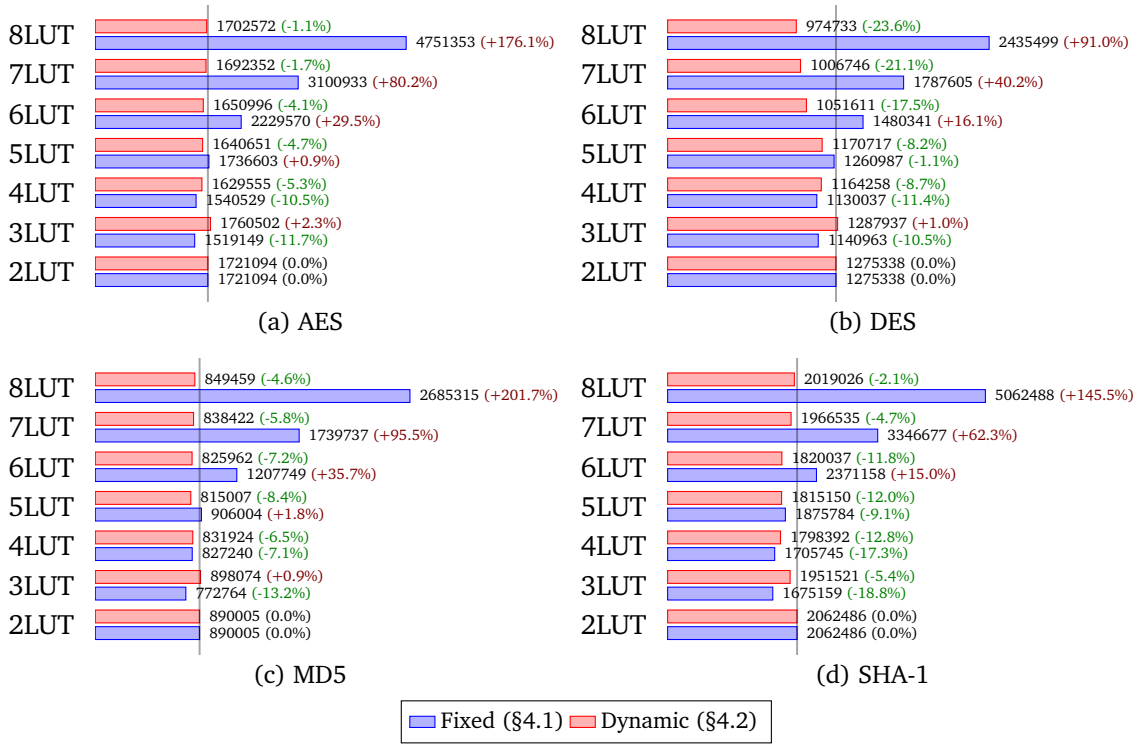
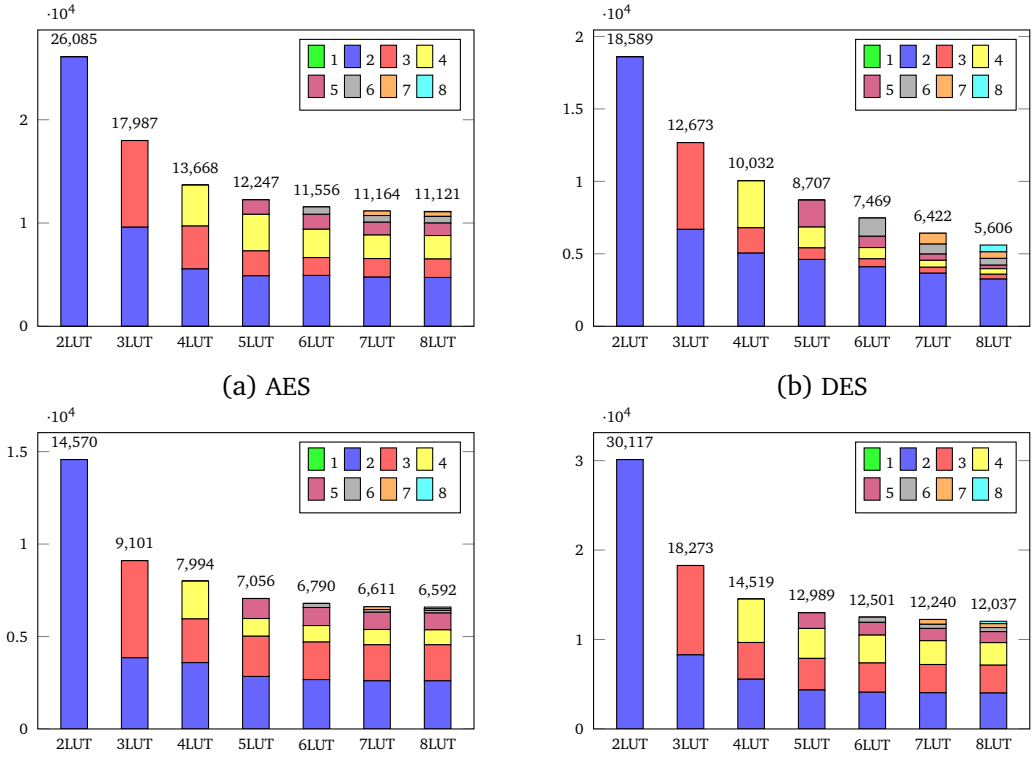


Figure 5.9: **Cryptographic:** Comparison of the UC sizes (number of AND gates) needed to embed cryptographic algorithms w.r.t. different LUT sizes. The vertical lines show the results of the binary approach. The number in braces is the relative improvement over the binary construction.

While we could always achieve a large UC size reductions with the prior arithmetic and distance circuits by using LUT sizes of 3 or 4, this no longer holds in such a scale for the cryptographic circuits (cf. Figure 5.9). Using multi-input gates, we could reduce the size of the AES embedding by 12% with the fixed construction, resp. by 5% with the dynamic construction. The best result could be achieved with the DES circuit. In this case, the dynamic construction with the 8LUT circuit achieves the smallest possible size, improving on the usual binary construction by 24%.

Comparing Figure 5.10 to the gate size distributions of the prior circuits (cf. Figure 5.2 and Figure 5.6), we again have the largest drop off going from 2LUT to 3LUT. For the DES circuit, there is even a circuit size reduction for higher LUT sizes, which explains why the dynamic construction is so efficient with a LUT size of 8.



(a) AES

(b) DES

(c) MD5

(d) SHA-1

Figure 5.10: **Cryptographic:** Distribution of the gate input sizes w.r.t. different LUT sizes. The different colors represent the used gate input sizes. Note that this figure only counts the number of gates and does not weight a multi-input gate more than a binary gate.

Figure 5.11 shows that the dynamic construction was able to yield the smallest UC for DES, although many copy gates were used to reduce the fanout of the circuit. In this case, the resulting circuit is more than two times the size of the circuit reduced to fanout 8 for the fixed construction.

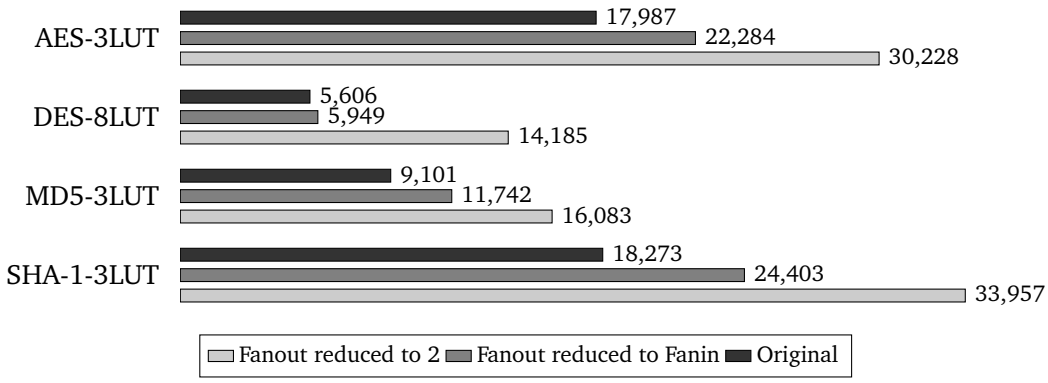


Figure 5.11: **Cryptographic:** Comparison between the original circuit sizes and the same circuits after reducing the fanout. The used versions of the circuits are the ones that yielded the smallest UC size using the fixed or dynamic approach. Note that these are the circuit sizes before replacing wires by auxiliary poles in the dynamic construction.

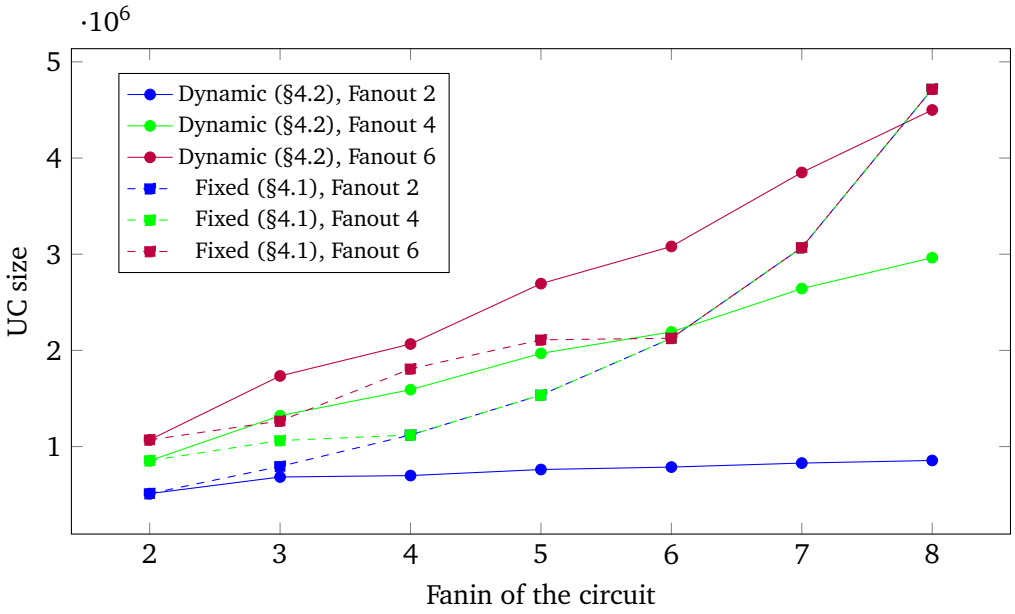


Figure 5.12: **Random:** Size of UCs when embedding random circuits with 1000 inputs, 10000 gates and 1000 outputs with different fanin and fanout. Solid lines are the results for the dynamic construction. Dashed lines are the results for the fixed construction.

Figure 5.12 demonstrates the potential of our dynamic construction if the circuits have highly varying gate input sizes. Especially for the random circuit with fanin 8 and fanout 2, most of the gates have few inputs, which is no problem for the dynamic construction as we only

"pay" per input that exceeds the usual 2 inputs. On the other hand, each Universal Gate in the fixed construction supports 8 inputs since this is the fanin. Therefore, 8 Γ_1 EUGs have to be merged. In this case, the size of the UC constructed by our dynamic construction is less than a fifth of the UC size with the fixed construction. However, due to the merging of 8 EUGs, each Universal Gate also supports 8 outgoing wires. The tides turn once the fanout of the circuit is large enough (fanout ≥ 6). At this point the dynamic construction must use many copy gates to forward the outputs of gates with outdegree greater than 2.

The random circuits show that a high fanin is better suited for the dynamic construction if there are enough gates with fewer inputs, while circuits with only little variation in the LUT sizes and no outliers to the top are better embedded using the fixed construction. This "rule of thumb" can be seen with random circuits with fanout 4 (lines in green). The fixed construction is smaller up to a fanin of 6, while the dynamic construction is preferred with fanins larger than 6.

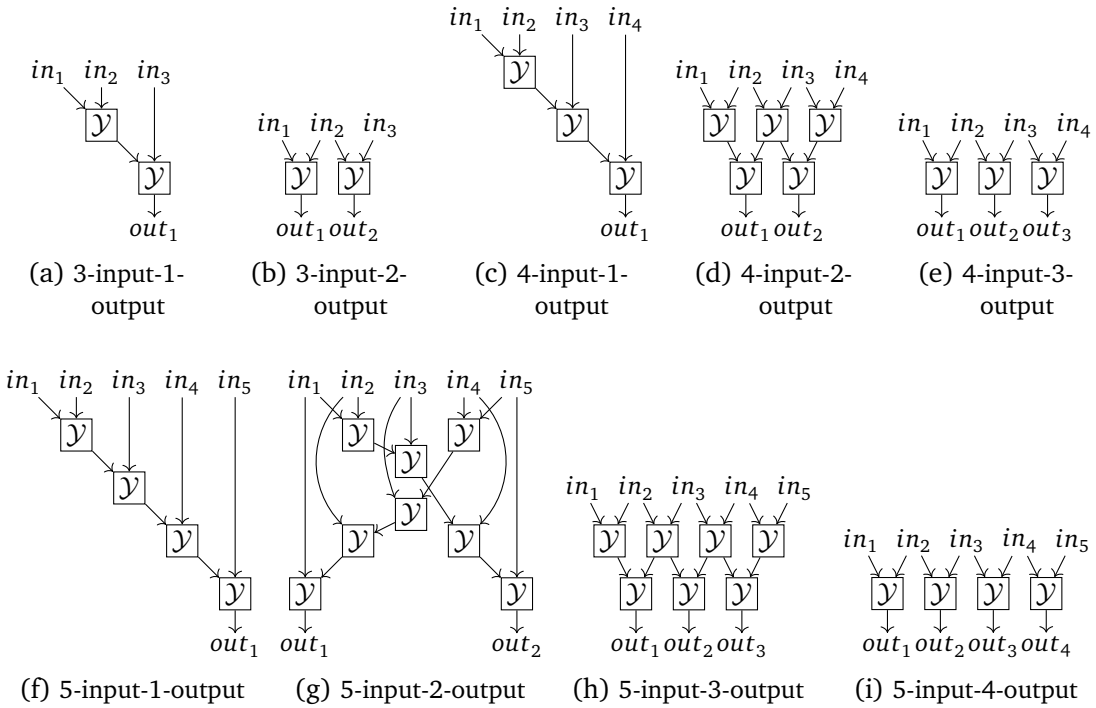
5.4 An Improvement Attempt: The Hybrid Construction

Section 5.3 showed the advantages that both multi-input constructions offer over the standard binary construction. However, depending on the situation, the dynamic construction is more efficient than the fixed one or vice-versa. In particular, the fixed constructions struggles with varying gate input sizes because each gate must support the highest indegree ρ, which results in ρ Γ_1 EUG mergings.

On the other hand, our dynamic construction suffers from the limited fanout of 2. Figures 5.4, 5.8 and 5.11 show that most circuits are almost doubled in size after reducing the fanout to 2. Reducing the fanout exactly to the fanin still results in an overhead. But, compared to the reduction to 2, the increase in size is drastically reduced. The worst case here is the 64-bit Karatsuba Multiplier with a maximum LUT size of 3 that increases by 34%.

Thus, we try to combine both approaches to a so called hybrid construction. The idea is that we use our new dynamic construction for Γ_1 graphs, but merge it now $k > 2$ times. This increases the minimum Universal Gate size to k, and we need to merge k Γ_1 EUGs. But, this decreases the need for copy gates since each gate now natively supports k outgoing edges. We refrain from providing a correctness proof since the proof for the hybrid construction is analogous to the proof for our basic dynamic multi-input gates construction. The only difference is that we choose a $\Gamma_k(n + \bar{\Delta})$ EUG for $\bar{\Delta} \coloneqq \sum_{i=0}^{n} \max\{\lceil \frac{p_i^+ - k}{k} \rceil, 0\}$ in the second step. In order to only forward the desired number of inputs to the Universal Gates, we need a more sophisticated construction than a simple Y-Switch for the auxiliary poles forwarding less than k inputs.

Recall that the problem is that every auxiliary pole, by default, forwards exactly k wires, where k is the number of Γ_1 EUG mergings. To only forward some of the ingoing wires, we use so called selection networks. These networks are small graphs which ensure that only

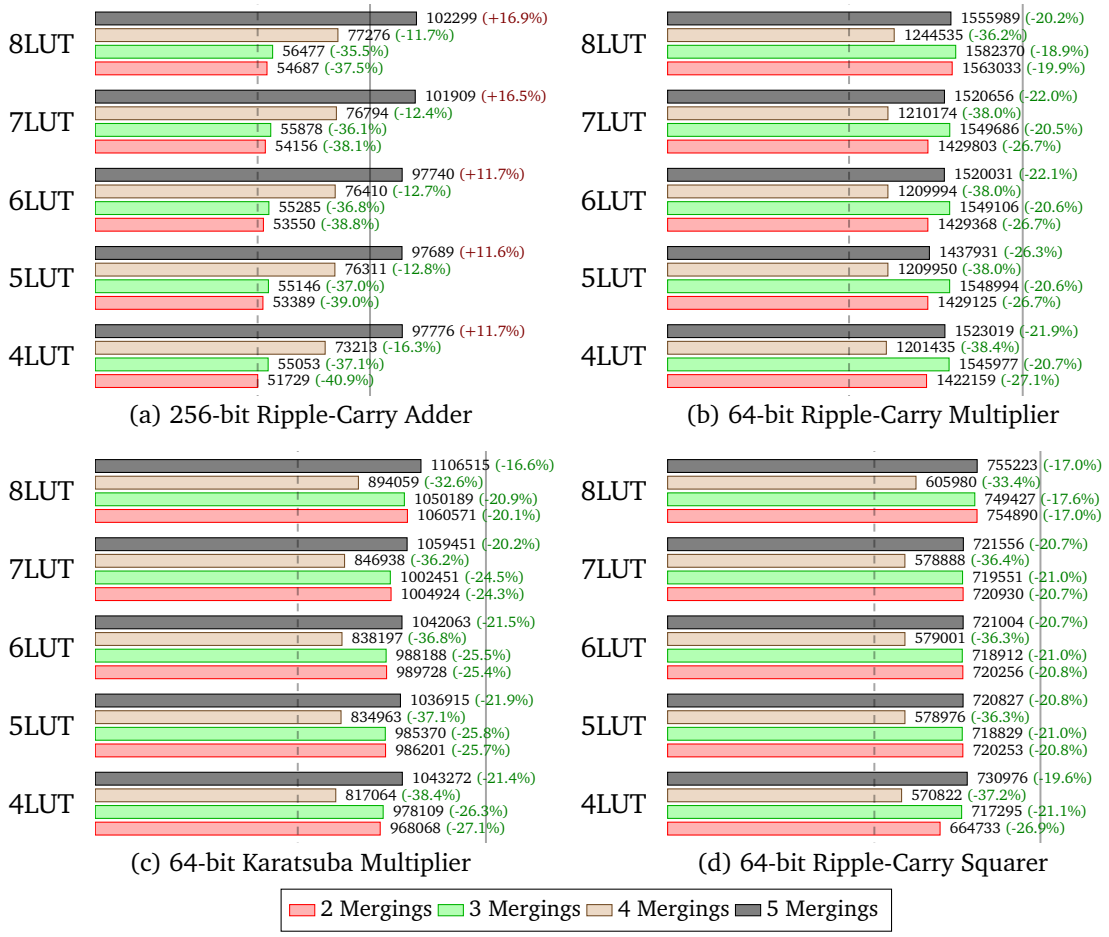


(a) 3-input-1-output (b) 3-input-2-output (c) 4-input-1-output (d) 4-input-2-output (e) 4-input-3-output

(f) 5-input-1-output (g) 5-input-2-output (h) 5-input-3-output (i) 5-input-4-output

Figure 5.13: The selection networks used in the hybrid construction (§ 5.4) for auxiliary poles with 3,4, or 5 inputs.

the desired inputs to the auxiliary poles are forwarded to the actual multi-input gate. The constructions consist of multiple Y-Switches and the control bits of these switches can be set such that only the desired inputs are forwarded. The constructions are given in Figure 5.13. Each auxiliary pole that forwards less than than k inputs will be replaced by such a selection network. The constructions only depend on the number of inputs and outputs, and the control bits remain private to the function owner. The order of the inputs to the multi-input gate is irrelevant since we can set the function table of the Universal Gates freely. Note that the hybrid construction is the same as the dynamic constructions if 2 mergings are used. On the other hand, the hybrid construction becomes the fixed construction if the number of mergings equals the highest gate input size because no auxiliary poles are needed in this case. Therefore, when referring to the hybrid construction, we exclude these cases.



(a) 256-bit Ripple-Carry Adder

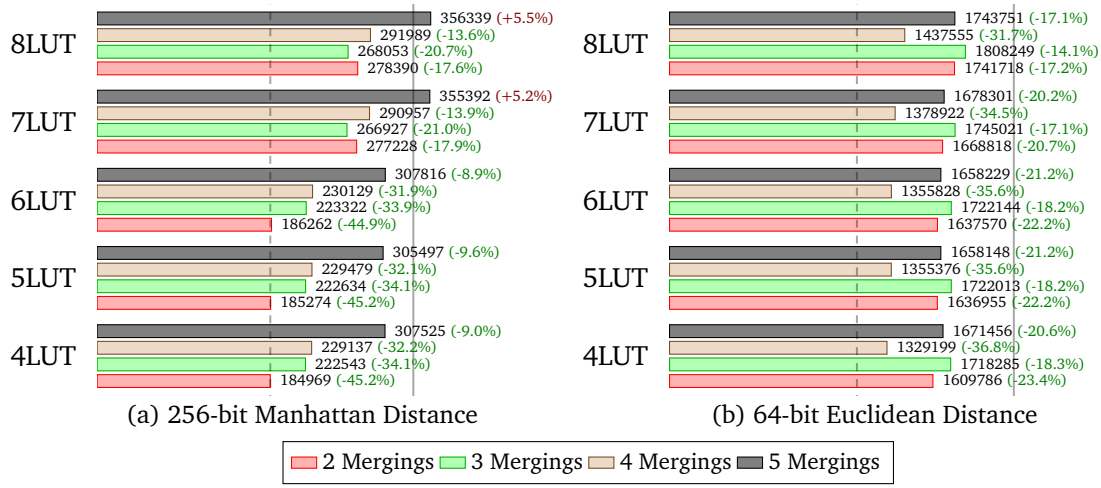
(b) 64-bit Ripple-Carry Multiplier

(c) 64-bit Karatsuba Multiplier

(d) 64-bit Ripple-Carry Squarer

Figure 5.14: **Hybrid, Arithmetic:** UC sizes (number of AND gates) of cryptographic algorithms with the hybrid construction (§5.4) w.r.t. the number of mergings. The dashed vertical lines show the best results with the fixed (§4.1) or dynamic (§4.2) construction. The solid vertical lines show the results for the standard binary construction. The number in braces is the relative improvement over the binary construction.

Figure 5.14 shows that the hybrid construction with 3 or 4 mergings always yields smaller UC sizes than the standard binary construction. Using 5 mergings always increases the UC sizes compared to 3 or 4 mergings. The problem is the same as with the fixed constructions because 5 Γ_1 EUGs are merged, although there are only few gates using this many inputs (cf. Figure 5.2). The benchmarks for the distance circuits in Figure 5.15 show the same behavior.



(a) 256-bit Manhattan Distance

(b) 64-bit Euclidean Distance

Figure 5.15: **Hybrid, Distance:** UC sizes (number of AND gates) of distance algorithms with the hybrid construction (§5.4) w.r.t. the number of mergings. The dashed vertical lines show the best results with the fixed (§4.1) or dynamic (§4.2) construction. The vertical lines show the results for the standard binary construction. The number in braces is the relative improvement over the binary construction.

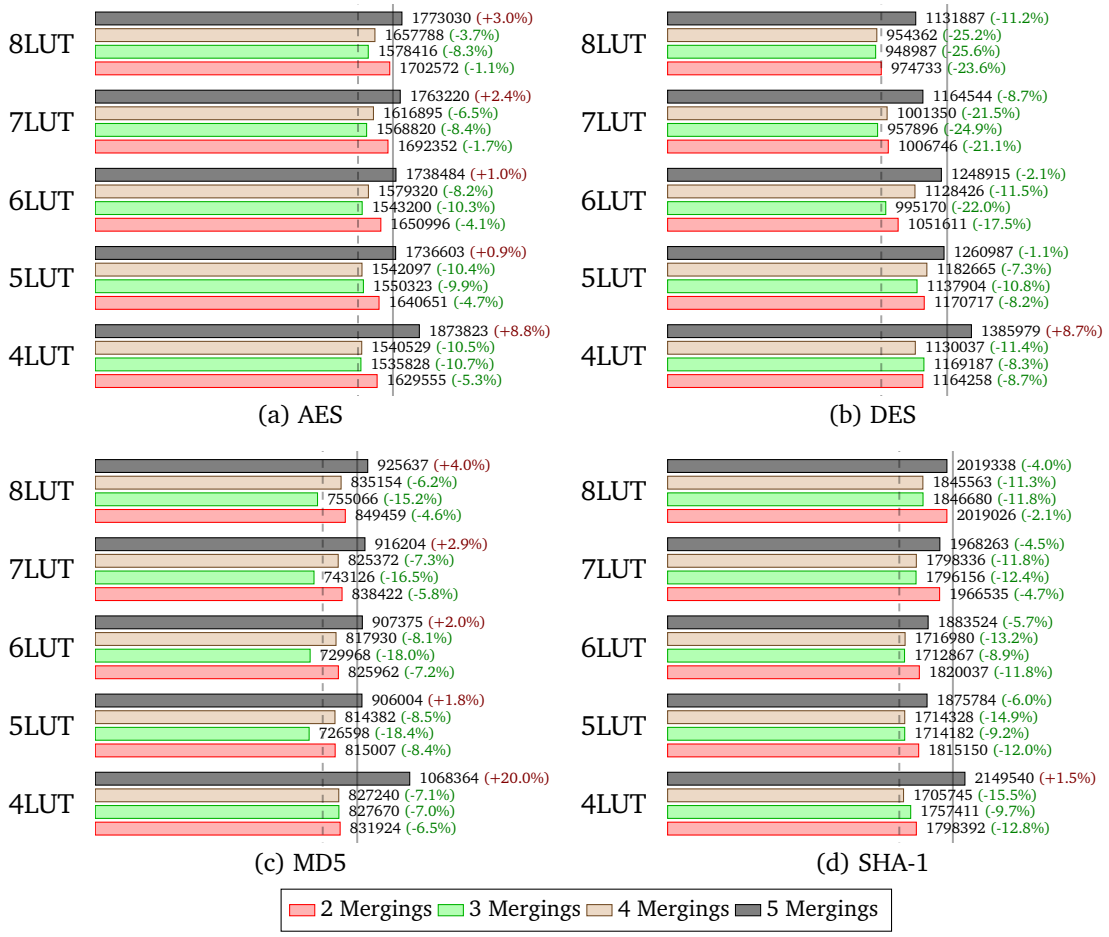


Figure 5.16: **Hybrid, Crypto:** UC sizes (number of AND gates) of cryptographic algorithms with the hybrid construction (§5.4) w.r.t. the number of mergings. The dashed vertical lines show the best results with the fixed (§4.1) or dynamic (§4.2) construction. The vertical lines show the results for the standard binary construction. The number in braces is the relative improvement over the binary construction.

The first circuits where the hybrid construction yields the smallest sizes (compared to the best case for the dynamic or fixed construction) are the DES and MD5 circuit. Those UC sizes could be reduced by 26% and 18% compared to the standard binary construction.

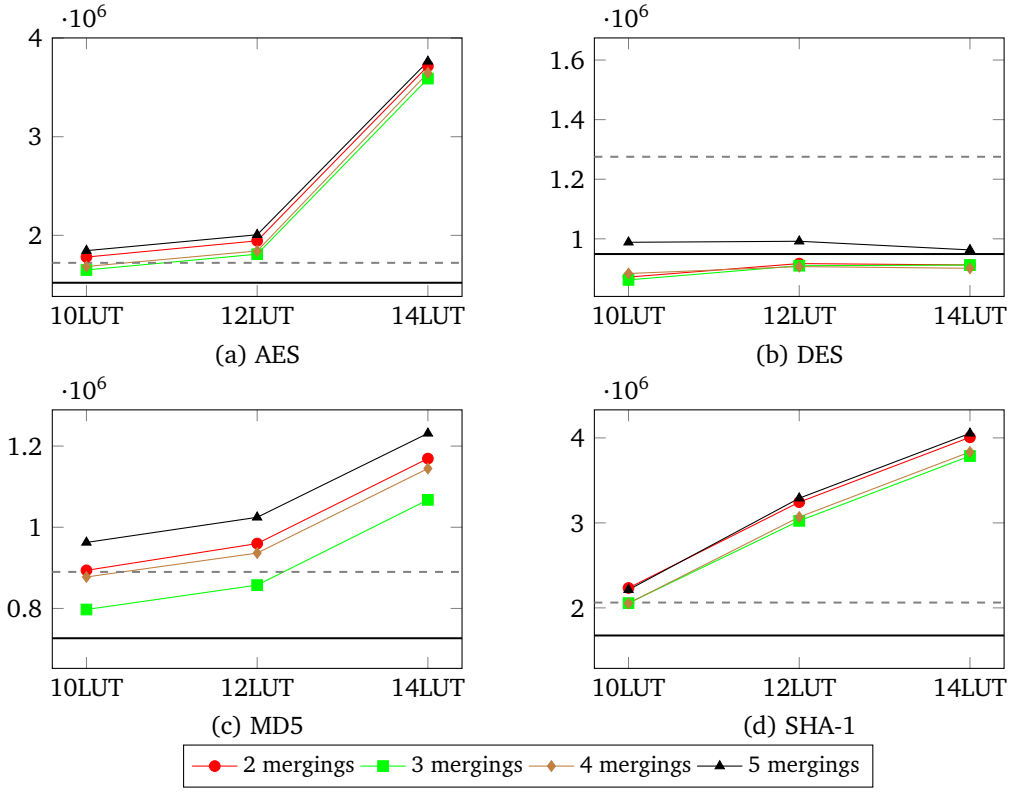


Figure 5.17: **Hybrid, Crypto:** UC sizes (number of AND gates) of cryptographic algorithms with the hybrid construction (§5.4) and very high LUT sizes. The solid black line represents the best UC size so far. The dashed gray line represents the result for the standard binary construction.

What happens if we further increase the LUT size? The only circuit that can be further reduced in size by allowing higher LUT sizes is the DES circuit. All other circuits have their sweet spot between 3 and 4. Figure 5.17 shows that we can further decrease the UC size of the DES circuit to 862,203 AND gates which is an improvement of 33% over the binary construction with 1,275,338 AND gates.

	Fixed (§4.1)	Dynamic (§4.2)	Hybrid (§5.4)
256-bit Ripple-Carry Adder	37%	**41%**	37%
64-bit Ripple-Carry Multiplier	**49%**	38%	38%
64-bit Karatsuba Multiplier	**48%**	36%	37%
64-bit Ripple-Carry Squarer	**45%**	37%	36%
256-bit Manhattan Distance	40%	**45%**	34%
64-bit Euclidean Distance	**45%**	37%	36%
AES	**12%**	5%	11%
DES	11%	**33%**	**33%**
MD5	13%	8%	**18%**
SHA-1	**19%**	13%	15%
Average	**32%**	29%	30%

Table 5.4: The relative improvement in the UC sizes over the binary construction. The improvement is calculated with the best possible LUT size. For the hybrid construction (§5.4) either 3 or 4 mergings are used, where we excluded cases in which the hybrid construction becomes the fixed construction (§4.1). The highest improvement per circuit is marked in bold.

6 Conclusion

In this work, we implemented and extended the state-of-the-art UC construction of Liu et al. [LYZ+21] to support multi-input gates and showed the practicality of this approach. The extension to multi-input gates uses the underlying EUG as a black box and can be used with any other EUG too. We showed that all three proposed constructions (fixed (§4.1), dynamic (§4.2), hybrid (§5.4)) reduced the UC sizes compared to the standard binary gate UC construction by almost a third on average (cf. Table 5.4). The UC size reduction for cryptographic circuits was always lower than for arithmetic or distance circuits. The concrete improvement depends heavily on the concrete circuit. For each construction, there are circuits where they yield the smallest UC size. Our new dynamic construction showed its potential if high LUT sizes are used with many smaller gates as with the random circuits or the DES circuit in the 8LUT version. It always yields smaller UCs than the binary construction for all testes LUT sizes. However, if the best possible LUT size is used, the fixed construction that relies on the ideas of [Val76][SS08, §4.3] is the most efficient. But, the size of fixed construction grows linearly with the fanin of the circuit, which can be a problem if only few gates with many inputs are used. In those cases, the dynamic and hybrid construction are the most efficient.

Future Work

Further UC size reduction could possibly be achieved by allowing gates to have multiple distinct outputs.

We saw that the sweet spot for the LUT size is between 3 and 4 for most of the circuits because this gives the largest circuit size reduction while keeping the LUT sizes small. Higher LUT sizes only seemed to work for the DES circuit. Therefore, a linear PFE scheme like [KM11; MS13; MSS14; BBK18; HKRS20] supporting gates with 3 or 4 inputs would be desirable. We also saw that a high number of outgoing wires per gate significantly increases the circuit sizes. An interesting area of research that directly improves the UC sizes is the synthesizing of circuits with higher LUT sizes and optimized gate outputs, which yield smaller circuits, and therefore, smaller UCs.

Further research can possibly find a way to circumvent the need for copy gates and directly allow higher fanouts.

Because our construction uses the concrete Γ_1 EUG construction as a black-box algorithm, any improvement on Γ_1 EUGs directly improves our proposed construction.

List of Figures

List of Tables

List of Abbreviations

STPC Secure Two-Party Computation

PFE Private Function Evaluation

SPFE Semi-Private Function Evaluation

UC Universal Circuit

EUG Edge-Universal Graph

LUT Lookup Table

SHDL Secure Hardware Definition Language

FPGA Field Programmable Gate Array

Bibliography

[AGKS20] M. Y. Alhassan, D. Günther, Á. Kiss, T. Schneider. **"Efficient and Scalable Universal Circuits"**. In: *J. Cryptol.* 33.3 (2020), pp. 1216–1271. Code: https://github.com/encryptogroup/UC.

[Alo03] N. Alon. **"A simple algorithm for edge-coloring bipartite multigraphs"**. In: *Information Processing Letters* 85.6 (2003), pp. 301–302.

[BBK18] O. Biçer, M. A. Bingöl, M. S. Kiraz. **"Highly Efficient and Reusable Private Function Evaluation with Linear Complexity"**. In: *IACR Cryptol. ePrint Arch.* (2018). Paper is accepted at TDSC, p. 515.

[BBKL19] M. A. Bingöl, O. Biçer, M. S. Kiraz, A. Levi. **"An Efficient 2-Party Private Function Evaluation Protocol Based on Half Gates"**. In: *Comput. J.* 62.4 (2019), pp. 598–613.

[BFK+09] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi, T. Schneider. **"Secure Evaluation of Private Linear Branching Programs with Medical Applications"**. In: *ESORICS*. Vol. 5789. Springer, 2009, pp. 424–439.

[BPSW07] J. Brickell, D. E. Porter, V. Shmatikov, E. Witchel. **"Privacy-preserving remote diagnostics"**. In: *CCS*. ACM, 2007, pp. 498–507.

[CMCB07] S. Cho, A. Mishchenko, S. Chatterjee, R. Brayton. **"Efficient FPGA mapping using Priority Cuts"**. In: *FPGA*. 2007.

[COS01] R. Cole, K. Ost, S. Schirra. **"Edge-Coloring Bipartite Multigraphs in O(E log D) Time"**. In: *Combinatorica* 21.1 (2001), pp. 5–12.

[Die10] R. Diestel. **"Graph Theory"**. Fourth. Vol. 173. Springer, 2010.

[DKS+17] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, M. Zohner. **"Pushing the Communication Barrier in Secure Computation using Lookup Tables"**. In: *NDSS*. The Internet Society, 2017.

[FAZ05] K. B. Frikken, M. J. Atallah, C. Zhang. **"Privacy-preserving credit checking"**. In: *EC*. ACM, 2005, pp. 147–154.

[FKSW19] S. Felsen, Á. Kiss, T. Schneider, C. Weinert. **"Secure and Private Function Evaluation with Intel SGX"**. In: *CCSW@CCS*. ACM, 2019, pp. 165–181.

[FVK+15] B. A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, S. M. Bellovin. **"Malicious-Client Security in Blind Seer: A Scalable Private DBMS"**. In: *S & P*. IEEE, 2015, pp. 395–410.

[GKS17] D. GÜNTHER, Á. KISS, T. SCHNEIDER. **"More Efficient Universal Circuit Constructions"**. In: *ASIACRYPT (2)*. Vol. 10625. Springer, 2017, pp. 443–470.

[GKSS19] D. GÜNTHER, Á. KISS, L. SCHEIDEL, T. SCHNEIDER. **"Poster: Framework for Semi-Private Function Evaluation with Application to Secure Insurance Rate Calculation"**. In: *CCS*. ACM, 2019, pp. 2541–2543.

[GMW87] O. GOLDREICH, S. MICALI, A. WIGDERSON. **"How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority"**. In: *STOC*. ACM, 1987, pp. 218–229.

[HKRS20] M. HOLZ, Á. KISS, D. RATHEE, T. SCHNEIDER. **"Linear-Complexity Private Function Evaluation is Practical"**. In: *ESORICS (2)*. Vol. 12309. Springer, 2020, pp. 401–420.

[Kar53] M. KARNAUGH. **"The map method for synthesis of combinational logic circuits"**. In: *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics* 72.5 (1953), pp. 593–599.

[KM11] J. KATZ, L. MALKA. **"Constant-Round Private Function Evaluation with Linear Complexity"**. In: *ASIACRYPT*. Vol. 7073. Springer, 2011, pp. 556–571.

[KS08a] V. KOLESNIKOV, T. SCHNEIDER. **"A Practical Universal Circuit Construction and Secure Evaluation of Private Functions"**. In: *FC*. Vol. 5143. Springer, 2008, pp. 83–97.

[KS08b] V. KOLESNIKOV, T. SCHNEIDER. **"Improved Garbled Circuit: Free XOR Gates and Applications"**. In: *ICALP (2)*. Vol. 5126. Springer, 2008, pp. 486–498.

[KS16] Á. KISS, T. SCHNEIDER. **"Valiant's Universal Circuit is Practical"**. In: *EUROCRYPT (1)*. Vol. 9665. Springer, 2016, pp. 699–728.

[LMS16] H. LIPMAA, P. MOHASSEL, S. S. SADEGHIAN. **"Valiant's Universal Circuit: Improvements, Implementation, and Applications"**. In: *IACR Cryptol. ePrint Arch.* (2016), p. 17.

[LYZ+21] H. LIU, Y. YU, S. ZHAO, J. ZHANG, W. LIU, Z. HU. **"Pushing the Limits of Valiant's Universal Circuits: Simpler, Tighter and More Compact"**. In: *CRYPTO (2)*. Vol. 12826. Springer, 2021, pp. 365–394.

[MNPS04] D. MALKHI, N. NISAN, B. PINKAS, Y. SELLA. **"Fairplay - Secure Two-Party Computation System"**. In: *USENIX Security Symposium*. USENIX, 2004, pp. 287–302.

[MS13] P. MOHASSEL, S. S. SADEGHIAN. **"How to Hide Circuits in MPC an Efficient Framework for Private Function Evaluation"**. In: *EUROCRYPT*. Vol. 7881. Springer, 2013, pp. 557–574.

[MSS14] P. MOHASSEL, S. S. SADEGHIAN, N. P. SMART. **"Actively Secure Private Function Evaluation"**. In: *ASIACRYPT (2)*. Vol. 8874. Springer, 2014, pp. 486–505.

[NSMS14] S. NIKSEFAT, B. SADEGHIYAN, P. MOHASSEL, S. S. SADEGHIAN. **"ZIDS: A Privacy-Preserving Intrusion Detection System Using Secure Two-Party Computation Protocols"**. In: *Comput. J.* 57.4 (2014), pp. 494–509.

[OI07] R. OSTROVSKY, W. E. S. III. **"Private Searching on Streaming Data"**. In: *J. Cryptol.* 20.4 (2007), pp. 397–430.

[Pin02] B. PINKAS. **"Cryptographic Techniques for Privacy-Preserving Data Mining"**. In: *SIGKDD Explor.* 4.2 (2002), pp. 12–19.

[PKV+14] V. PAPPAS, F. KRELL, B. VO, V. KOLESNIKOV, T. MALKIN, S. G. CHOI, W. GEORGE, A. D. KEROMYTIS, S. M. BELLOVIN. **"Blind Seer: A Scalable Private DBMS"**. In: *S & P*. IEEE, 2014, pp. 359–374.

[PSS09] A. PAUS, A.-R. SADEGHI, T. SCHNEIDER. **"Practical Secure Evaluation of Semi-private Functions"**. In: *ACNS*. Vol. 5536. 2009, pp. 89–106.

[Qui52] W. V. QUINE. **"The Problem of Simplifying Truth Functions"**. In: *The American Mathematical Monthly* 59.8 (1952), pp. 521–531.

[RR21] M. ROSULEK, L. ROY. **"Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits"**. In: *CRYPTO (1)*. Vol. 12825. Springer, 2021, pp. 94–124.

[SS08] A.-R. SADEGHI, T. SCHNEIDER. **"Generalized Universal Circuits for Secure Evaluation of Private Functions with Application to Data Classification"**. In: *ICISC*. Vol. 5461. Springer, 2008, pp. 336–353.

[ST] N. SMART, S. TILLICH. **"'Bristol Fashion' MPC Circuits"**. https://homes.esat.kuleuven.be/~nsmart/MPC/old-circuits.html.

[SYY99] T. SANDER, A. L. YOUNG, M. YUNG. **"Non-Interactive CryptoComputing For NC^1"**. In: *FOCS*. IEEE, 1999, pp. 554–567.

[Val76] L. G. VALIANT. **"Universal Circuits (Preliminary Report)"**. In: *STOC*. ACM, 1976, pp. 196–203.

[Weg87] I. WEGENER. **"The Complexity of Boolean Functions"**. John Wiley; Sons, Inc., 1987, pp. 90–92.

[Yao86] A. C.-C. YAO. **"How to Generate and Exchange Secrets (Extended Abstract)"**. In: *FOCS*. IEEE, 1986, pp. 162–167.

[ZRE15] S. ZAHUR, M. ROSULEK, D. EVANS. **"Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates"**. In: *EUROCRYPT (2)*. Vol. 9057. Springer, 2015, pp. 220–250.

[ZYZL19] S. ZHAO, Y. YU, J. ZHANG, H. LIU. **"Valiant's Universal Circuits Revisited: An Overall Improvement and a Lower Bound"**. In: *ASIACRYPT (1)*. Vol. 11921. Springer, 2019, pp. 401–425.

A Appendix

A.1 Benchmarks with Valiant's EUG Construction

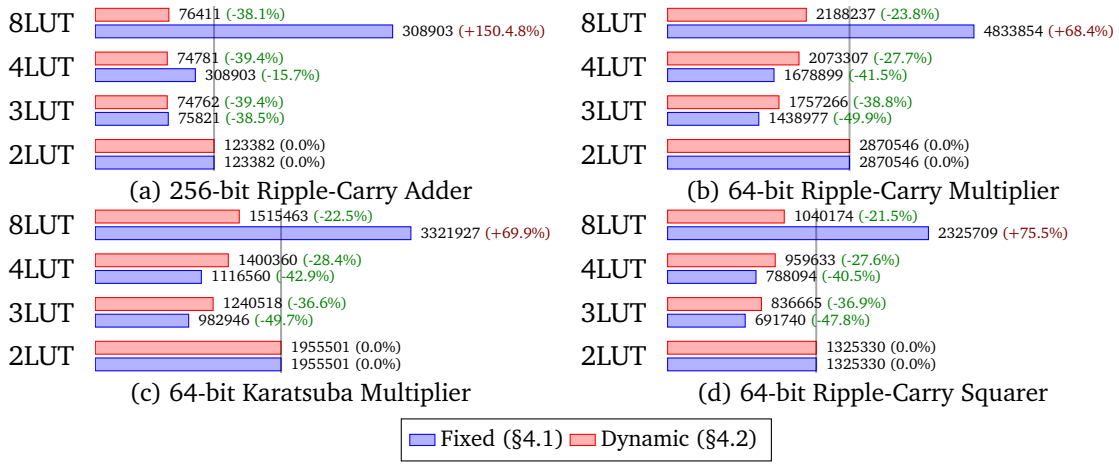


Figure A.1: **Valiant, Arithmetic:** Comparison of the UC sizes (number of AND gates) needed to embed arithmetic operations w.r.t. different LUT sizes and Valiant's 2-Way split EUG [Val76]. The vertical lines show the results of the binary approach. The number in braces is the relative improvement over the binary construction.

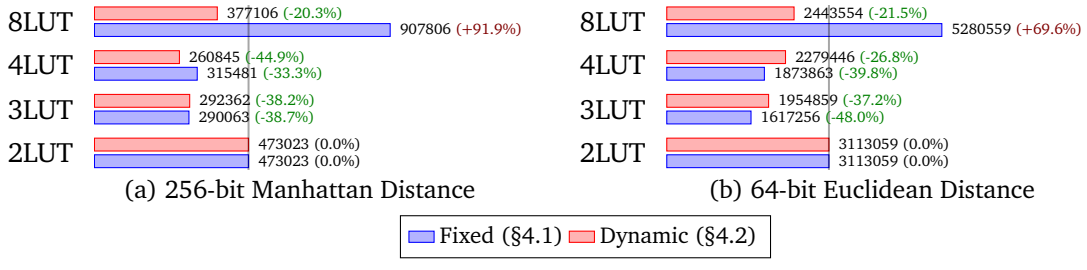


Figure A.2: **Valiant, Distance:** Comparison of the UC sizes (number of AND gates) needed to embed arithmetic operations w.r.t. different LUT sizes and Valiant's 2-Way split EUG [Val76]. The vertical lines show the results of the binary approach. The number in braces is the relative improvement over the binary construction.

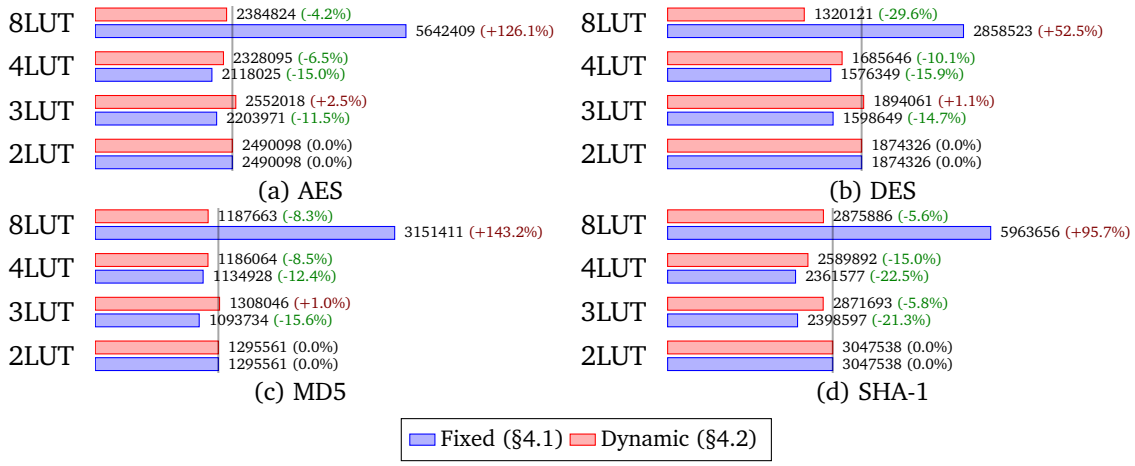


Figure A.3: **Valiant, Crypto:** Comparison of the UC sizes (number of AND gates) needed to embed arithmetic operations w.r.t. different LUT sizes and Valiant's 2-Way split EUG [Val76]. The vertical lines show the results of the binary approach. The number in braces is the relative improvement over the binary construction.

A.2 Compilation Times

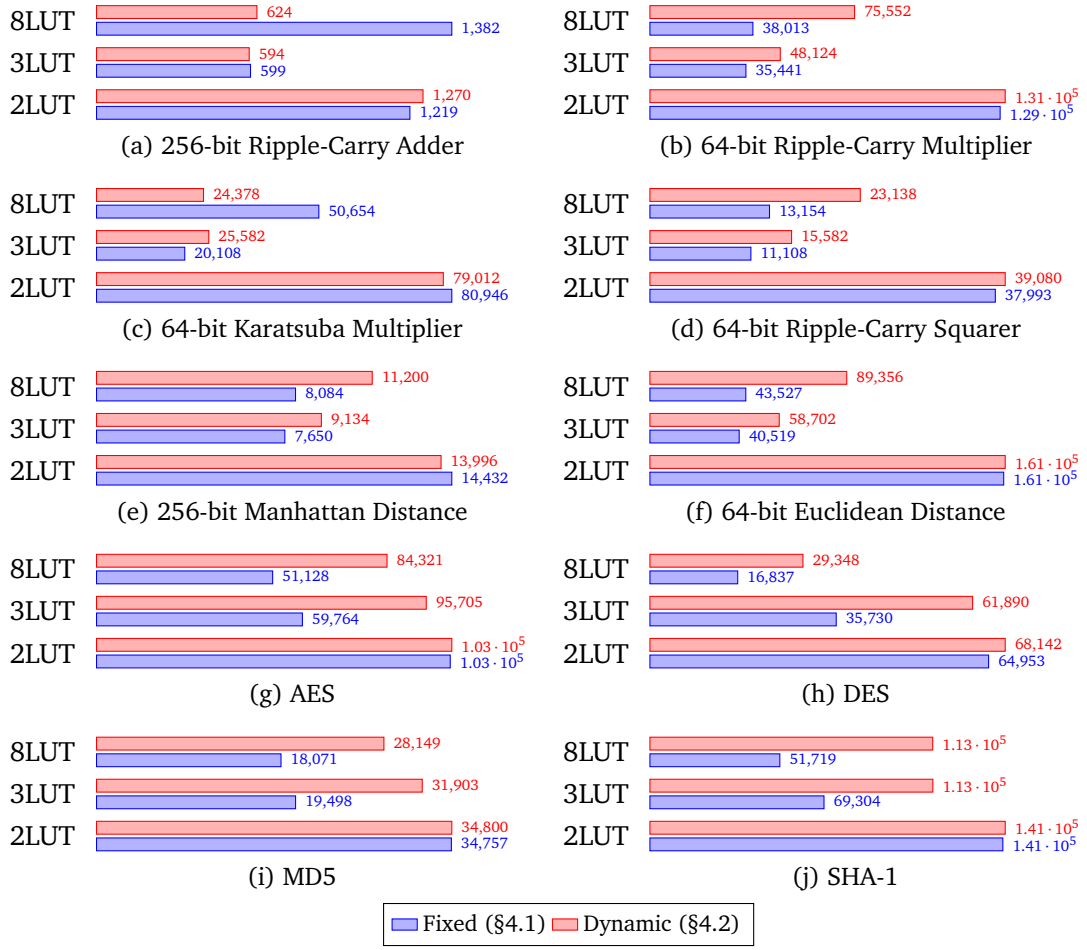


Figure A.4: Compilation times in milliseconds for creating the UC (with Liu et al.'s EUG [LYZ+21]) with corresponding programming bits without correctness checks. Note that compilation time is a one-time expense that can be done offline once the function is known. The circuit size is the crucial factor in the online evaluation. The circuits were input in SHDL format. The times are the average of 100 runs. No special compiler optimization was used. The used system consisteted of an AMD Ryzen 3700x (3,6GHz) and 16GB DDR4-3000 RAM. The used OS was Linux Mint 20.1 with kernel version 5.11.0-051100-generic.