Secure Maximum Weight Matching Approximation on General Graphs

Andreas Brüggemann[*] Malte Breuer[†] Andreas Klinger[‡] Thomas Schneider[§]
Ulrike Meyer[¶]

September 7, 2022

Abstract

Privacy-preserving protocols for matchings on general graphs can be used for applications such as online dating, bartering, or kidney donor exchange. In addition, they can act as a building blocks for more complex protocols. While privacy preserving protocols for matchings on bipartite graphs are a well-researched topic, the case of general graphs has experienced significantly less attention so far. We address this gap by providing the first privacy-preserving protocol for maximum weight matching on general graphs. We present two protocol variants, which both compute an $1/2-$approximation instead of an optimal solution in favor of scalability. For N nodes, the first variant requires $\mathcal{O}(N \log^2 N)$ rounds and $\mathcal{O}(N^3 \log N)$ communication, and the second variant requires only $\mathcal{O}(N \log N)$ rounds and $\mathcal{O}(N^3)$ communication. We implement both variants and find that the first variant runs in 14.9 minutes for $N = 300$ nodes, while the second variant requires only 5.1 minutes for $N = 300$, and 12.5 minutes for $N = 400$.

1 Introduction

Secure Multi-Party Computation (MPC) allows multiple parties to evaluate some function on their private inputs using a distributed protocol that keeps everything that a party cannot derive from its input and output private [Yao82]. It has been used to devise privacy-preserving protocols for a variety of matching problems on graphs that are used, e.g., for fingerprint identification [BS15], kidney donor exchange [BMW22], matching medical students to medical residency programs or students to schools or universities [Gol06; DE16; Ria+17], and are used as subroutines for tackling entirely different problems [Wül+17]. While almost all of these approaches are specifically designed for bipartite graphs only, use cases such as online dating, bartering, or kidney exchange can profit from or even require to use general graphs.

To the best of our knowledge, Breuer et al. [BMW22] have proposed the first and to date only privacy-preserving protocol for matching on general graphs. Their protocol always returns an optimal solution, but exhibits a high complexity of $\mathcal{O}(N^4)$ (communication) rounds and $\mathcal{O}(N^5)$ communication (i.e., traffic) for N nodes. This yields a run time of more than a day for $N = 40$ which may be infeasible for use cases where strict time constraints or large graphs are given.

Our Contributions. We present the first approximation protocol for maximum weight matching (MWM) on weighted general graphs. Here, the objective is to find a matching that maximizes the sum of weights of its edges. Our protocol thus approximatively solves a more general problem than

[*]Technical University of Darmstadt, Germany. brueggemann@encrypto.cs.tu-darmstadt.de
[†]RWTH Aachen University, Germany. breuer@itsec.rwth-aachen.de
[‡]RWTH Aachen University, Germany. klinger@itsec.rwth-aachen.de
[§]Technical University of Darmstadt, Germany. schneider@encrypto.cs.tu-darmstadt.de
[¶]RWTH Aachen University, Germany. meyer@itsec.rwth-aachen.de

the maximum matching problem (i.e., the MWM problem where all weights are 1) for which Breuer et al. [BMW22] present an optimal solution. Our protocol achieves a high scalability while computing an approximation of the optimal solution only. Yet it guarantees that the computed matching has at least half of the maximally possible weight, which dependent on the use case may be well-acceptable.

We use randomization to obtain two variants of the protocol that give certain unbiasedness guarantees between edges of equal weight or their corresponding nodes. The complexity of the first variant for N nodes is $\mathcal{O}(N \log^2 N)$ rounds and $\mathcal{O}(N^3 \log N)$ communication. The faster, second variant requires $\mathcal{O}(N \log N)$ rounds and $\mathcal{O}(N^3)$ communication. We prove the protocol's correctness and security in presence of a passive adversary and an honest majority and implement it using the MPC framework MP-SPDZ [Kel20] to demonstrate its scalability. Using three parties in a LAN setting and the three-party MPC protocol by Araki et al. [Ara+16], we are able to run the first variant that provides stronger unbiasedness guarantees in 14.9 minutes with 17.1 GB communication for $N = 300$. The second variant runs in 12.5 minutes with 16.4 GB communication for $N = 400$.

Applications. In general, privacy-preserving matching protocols are of interest to be used as a building block for more complex protocols. For instance, Wüller et al. [Wül+17] use a building block for maximum weight perfect matchings in the bipartite case to find exchange cycles for bartering. We broaden our motivation for MWM approximation by the following two use cases:

For *online dating* it is especially important to protect the users' data like sexuality and sexual preferences that may be subject to social stigma. Privacy-preserving approaches that allow users to query similar user profiles from an encrypted database [Yi+16; WW18] exist, but another approach is to directly match pairs of people with high expected compatibility. While dating is even used to illustrate, e.g., the stable matching problem (also called stable *marriage* problem) or Hall's *marriage* theorem [Hal35] (e.g., in [Knu96; Gol06; Die17]), these cases concern bipartite graphs and thus assume heterosexuality where man and women are matched. Matchings on general graphs can instead be used to additionally allow to model monogamous relationships free of further assumptions regarding sexuality. Furthermore, weights can be used to represent a non-binary interpersonal compatibility measure. We argue that for online dating it is valid to tolerate losses from the approximation to enhance scalability as good but suboptimal matches may be sufficient and the quality of matches could even be increased by using larger pools of participants.

Another use case is multi-party *bartering* where commodities are traded between parties without involvement of a currency. MPC can be applied to keep the commodities offered and wanted by the parties private. This has already been done letting parties trade their commodities in exchange cycles (e.g., [Wül+17; WMW17b; WMW17a]). For cases where each party wants to trade with at most a single designated party to reduce logistic effort, a matching protocol for general graphs can be used to match parties that can trade among themselves. Weights can be used to model the participants' satisfaction with certain trades. If it is not indispensable to find the optimal trades, or the benefit of using larger pools of participants outweighs the benefit of optimal solutions, using an approximation instead of an exact solution may again be a valid option.

Outline. The remainder of this paper is structured as follows: In Section 2, we introduce the required definitions, notations, and building blocks regarding MPC and graph theory. Section 3 discusses related work. Our protocol for privacy-preserving MWM approximation is presented in Section 4 and evaluated in Section 5. In Section 6, we give an outlook regarding future work.

2 Preliminaries

2.1 Secure Multi-Party Computation

MPC allows p parties to evaluate a function on their private inputs using a distributed protocol such that no party is able to learn information that it cannot derive from its own input and output [Yao82]. In this section we give a formal definition of the security to be provided by our protocols, specify how inputs and intermediate results are represented and used in basic arithmetic operations, and discuss building blocks for operations that we require later.

2.1.1 Security

To formalize the security guarantees of our protocol, we use standard definitions as in [Can00; AL17; Lin17]. We assume an honest majority, meaning that an adversary can only corrupt a minority of parties. Furthermore, we suppose that each pair of parties is connected by a private, authenticated channel. We focus on passive security where an adversary learns the corrupted parties' views on the computation but cannot modify their behavior and proceed by giving a brief definition of passive security.

Let parties $P_0, ..., P_{p-1}$ compute a function $\mathcal{F} : (\{0,1\}^*)^p \to (\{0,1\}^*)^p$ that may be probabilistic using a corresponding protocol π. We define I as the set of all corrupted parties' indexes, let $\hat{x}$ contain all parties' inputs, and let $\hat{x}_I$ only contain the corrupted parties' inputs. Let $\mathcal{F}_I(\hat{x})$ be all corrupted parties' results of $\mathcal{F}$ on input $\hat{x}$. Furthermore, we introduce a statistical security parameter $\sigma \in \mathbb{N}$. Finally, for an execution of protocol π with fixed σ let $\textsc{view}_I^\pi(\hat{x}, \sigma)$ contain the input, random tape, and received messages of all corrupted parties and let $\textsc{output}^\pi(\hat{x}, \sigma)$ contain all parties' outputs. Then, protocol π securely computes $\mathcal{F}$ if there is a probabilistic polynomial-time algorithm $\mathcal{S}$, called a simulator, s.t. for all I with $|I| < p/2$

$$\{(\mathcal{S}(1^\sigma, I, \hat{x}_I, \mathcal{F}_I(\hat{x})), \mathcal{F}(\hat{x}))\}_{\hat{x},\sigma}$$

and

$$\{(\textsc{view}_I^\pi(\hat{x}, \sigma), \textsc{output}^\pi(\hat{x}, \sigma)\}_{\hat{x},\sigma}$$

are statistically indistinguishable. Note that this definition implies that the view of the corrupted parties leaks nothing that cannot be efficiently derived from their own inputs and outputs as $\mathcal{S}$ produces statistically indistinguishable views. It also implies correctness of the protocol except for negligible probability.

2.1.2 Representation of Secret Values

We require arbitrary linear secret-sharing schemes in the arithmetic domain, i.e., for values in either $\mathbb{F}_q$ for large prime q or $\mathbb{Z}_{2^k}$ for large integer k, and in the binary domain, i.e., for values in $\mathbb{F}_2$. The binary domain is used alongside the arithmetic domain to implement non-linear functionality efficiently using edaBits [Esc+20] and daBits [RW19]. Furthermore, we require multiplication protocols for values shared using the secret-sharing schemes for the arithmetic and binary domains. Linear operations and the multiplication protocol are required to provide passive security given an honest majority. Our requirements are met, e.g., by the BGW-protocol [BGW88] or the arithmetic three-party protocol by Araki et al. [Ara+16] for the arithmetic domain, and by a variant of the BGW-protocol on $\mathbb{F}_{2^k}$ [CCD88] or the binary three-party protocol by Araki et al. [Ara+16] for the binary domain.

The linearity of the used secret-sharing schemes allows computing any linear combination of secret-shared values in the same domain using no communication. Communication is only required for multiplications and parties sending a share of some secret-shared value to each other party to share or reveal a private value. Similar to Catrina and de Hoogh [CH10] we use the invocations of such primitives that require communication as a metric for communication complexity. Note that an invocation can also be described as one atomic interactive operation. Thus, the number of invocations yields the communication complexity. As we work on two domains, we distinguish between arithmetic and binary invocations. Multiple invocations can be executed in parallel in a single round if none of them depends on the result of another one. In this context, parallelization refers to executing the required communication in a joined manner instead of sequentially executing all invocations to minimize the impact of network latency.

For a secret-shared value x we write $[x]$. Addition and multiplication operators are overloaded allowing to write terms like $[x] + c \cdot [y] + [x] \cdot [y]$ representing the corresponding operations on secret-shared values for more concise notation. In the following, let notation $[x]$ always correspond to the arithmetic domain if not explicitly stated otherwise.

Finally, we restrict our values in the arithmetic domain to the interval $\{-2^{b-1}, ..., 2^{b-1}-1\}$ for a fixed number of bits b. This later is required to enable secure comparison between secret-shared values and expects $\mathbb{Z}_p$ or $\mathbb{Z}_{2^k}$ to be sufficiently large w.r.t. b and the statistical security parameter σ [Esc+20]. A secret integer x is represented by $x \bmod p$ respectively $x \bmod 2^k$ in the chosen arithmetic domain [CH10; Esc+20]. The previously discussed arithmetic operations can be extended to secret-shared signed integers by running them on their corresponding representations [CH10; Esc+20].

2.1.3 Building Blocks

We use a MUX gate where one of two secret-shared values $[x], [y]$ can be selected depending on an additional secret-shared bit $[b]$. We use notation $[b] \mathbin{?} [x] : [y]$ which evaluates to $[x]$ if $b = 1$ and $[y]$ otherwise. Recall that by an invocation we denote an atomic interactive operation in the arithmetic or binary domain. It is easy to see that said functionality can be implemented as $([x] - [y]) \cdot [b] + [y]$ requiring a single arithmetic invocation. For public values x and y, $[b] \mathbin{?} x : y := (x - y) \cdot [b] + y$ requires no communication.

For two secret-shared vectors $[x_i]_{0 \leq i < n}, [y_i]_{0 \leq i < n}$ we write $[x_i]_{0 \leq i < n} \cdot [y_i]_{0 \leq i < n}$ to denote their dot product. This can trivially be computed using n multiplications in a single round. Note that for specific secret-sharing schemes, significantly more efficient variants may exist. For instance, for Shamir's secret-sharing scheme [Sha79] dot products can be computed using a single arithmetic invocation [CH10].

Escudero et al. [Esc+20] provide a comparison gate LTZ that given a secret-shared signed b bit integer $[x]$ as input returns $[1]$ if $x < 0$ and $[0]$ otherwise. Note that this gate extends one by Catrina and de Hoogh [CH10] by using edaBits and daBits. We also require a comparison gate EQZ that given input $[x]$ returns $[1]$ if $x = 0$ and $[0]$ otherwise. This can be constructed by applying edaBits and daBits to Catrina and de Hoogh's [CH10] equality gate in its variant using their gate KOrL. Gates LTZ and EQZ require $\mathcal{O}(1)$ arithmetic invocations, $\mathcal{O}(b)$ binary invocations and $\mathcal{O}(\log b)$ rounds. Furthermore, a constant number of edaBits and daBits is required. Regarding the efficient generation of edaBits and daBits, we refer to [Esc+20; RW19].

We also require a gate DEMUX^n that given input $[x]$ where n is the maximum value of x, i.e., $0 \leq x \leq n$ returns $[y_i]_{0 \leq i < n}$ s.t. for all $0 \leq i < n$, $[y_i] = [1]$ if $i = x$ and $[y_i] = [0]$ otherwise.[1] First, an edaBit can be used to compute a bit-decomposition of $[x]$ in the binary domain [Esc+20]. Then, the DEMUX protocol by Launchbury et al. [Lau+12] can be applied to the bit-decomposition to compute the indicator vector in the binary domain.[2] The entries of the vector can then be converted back to the arithmetic domain using a daBit for each entry [RW19]. Gate DEMUX^n requires $\mathcal{O}(1)$ arithmetic invocations, $\mathcal{O}(n)$ binary invocations, $\mathcal{O}(\log \log n)$ rounds, $\mathcal{O}(1)$ edaBits and $\mathcal{O}(n)$ daBits.

Finally, we need a gate RND_BIT that has no input and returns a secret-shared bit in the arithmetic domain that is chosen uniformly at random from $\{0, 1\}$. For the domain $\mathbb{F}_q$ such gate requiring constant arithmetic invocations and rounds exists as shown by Damgård et al. [Dam+06]. For $\mathbb{Z}_{2^k}$, similar results exist, e.g., for SPD$\mathbb{Z}_{2^k}$ [Cra+18; Dam+19].

2.2 Graph Theory

In this work, we only consider graphs that are weighted, finite, and undirected. Furthermore, we assume all edge weights to be positive integers. Without loss of generality, such a graph can be represented as $G := (V, E, \omega)$ with a set of vertices $V := \{0, ..., N-1\}$ for some $N \in \mathbb{N}$, a set of undirected edges $E \subseteq \{\{u, v\} | u, v \in V \wedge u \neq v\}$, and edge weights $\omega : E \to \mathbb{N}$. Let M be the number

[1] We include n in the input domain as one option to obtain a vector only containing $[0]$ which will become useful later.

[2] The resulting vector is padded to a dimension being a power of two but the padding can simply be discarded without changing asymptotic complexity.

of edges. For $E' \subseteq E$ we define $\omega(E') := \sum_{e \in E'} \omega(e)$. A matching $\mathfrak{M} \subseteq E$ on G is a set of edges s.t. all edges are disjoint, i.e., $e \cap f = \emptyset$ for all $e \neq f \in \mathfrak{M}$. Then, an MWM is a matching that has maximal $\omega(\mathfrak{M})$ among all matchings. As a $c-$approximation for MWM, we define an algorithm that on each graph with an MWM $\mathfrak{M}^*$ computes a matching $\mathfrak{M}$ s.t. $\omega(\mathfrak{M}) \geq c \cdot \omega(\mathfrak{M}^*)$, i.e., that has at least fraction c of the maximally possible weight.

A graph $G := (V, E, \omega)$ can be represented by an adjacency matrix $(\text{weight}_{u,v})_{u,v \in V}$ where $\text{weight}_{u,v} = \omega(\{u, v\})$ if $\{u, v\} \in E$ and $\text{weight}_{u,v} = 0$ otherwise for all $u, v \in V$. Note that the matrix is symmetrical due to the graph being undirected and its diagonal are all zeros. Thus, it is sufficient to use an upper triangular adjacency matrix $(\text{weight}_{u,v})_{0 \leq u < v < N}$. In the following, we call each pair of different nodes $\{u, v\}$ a potential edge and differentiate actual edges by calling them real edges. We denote the set of potential edges by $\hat{E}$ and its cardinality by $\hat{M} := |\hat{E}| = \binom{N}{2} \in \mathcal{O}(N^2)$.

3 Related Work

We identify two main research directions that are related to our work. (1) Non-privacy-preserving approximation algorithms for MWM on general graphs. (2) Existing privacy-preserving protocols for matching problems that are related to MWM on general graphs.

3.1 Approximation Algorithms for Maximum Weight Matching

Two well-known polynomial time algorithms for MWM on general graphs are Edmond's blossom algorithm [Edm65] with run time $\mathcal{O}(MN^2)$ and its optimization by Gabow [Gab90] with run time $\mathcal{O}(MN + N^2 \log N)$. There also exist further optimizations that require additional constraints (e.g., [GT91; HK12; Pet12]). Due to the exact algorithms' high complexity, multiple approximations have been developed to further increase scalability if exact solutions are not essential (e.g., [Pre99; DH03; DH05; PS04; HH10; DP14]).

A trivial $1/2-$approximation for MWM is a greedy algorithm that has been mentioned by Avis [Avi83] and Preis [Pre99] (cf. Algorithm 1). It can be implemented to run in time $\mathcal{O}(M \log N)$ [Pre99] and has a particularly easy structure. In Section 4 we show that due to its structure it is suitable as foundation of our protocol requiring the execution of only $\mathcal{O}(N^3)$ operations in a secure setting.

Algorithm 1: Greedy MWM Approximation [Pre99]

```
Input  : Weighted graph $G = (V, E, \omega)$
Output: Matching $\mathfrak{M}$ on $G$
1 $\mathfrak{M} \leftarrow \emptyset$;
2 while $E \neq \emptyset$ do
3     Let $e \in E$ be an edge of maximal weight;
4     $\mathfrak{M} \leftarrow \mathfrak{M} \cup \{e\}$ ;                         // add edge $e$ to $\mathfrak{M}$
5     $E \leftarrow E \setminus \{f \in E \mid e \cap f \neq \emptyset\}$ ;        // del. incident edges
6 end
```

Further $1/2-$approximations [Pre99; DH03] have a lower time complexity of $\mathcal{O}(M)$. The approximation by Preis [Pre99] uses multiple nested loops, recursive calls and conditional blocks that depend on the input. As a secure protocol must not leak private knowledge by its control flow, this approximation would require significant changes to be used as foundation of such a protocol. Most likely these changes would nullify the run time advantage the approximation has compared to the greedy algorithm. The approximation by Drake and Hougardy [DH03] utilizes two nested loops where each could run up to $\mathcal{O}(N)$ times. To not leak private knowledge, a secure protocol would need to execute both loops for the maximally required iterations yielding $\mathcal{O}(N^2)$ iterations already. As each iteration searches for a heaviest edge adjacent to a given node, a secure protocol's complexity would increase to at least $\mathcal{O}(N^3)$. Thus, there is no advantage of using it instead of the greedy approach.

There also exist $(2/3 - \epsilon)-$approximations for arbitrary $\epsilon > 0$ that run in $\mathcal{O}(1/\epsilon \cdot M)$ [DH05] and $\mathcal{O}(\log(1/\epsilon) \cdot M)$ [PS04]. They are based on sophisticated approaches to find paths to augment intermediate matchings in an efficient manner. These techniques appear not to be well suited as basis for an efficient secure protocol.

Further $(3/4-\epsilon)-$ and $(4/5-\epsilon)-$approximations exist [HH10] but have a higher run time and more complex definitions that render them unsuitable for maximizing scalability. Duan and Pettie [DP14] have shown that arbitrarily good approximations that even run in linear time w.r.t. M exist, but the general complexity of their approach makes it appear unfeasible to derive an efficient secure protocol. Furthermore, the sophisticated techniques used in these advanced approaches may hide significant constant factors regarding time complexity.

3.2 Secure Protocols for Matching Problems

To the best of our knowledge, to date there is no privacy-preserving protocol for MWM on general graphs. However, the protocol of Breuer et al. [BMW22] computes an unweighted maximum matching on a general graph based on an algorithm that is specifically designed for the unweighted case. While the protocol guarantees optimality of the computed matching, it has a complexity of $\mathcal{O}(N^4)$ rounds and $\mathcal{O}(N^5)$ communication for N nodes. This results in its run time exceeding one day for $N = 40$. In contrast to this, our protocol only computes a $1/2-$approximative solution but exhibits far lower complexities, i.e., $\mathcal{O}(N \log^2 N)$ rounds and $\mathcal{O}(N^3 \log N)$ communication (cf. Section 4.3) or $\mathcal{O}(N \log N)$ rounds and $\mathcal{O}(N^3)$ communication (cf. Section 4.4). Furthermore, we consider the more general problem of maximum *weight* matching on general graphs whereas the protocol of Breuer et al. [BMW22] only operates on unweighted graphs.

A related problem to maximum matching on general graphs is maximum matching on bipartite graphs, where the graph is partitioned into two sets of nodes s.t. there are no edges that connect two nodes of the same set. Several privacy-preserving protocols for maximum (weight) matching on such bipartite graphs have already been proposed (e.g., [AC17; BS15; Wül+17]) for different use cases such as multi-party bartering [Wül+17] or secure fingerprint identification [BS15].

A special use case for secure bipartite matching is private stable matching where each node states its preferences over the nodes in the other set. The goal is then to compute a matching between the nodes s.t. there are no pairs of nodes from the two groups who would prefer each other over their computed match. There exist multiple secure protocols [Gol06; DE16; Ria+17] for this use case that ensure that the preferences of the nodes are kept private.

However, none of these existing protocols for bipartite matching can act as the foundation for our protocol as they heavily rely on the graphs being bipartite whereas our goal is to devise a protocol that computes a matching on general graphs.

Finally, Araki et al. [Ara+21] propose a highly scalable protocol for graph analysis. While they show how to apply it to a selection of graph problems, they depend on existing message-passing algorithms for these problems. As it is unclear how to base a message-passing algorithm on any of the discussed matching algorithms, and especially the greedy approximation (cf. Algorithm 2), their protocol appears not to be applicable here.

4 Privacy-Preserving Maximum Weight Matching Approximation

We base our protocol for MWM approximation on the greedy MWM approximation algorithm documented by Avis [Avi83] and Preis [Pre99] (cf. Algorithm 1) due to its low concrete run time complexity to maximize scalability. Note that the algorithm specifies that in each iteration an edge of maximal weight is chosen but does not specify which edge to choose if there are multiple options. Thus, we start by defining an unambiguous version.

Let $\prec_d$ be defined as the following ordering of potential edges for a graph with N nodes:

$$\{0,1\} \prec_d \{0,2\} \prec_d ... \prec_d \{0,N-1\} \prec_d \{1,2\} \prec_d \{1,3\} \prec_d ...$$

Choosing an edge of maximal weight that is minimal w.r.t. $\prec_d$ then is sufficient to unambiguously give the deterministic greedy MWM approximation algorithm in Algorithm 2.

Algorithm 2: Deterministic Greedy MWM Approximation (based on [Pre99])

```
  Input  : Weighted graph $G = (V, E, \omega)$
  Output: Matching $\mathfrak{M}$ on $G$
1 $\mathfrak{M} \leftarrow \emptyset$;
2 while $E \neq \emptyset$ do
3     Let $E_{\max} \subseteq E$ exactly contain all edges of maximal weight;
4     Let $e \in E_{\max}$ be the minimal edge in $E_{\max}$ w.r.t. $\prec_d$;
5     $\mathfrak{M} \leftarrow \mathfrak{M} \cup \{e\}$ ;                          // add edge $e$ to $\mathfrak{M}$
6     $E \leftarrow E \setminus \{f \in E \mid e \cap f \neq \emptyset\}$ ;      // del.  incident edges
7 end
```

Let the input of our protocol for arbitrary but fixed number of nodes N be a secret-shared adjacency matrix $[\text{weight}_{u,v}]_{0 \le u<v<N}$. As no information about the edges besides the output may be leaked, our protocol works on potential edges, i.e., pairs of nodes $u \neq v$. By the definition of the adjacency matrix, potential edges $\{u, v\}$ that are no real edges can be identified by $[\text{weight}_{u,v}]$ being $[0]$.

We proceed by giving a building block for our protocol that can be used to efficiently search for heaviest edges in Section 4.1. Then, we describe our basis protocol that implements the functionality computed by Algorithm 2 in Section 4.2. Finally, we propose two ideas on randomizing functionality and protocol and briefly discuss their implementation in Sections 4.3 and 4.4 yielding the two variants of our protocol.

4.1 Heaviest Edge Search

A central part of our protocol is to efficiently search for an edge of maximal weight to be added to the matching in each iteration. Our desired functionality takes a collection of $n \in \mathbb{N}$ potential edges $\{[\text{edge\_u}_i], [\text{edge\_v}_i]\}$ with weights $[\text{edge\_weight}_i]$ for $0 \le i < n$ as input. Let $k \in \{0, ..., n-1\}$ be minimal s.t. edge_weight_k is maximal. Then we define $([\text{edge\_u}_k], [\text{edge\_v}_k])$ as the output if $\text{edge\_weight}_k > 0$. If $\text{edge\_weight}_k = 0$, there exists no real edge and we define $([N], [N])$ as the output where it holds that $N \notin V$.

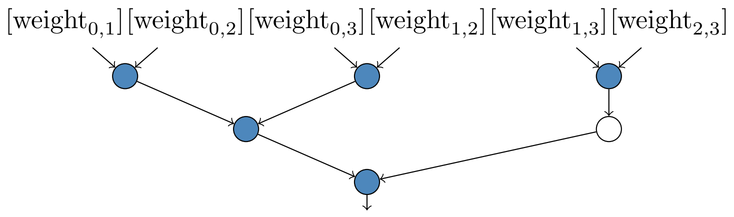

Figure 1: Structuring of potential edge weight comparisons for $N = 4$. Comparisons are performed in filled nodes only.

The naïve approach to realize given functionality is to run a linear search over all $\hat{M} = (N^2 - N)/2$ potential edges which requires $\hat{M} - 1 \in \mathcal{O}(N^2)$ comparisons run in sequential order. While no approach based on binary comparisons that uses less comparisons exists, multiple comparisons can be executed in parallel to reduce the number of required communication rounds. Hence, we execute comparisons as reduction operators on a binary tree as is depicted in Figure 1.

If on some layer $n > 1$ potential edges are being compared, the next layer handles $\lceil n/2 \rceil$ potential edges. Thus, $\mathcal{O}(\log \hat{M}) = \mathcal{O}(\log N)$ layers are required where the comparisons on a single layer can be executed in parallel.

To realize the functionality using the given binary tree structure, we use a recursive approach as given in the gate specification in Algorithm 3. If only one edge is given, its encoding is returned if and only if it is a real edge, i.e., its weight is greater than zero. Otherwise, $([N], [N])$ is returned as it is required by the functionality. For multiple edges, pairs of edges are compared. When a pair consisting of edges $2i$ and $2i + 1$ for appropriate i is compared the first one proceeds to the next layer if and only if $1 - \mathrm{LTZ}([weight_{2i}] - [weight_{2i+1}])$ evaluates to $[1]$, i.e., $weight_{2i} \geq weight_{2i+1}$ (line 11 in Algorithm 3). Note that this way edges with lower index are preferred if the weight is equal as required by the functionality's definition.

Algorithm 3: Heaviest Edge Search

```
1  Gate HEAVIEST$^n$([edge_u$_i$]$_{0\le i<n}$, [edge_v$_i$]$_{0\le i<n}$, [edge_weight$_i$]$_{0\le i<n}$):
       Input  : Collection of potential edges {edge_u$_i$, edge_v$_i$} with weights edge_weight$_i$ for
                $0 \le i < n$
       Output: Edge ([edge_u$_i$], [edge_v$_i$]) with minimal $i$ where weight edge_weight$_i > 0$ is
                maximal or $([N],[N])$ if edge_weight$_i = 0$ for all $0 \le i < n$
2      if $n = 1$ then
           // Check if heaviest edge is a real edge; return $([N],[N])$ otherwise
3          [invalid] ← EQZ(edge_weight$_0$);
4          $[a]$ ← [invalid] ? $[N]$ : [edge_u$_0$];
5          $[b]$ ← [invalid] ? $[N]$ : [edge_v$_0$];
6          return $([a],[b])$
7      else
8          $n' \leftarrow \lceil n/2 \rceil$;
9          pairs ← $\lfloor n/2 \rfloor$;
           // Compare pairs of edges
10         for $i \leftarrow 0$ to pairs $- 1$ in parallel do
11             [first] ← $1 - \mathrm{LTZ}$([edge_weight$_{2i}$] − [edge_weight$_{2i+1}$]);
12             [edge_u$'_i$] ← [first] ? [edge_u$_{2i}$] : [edge_u$_{2i+1}$];
13             [edge_v$'_i$] ← [first] ? [edge_v$_{2i}$] : [edge_v$_{2i+1}$];
14             [edge_weight$'_i$] ← [first] ? [edge_weight$_{2i}$] : [edge_weight$_{2i+1}$];
15         end
           // Handle last edge if number of edges is odd
16         if $n' >$ pairs then
17             [edge_u$'_{n'-1}$] ← [edge_u$_{n-1}$];
18             [edge_v$'_{n'-1}$] ← [edge_v$_{n-1}$];
19             [edge_weight$'_{n'-1}$] ← [edge_weight$_{n-1}$];
20         end
           // Recursive call
21         return HEAVIEST$^{n'}$([edge_u$'_i$]$_{0\le i<n'}$, [edge_v$'_i$]$_{0\le i<n'}$, [edge_weight$'_i$]$_{0\le i<n'}$)
22     end
```

4.1.1 Correctness and Security

It is easy to see that for $n = 1$, the gate implements the previously described functionality correctly. For arbitrary $n > 1$, let k be minimal s.t. edge_weight_k is maximal. The potential edge with index k is an input to $\text{HEAVIEST}^{\lceil n/2 \rceil}$ as it is compared to at most one edge and by definition of k has higher weight or equal weight and lower index. It also holds that multiple edges proceeding to the next layer do not change their order. Thus, gate HEAVIEST^n implements the functionality correctly for arbitrary integer $n \geq 1$ by induction.

All used gates and arithmetic primitives are secure. Thus, gate HEAVIEST^n implements the functionality securely as it can be simulated by calling the simulators for the used operations on private data following the publicly known flow of execution.

4.1.2 Complexity

Regarding the gate's complexity, it is easy to see that the complexity for $n = 1$ is $\mathcal{O}(1)$ arithmetic invocations and $\mathcal{O}(b)$ binary invocations in $\mathcal{O}(\log b)$ rounds, and that $\mathcal{O}(1)$ edaBits and daBits are required. For arbitrary $n > 1$, the complexity excluding the recursive call is $\mathcal{O}(n)$ arithmetic invocations, $\mathcal{O}(nb)$ binary invocations, $\mathcal{O}(\log b)$ rounds, and $\mathcal{O}(n)$ edaBits and daBits. Thus, the total complexity including the recursive call is $\mathcal{O}(n)$ arithmetic invocations, $\mathcal{O}(nb)$ binary invocations, $\mathcal{O}(\log n \cdot \log b)$ rounds, and $\mathcal{O}(n)$ edaBits and daBits.

4.1.3 Optimization of the First Layer

If HEAVIEST^n is called for a collection of publicly known potential edges in publicly known order but with private weights, edge_u_i and edge_v_i for $0 \leq i < n$ do not have to be secret-shared. Thus, lines 12 f. in Algorithm 3 only require a multiplication of a secret-shared and a constant value reducing the amount of required communication. Note that this optimization does not change the gate's asymptotic complexity. We denote the resulting modified gate by $\text{HEAVIEST}^n_{\text{CLEAR}}$. Note that it still recursively calls $\text{HEAVIEST}^{\lceil n/2 \rceil}$ as the selection of potential edges being the input to the recursive call depends on the weights and thus is private. It is easy to see that the modified gate implements a modification of former functionality where public inputs $\text{edge\_u}_i, \text{edge\_v}_i$ for $0 \leq i < n$ are used correctly and securely.

4.2 Deterministic MWM Approximation

Recall that Algorithm 2 executes multiple iterations until no edges remain where in each iteration it

1. selects the minimal edge w.r.t. $\prec_d$ among all edges of maximal weight (Section 4.2.1),
2. adds the selected edge to the matching (Section 4.2.2),
3. and deletes all edges that are incident to the selected one (Section 4.2.3).

For our approach, a fixed number of iterations is necessary as only running iterations until no edges remain would leak the size of the resulting matching. Thus, our protocol executes $\lfloor N/2 \rfloor$ iterations which is the maximal possible size of a matching between N nodes. If no real edges remain, further iterations do not extend the matching any longer.

The resulting base protocol is given in Algorithm 4. In the following, we explain how the different steps of each iteration are implemented. Then, we conclude correctness and security in Section 4.2.4 and the overall complexity in Section 4.2.5.

Algorithm 4: Secure Deterministic MWM Approximation

```
Input  : Weighted graph $G$ encoded by $[\text{weight}_{u,v}]$ for $0 \le u < v < N$
Output: Matching $\mathfrak{M}$ on $G$, encoded by $[\text{partner}_u]$ for $u \in V$ s.t. $\text{partner}_u = v$ if
        $\exists v \in V : \{u, v\} \in \mathfrak{M}$, $\text{partner}_u = N$ otherwise
1  $[\text{partner}_u] \leftarrow [N]$ for $u \in V$;
2  for $0 \le u < v < N$ do
3      $e \leftarrow \text{edge\_index}(u, v)$;
4      $\text{edge\_u}_e \leftarrow u$;
5      $\text{edge\_v}_e \leftarrow v$;
6      $[\text{edge\_weight}_e] \leftarrow [\text{weight}_{u,v}]$;
7  end
8  for $\lfloor N/2 \rfloor$iterations do
       // Search for heaviest edge
9      $([a], [b]) \leftarrow \text{HEAVIEST}^{\hat{M}}_{\text{CLEAR}}((\text{edge\_u}_e)_{0 \le e < \hat{M}}, (\text{edge\_v}_e)_{0 \le e < \hat{M}}, [\text{edge\_weight}_e]_{0 \le e < \hat{M}})$;
10     $[\text{indicator\_a}_u]_{u \in V} \leftarrow \text{DEMUX}^N([a])$;
11     $[\text{indicator\_b}_u]_{u \in V} \leftarrow \text{DEMUX}^N([b])$;
       // Add selected edge to the matching
12     $[\text{partner}_u]_{u \in V} \leftarrow [\text{partner}_u]_{u \in V} + ([b] - N) \cdot [\text{indicator\_a}_u]_{u \in V} + ([a] - N) \cdot [\text{indicator\_b}_u]_{u \in V}$;
13     if not last iteration then
           // Delete blocked edges
14         $[\text{indicator}_u]_{u \in V} \leftarrow [\text{indicator\_a}_u]_{u \in V} + [\text{indicator\_b}_u]_{u \in V}$;
15         for $0 \le u < v < N$ in parallel do
16             $e \leftarrow \text{edge\_index}(u, v)$;
17             $[\text{edge\_weight}_e] \leftarrow [\text{edge\_weight}_e] \cdot (1 - [\text{indicator}_u]) \cdot (1 - [\text{indicator}_v])$;
18         end
19     end
20 end
```

4.2.1 Selecting an Edge

Recall that in each iteration of Algorithm 4, the minimal edge w.r.t. $\prec_d$ among all edges of maximal weight is to be selected. Gate $\text{HEAVIEST}^{\hat{M}}_{\text{CLEAR}}$ gets a collection of potential edges as input and returns the edge of minimal index among all edges of maximal weight if at least one real edge exists. Thus, ordering all edges w.r.t. $\prec_d$ leads to $\text{HEAVIEST}^{\hat{M}}_{\text{CLEAR}}$ yielding the desired output if there still are real edges.

We use a public function edge_index(u, v) that maps two nodes $u < v$ to the index of the corresponding potential edge w.r.t. $\prec_d$, i.e., $\text{edge\_index}(0, 1) = 0, \text{edge\_index}(0, 2) = 1, ..., \text{edge\_index}(0, N-1) = N - 2, \text{edge\_index}(1, 2) = N - 1, ...$ Thus, for two potential edges $\{u, v\}$ and $\{w, x\}$ it holds that $\{u, v\} \prec_d \{w, x\}$ if and only if $\text{edge\_index}(u, v) < \text{edge\_index}(w, x)$. Note that using gate $\text{HEAVIEST}^{\hat{M}}_{\text{CLEAR}}$ instead of gate $\text{HEAVIEST}^{\hat{M}}$ is valid because mapping edge_index(u, v) is public.

Before the first iteration is executed, all potential edges are ordered by $\prec_d$ in lines 2 ff. which does not require any communication. In each iteration, the ordered edges then are used as input to $\text{HEAVIEST}^{\hat{M}}_{\text{CLEAR}}$ in lines 9 ff. The complexity of selecting an edge hence is $\mathcal{O}(N^2)$ arithmetic invocations, $\mathcal{O}(N^2 b)$ binary invocations, $\mathcal{O}(\log N \cdot \log b)$ rounds, and $\mathcal{O}(N^2)$ edaBits and daBits (cf. Section 4.1.2).

4.2.2 Adding an Edge to the Matching

We represent each edge $\{u, v\}$ in the matching by setting $[\text{partner}_u]$ to $[v]$ and $[\text{partner}_v]$ to $[u]$ in Algorithm 4. By $[\text{partner}_u] = [N]$ we encode that node u is matched to no other node. Thus, all secret-shared values $[\text{partner}_u]$ are initialized to $[N]$.

If a real edge $\{a, b\}$ is chosen in some iteration, it is not possible just to access $[\text{partner}_a]$ and $[\text{partner}_b]$ as this would leak the chosen edge. Instead, we use gate DEMUX^N to generate two indicator vectors that contain value [1] at index a respectively b and value [0] at other indexes. If edge selection and deletion work correctly, a and b are not matched at the start of the iteration where they are selected implying that $[\text{partner}_a] = [\text{partner}_b] = [N]$. Additionally, it holds that $a \neq b$. Then, lines 12-12 update $[\text{partner}_u]$ for $u \in V$ to:

$$\begin{aligned}
&[\text{partner}_u] + ([b] - N) \cdot [\text{indicator\_a}_u] \\
&\qquad + ([a] - N) \cdot [\text{indicator\_b}_u] \\
&= \left\{ \begin{array}{ll} [N] + ([b] - N) \cdot [1] + ([a] - N) \cdot [0], & \text{if } u = a, \\ [N] + ([b] - N) \cdot [0] + ([a] - N) \cdot [1], & \text{if } u = b, \\ [\text{partner}_u] + ([b] - N) \cdot [0] + ([a] - N) \cdot [0], & \text{if } u \notin \{a, b\} \end{array} \right\} \\
&= \begin{cases} [b], & \text{if } u = a, \\ [a], & \text{if } u = b, \\ [\text{partner}_v], & \text{if } u \notin \{a, b\}. \end{cases}
\end{aligned}$$

Thus, $[\text{partner}_u]_{u \in V}$ is correctly updated if a real edge $\{a, b\}$ is chosen. Otherwise, no real edges remain and $[a] = [b] = [N]$ are returned by $\text{HEAVIEST}^{\hat{M}}_{\text{CLEAR}}$. In this case, each indicator entry is [0] so that $[\text{partner}_u]_{u \in V}$ keeps unchanged.

Generating the two indicator vectors requires $\mathcal{O}(1)$ arithmetic invocations, $\mathcal{O}(N)$ binary invocations, $\mathcal{O}(\log \log N)$ rounds, $\mathcal{O}(1)$ edaBits and $\mathcal{O}(N)$ daBits. The subsequent operation on vectors takes $\mathcal{O}(1)$ arithmetic invocations per index and therefore a total of $\mathcal{O}(N)$ arithmetic invocations which can be parallelized into a single round. Thus, the total complexity consists of $\mathcal{O}(N)$ arithmetic and binary invocations, $\mathcal{O}(\log \log N)$ rounds, $\mathcal{O}(1)$ edaBits, and $\mathcal{O}(N)$ daBits.

4.2.3 Deleting Incident Edges

Deleting all edges incident to a real edge $\{a, b\}$ in Algorithm 4 again requires to access all edges in order to not leak any information about $\{a, b\}$. In the used representation it is sufficient to set the weights of all edges to delete to zero. This is achieved by lines 13-19. Note that as a small optimization this deletion procedure is not executed in the last iteration as the potential edge weights are never queried afterwards.

Both indicator vectors from the previous step in Section 4.2.2 are added together yielding a vector $[\text{indicator}_u]_{u \in V}$ that contains value [1] at indexes a and b and value [0] at other indexes. Then, the weight of each potential edge $\{u, v\}$ with corresponding index e is set to

$$\begin{aligned}
&[\text{edge\_weight}_e] \cdot (1 - [\text{indicator}_u]) \cdot (1 - [\text{indicator}_v]) \\
&= \left\{ \begin{array}{ll} [\text{edge\_weight}_e] \cdot [1] \cdot [1], & \text{if } u, v \notin \{a, b\}, \\ [\text{edge\_weight}_e] \cdot [1] \cdot [0], & \text{if } u \notin \{a, b\} \wedge v \in \{a, b\}, \\ [\text{edge\_weight}_e] \cdot [0] \cdot [1], & \text{if } u \in \{a, b\} \wedge v \notin \{a, b\}, \\ [\text{edge\_weight}_e] \cdot [0] \cdot [0], & \text{if } u, v \in \{a, b\} \end{array} \right\} \\
&= \begin{cases} [\text{edge\_weight}_e], & \text{if } \{u, v\} \cap \{a, b\} = \emptyset, \\ [0], & \text{if } \{u, v\} \cap \{a, b\} \neq \emptyset. \end{cases}
\end{aligned}$$

Thus, all incident edges are correctly deleted if a real edge $\{a, b\}$ is added to the matching. If no real edge is found it holds that $[a] = [b] = [N]$ so that each indicator entry is $[0]$ and no weights are changed.

Both formerly computed indicator vectors can be added without communication. Updating all edge weights takes $\mathcal{O}(N^2)$ arithmetic invocations in $\mathcal{O}(1)$ rounds.

4.2.4 Correctness and Security

By the correctness of the gate $\text{HEAVIEST}_{\text{CLEAR}}$ and the considerations before, it is easy to see that our protocol given in Algorithm 4 would implement Algorithm 2 correctly if it terminated as soon as no edges remain. If no edges remain, $\text{HEAVIEST}_{\text{CLEAR}}^{\hat{M}}$ returns $([N], [N])$. As previously discussed, neither weights nor the encoding of the resulting matching are altered in this case. Thus, the protocol given in Algorithm 4 implements Algorithm 2 correctly.

All used primitives and gates are secure. Thus, the protocol can be simulated by following the flow of execution and calling the corresponding simulator for each operation on secret data.

4.2.5 Complexity

Recall that by N, we denote the number of nodes and by b, we denote the number of bits for secret-shared integers. Summing up the complexities for each single phase of an iteration yields a complexity of $\mathcal{O}(N^2)$ arithmetic invocations, $\mathcal{O}(N^2 b)$ binary invocations, $\mathcal{O}(\log N \cdot \log b)$ rounds, and $\mathcal{O}(N^2)$ edaBits and daBits. Thus, the total complexity is $\mathcal{O}(N^3)$ arithmetic invocations, $\mathcal{O}(N^3 b)$ binary invocations, $\mathcal{O}(N \log N \cdot \log b)$ rounds, and $\mathcal{O}(N^3)$ edaBits and daBits.

4.3 Random Edge Selection

Recall that to get an unambiguous version of the original greedy MWM approximation algorithm (cf. Algorithm 1) we introduced an ordering $\prec_d$ on all potential edges to decide which edge to match if multiple edges of the same maximal weight exist. Thus, the static ordering $\prec_d$ induces advantages in getting matched for some edges compared to other edges of the same weight. Depending on the specific use case, this may be an undesired bias, especially if entities modeled by nodes or edges compete in getting matched or selected. Ideally, no entity should have an unfair disadvantage solely resulting from the arbitrary usage of ordering $\prec_d$.

A natural way to tackle this problem is to select an edge among all edges of maximal weight $E_{\max}$ (cf. Algorithm 2) uniformly at random in each iteration. In the following, we discuss and sketch how such a random edge selection can be implemented in a secure manner. We start by showing how a single randomly chosen ordering can be used to realize the desired functionality in Section 4.3.1. Then, we proceed by discussing why such an ordering has to be kept private in Section 4.3.2 and modify our protocol from Algorithm 4 accordingly in Section 4.3.3.

4.3.1 Using Random Orderings

In the following, by O_X we denote the set of all strict total orderings on X. We propose to replace the deterministic edge ordering $\prec_d$ in Algorithm 2 by an ordering $\prec_r$ drawn uniformly at random from all strict total orderings on the potential edges, i.e., $O_{\hat{E}}$. This approach only requires small changes in our protocol and is equivalent to the previously discussed random edge selection:

Proposition 1. *Drawing an ordering $\prec_r$ uniformly at random from $O_{\hat{E}}$ and using it instead of $\prec_d$ in Algorithm 2 is equivalent to selecting an edge uniformly at random from all edges of maximal weight in each iteration.*

Proof. Let Algorithm 2 be in some fixed iteration i in which the set of all still existing real edges of maximal weight is $E_{\max}$. Note that $E_{\max}$ may depend on the ordering $\prec_r$. Furthermore, let $\prec'_r \in O_{E_{\max}}$

be arbitrary but fixed. The probability that $\prec'_r$ on $E_{\max}$ is induced by the initially randomly drawn $\prec_r$ in iteration i is

$$\mathbb{P}(\prec_r \text{ induces } \prec'_r | E_{\max} \text{ set of max. weight edges in iteration } i).$$

We claim that $\prec_r$ inducing $\prec'_r$ and $E_{\max}$ being the set of maximum weight edges in iteration i are statistically independent events. Let E' be the set of all edges that have equal weight as the edges in $E_{\max}$. Thus, previous iterations have removed edges from E' to eventually obtain $E_{\max}$. Let e be such a removed edge.

Edge e may have been removed in an iteration where it was added to the matching. This implies that it was minimal w.r.t. $\prec_r$ among all possible candidates which is statistically independent of the ordering of the remaining candidates. Thus, this event is also independent of $\prec'_r$. The same iteration may also have removed edge $e' \in E'$ due to being incident to e which trivially also is statistically independent of $\prec'_r$. The remaining case is that e has been removed in an iteration where an edge of higher weight was added to the matching. As such step only considers the ordering of higher weight edges, the deletion of e also is statistically independent of $\prec'_r$.

Thus, the probability that $\prec'_r$ on $E_{\max}$ is induced by the initially randomly drawn $\prec_r$ simplifies to $\mathbb{P}(\prec_r \text{ induces } \prec'_r)$. As each possible ordering on $E_{\max}$ can be embedded in equally many orderings on $\hat{E}$, each possible $\prec'_r$ has equal possibility of being induced by $\prec_r$ drawn uniformly at random. Thus, each element of $E_{\max}$ has equal possibility of being minimal w.r.t. $\prec'_r$ and hence being added to the matching in iteration i. □

4.3.2 Public vs Private Random Ordering

Modifying our protocol to use a public randomly chosen ordering $\prec_r$ of the potential edges can be achieved easily by changing the mapping edge_index(u, v) in Algorithm 4 according to ordering $\prec_r$. We show next that this approach is not secure as it leaks information that is not leaked by the ideal functionality.

Let there be be a graph with two edges $e := \{0, 1\}$ and $f := \{1, 2\}$ and let $\mathfrak{M} := \{e\}$ be the output of a protocol implementing random edge selection using a random ordering $\prec_r$. If $\prec_r$ is private, it can only be derived that $\omega(e) > \omega(f)$ or $\omega(e) = \omega(f) \wedge e \prec_r f$. If $\prec_r$ is public, it is known whether $e \prec_r f$ or $f \prec_r e$. In the first case, it can only be derived that $\omega(e) \geq \omega(f)$. The second case allows to derive that $\omega(e) > \omega(f)$ as f would be chosen otherwise. Note that this is strictly more knowledge on the edge weights than for private $\prec_r$. Hence, it is important to keep the random ordering $\prec_r$ on the potential edges private.

4.3.3 Protocol for Random Edge Selection

One natural way of using a private random ordering $\prec_r$ of all potential edges is to privately shuffle the collection of edges generated in lines 2 ff. of Algorithm 4. Such random shuffling would directly induce a random ordering $\prec_r$. Gate HEAVIEST$^{\hat{M}}$ returning the edge of minimal index among all edges of maximal weight $E_{\max}$ would be equivalent to it returning the edge minimal w.r.t. $\prec_r$ in $E_{\max}$.[3] Note that this approach would lead to mapping edge_index(u, v) not being public. This would make significant changes especially to the edge deletion in lines 13-19 of Algorithm 4 necessary as they strongly depend on the mapping being public.

Instead, we directly modify gate HEAVIEST$^{\hat{M}}$ and its version HEAVIEST$^{\hat{M}}_{\text{CLEAR}}$ to respect a private random ordering $\prec_r$. At the start of the protocol, a random value $[r_e]$ is drawn for each potential edge $e \in \hat{E}$. To this end, gate RND_BIT is called $b - 1$ times for each $[r_e]$ to obtain bits $[b_0], ..., [b_{b-2}]$ and set $[r_e] \leftarrow \sum_{i=0}^{b-2} 2^i \cdot [b_i]$. This securely generates $[r_e]$ s.t. r_e is drawn uniformly at random from $\{0, ..., 2^{b-1} - 1\}$. Note that the restriction of the domain is required by our used

[3]Note that HEAVIEST$^{\hat{M}}_{\text{CLEAR}}$ cannot be used as the nodes of the input edges are not publicly known after private shuffling.

integer representation that is utilized by the comparison gates. The complexity of generating all $[r_e]$ is $\mathcal{O}(N^2 b)$ arithmetic invocations in $\mathcal{O}(1)$ rounds which does not alter the total asymptotic complexity of our protocol. Then, we define $\prec_r$ as

$$\prec_r := \{(e,f) \in \hat{E} \times \hat{E} | r_e < r_f\}.$$

Note that the resulting $\prec_r$ only is a strict total ordering if all drawn random numbers are distinct.

Recall that HEAVIEST$^{\hat{M}}$ and its variant select the first edge e of two selected edges e, f if and only if $\text{weight}_e \geq \text{weight}_f$ which prefers the edge of lower index if the weights are equal. Instead, we now require e to be selected if and only if $\text{weight}_e > \text{weight}_f \vee (\text{weight}_e = \text{weight}_f \wedge e \prec_r f)$ which is equivalent to $\text{weight}_e > \text{weight}_f \vee (\text{weight}_e = \text{weight}_f \wedge r_e < r_f)$. It is easy to see that this predicate can be evaluated using three comparison gates (LTZ and EQZ) and one multiplication.[4] In addition, the random values have to be carried along the computation together with the nodes and weights of each edge in HEAVIEST$^{\hat{M}}$ and its variant. The asymptotic complexity of the gates and thus also the one of the protocol remains unchanged by implementing random edge selection. Note that these gate calls contribute the highest asymptotic complexity to each of the protocol's iterations which is still increased by a constant factor by the proposed changes. Thus, an only constant but not negligible overhead is to be expected.

Finally, recall that the proposed modification depends on all random values being distinct. Increasing the number of bits b decreases the probability of a collision. Let b_0 be the minimal number of bits necessary to represent all arithmetic values handled by our protocol correctly. Note that b_0 depends on the exact range that weights are picked from. Then, our protocol works correctly except for a probability that is negligible in statistical security parameter σ if we set

$$b = \max\{b_0, \lceil \sigma/2 + 4\log_2 N \rceil\}. \tag{1}$$

Proposition 2. *Let $b = \max\{b_0, \lceil \sigma/2 + 4\log_2 N \rceil\}$. Then, all drawn random values r_e for $e \in \hat{E}$ using $b-1$ bits each are distinct except for negligible probability.*

Proof.

$$\begin{aligned}\mathbb{P}(\exists e \neq f \in \hat{E} : r_e = r_f) &\leq \frac{\hat{M}^2}{2 \cdot 2^{b-1}} = \frac{\binom{N}{2}^2}{2^{\max\{b_0, \lceil \sigma/2 + 4\log_2 N \rceil\}}} \\ &\leq \frac{N^4}{2^{\sigma/2 + 4\log_2 N}} = \frac{N^4}{2^{\sigma/2} \cdot N^4} = \frac{1}{2^{\sigma/2}}\end{aligned}$$

with the first inequation given by Katz and Lindell [KL14]. □

It is easy to see that the resulting protocol therefore is secure.

By setting b according to equation (1), the round complexity is now given by $\mathcal{O}(N \log N \cdot \log b) = \mathcal{O}(N \log N \cdot (\log N + \sigma + b_0))$ and the number of binary invocations is $\mathcal{O}(N^3 b) = \mathcal{O}(N^3(\log N + \sigma + b_0))$. This also increases the length of required edaBits. Other previously discussed complexities are independent of b.

4.4 Node Shuffling

In specific applications as online dating or multi-party bartering, different entities can be represented by nodes and compete in getting matched to other entities' nodes. Some kind of compatibility measure can be used to determine if two nodes are connected by an edge and to define edge weights. While random edge selection may not necessarily be required, none of the entities should gain any advantage from the specific node that it is represented by together with the static ordering $\prec_d$. In this case,

[4]Note that for evaluating the logical or an addition is sufficient as $\text{weight}_e > \text{weight}_f$ and $\text{weight}_e = \text{weight}_f$ are mutually exclusive.

a solution that we will later show to be more efficient than random edge selection (cf. Section 4.3) is to randomize the mapping between entities and their nodes. In the following, we generalize and formalize this approach, point out its difference to random edge selection, and then elaborate on its implementation.

4.4.1 Using Graph Isomorphisms

Let $G = (V, E, \omega)$ be a weighted graph and π be a permutation on V. In the following, by $\pi(G)$ we denote the result of using π on G as a graph isomorphism, i.e.,

$$\begin{aligned} \pi(G) &:= (V, E', \omega') \text{ where} \\ E' &:= \{\{\pi(u), \pi(v)\} | \{u, v\} \in E\} \\ \omega'(\{\pi(u), \pi(v)\}) &:= \omega(\{u, v\}) \forall \{u, v\} \in E. \end{aligned}$$

Furthermore, we define $\pi(\mathfrak{M}) := \{\{\pi(u), \pi(v)\} | \{u, v\} \in \mathfrak{M}\}$ for a matching $\mathfrak{M}$ on G. Clearly, $\pi(\mathfrak{M})$ is a matching on $\pi(G)$ of equal weight.

We define our node shuffling approach as follows: For a weighted graph G, a permutation π on its nodes is drawn uniformly at random from all such permutations and $\pi(G)$ is computed. Then, Algorithm 2 is used to deterministically compute a matching $\mathfrak{M}'$ on $\pi(G)$ using ordering $\prec_d$. Finally, $\pi^{-1}(\mathfrak{M}')$ is computed and returned. It is easy to see that this approach returns a matching on G that has at least half of the maximally possible weight. Also, the values representing the nodes of G do not play any role when computing the matching due to being randomly permuted before.

More formally, let the new approach compute a probabilistic function f mapping weighted graphs to matchings. The computed function f then is *invariant under node permutation*:

Definition 1. *A (probabilistic) function f mapping weighted graphs to matchings is invariant under node permutation if and only if for any weighted graph G and permutation τ on its nodes, the distributions induced by $f(G)$ and $\tau^{-1}(f(\tau(G)))$ are equal.*

Note that the proposed approach is more general than randomly mapping entities to nodes as it works on any graph input. Thus, we have realized an additional layer of abstraction. In use cases as described in the beginning of this section, entity i may be mapped to node i which then is mapped to a random node by permutation π.

4.4.2 Node Shuffling vs Random Edge Selection

The motivation of node shuffling specifically is to nullify any effect that the values representing the nodes of G have on the result, i.e., realize invariance under node permutation. It is easy to see that this is also achieved by the previously proposed random edge selection approach which does not use $\prec_d$ and thus does not depend on the nodes' values. In the following, we will briefly show that on the other hand, node shuffling computes a different function than the previous approach and thus does not provide random edge selection.

Proposition 3. *Using node shuffling leads to computing another functionality than when using random edge selection.*

Proof. Let a weighted graph with four nodes be given that is a path of three edges $\{0, 1\}, \{1, 2\}, \{2, 3\}$ of equal weight. In the first iteration, random edge selection picks edge $\{1, 2\}$ with probability $1/3$. The matching cannot be extended further. With probability $2/3$, the matching $\{\{0, 1\}, \{2, 3\}\}$ results where any of both edges may be selected first.

Regarding node shuffling, there are $4! = 24$ possible permutations. We observe that in the first iteration, edge $\{0, 1\}$ corresponding to edge $\{\pi(0), \pi(1)\}$ in the permuted graph is selected due to being minimal among all candidates w.r.t. $\prec_d$ if and only if $\pi(0) = 0$ or $\pi(1) = 0 \wedge \pi(0) < \pi(2)$. This holds for exactly $3! + 3!/2 = 9$ possible permutations. Thus, the edge is selected with probability $9/24 = 3/8$. The symmetrical case holds for edge $\{2, 3\}$ yielding that matching $\{\{0, 1\}, \{2, 3\}\}$ results with probability $6/8 > 2/3$. $\square$

4.4.3 Protocol for Node Shuffling

It is easy to see that given an adjacency matrix of the input graph G, a permutation can be applied to the graph by permuting first the rows and then the columns of its adjacency matrix by the same permutation. Note that this approach does not directly work on an upper triangular adjacency matrix $[\text{weight}_{u,v}]_{0\leq u<v<N}$ as used in our protocol. An easy fix is to set $[\text{weight}_{v,u}] \leftarrow [\text{weight}_{u,v}]$ for $0 \leq u < v < N$ and $[\text{weight}_{u,u}] \leftarrow [0]$ for $u \in V$ temporarily without communication.

Zahur et al. [Zah+16] demonstrate how an oblivious Waksman network [Wak68] can be used to shuffle, i.e., permute a set of items according to a permutation drawn uniformly at random. Their approach is also used in MP-SPDZ [Kel20] that we use to implement our protocol. The oblivious Waksman network consists of gadgets for oblivious swapping of two elements $[x], [y]$ depending on a control bit $[b]$. In our setting, this is possible by setting the outputs to $[x'] \leftarrow [b]\ ?\ [y] : [x]$ and $[y'] \leftarrow [x] + [y] - [x']$ using a single arithmetic invocation. Now, we let a party locally compute a random permutation and derive control bits for the Waksman network given the permutation. These control bits are secret-shared and all parties jointly execute the permutation evaluating the Waksman network using the oblivious swapping gadgets. We repeat this procedure with another party computing a random permutation until the inputs are shuffled $\lceil p/2 \rceil$ times with p being the number of parties. Thus, at least one permutation is computed by an honest party and the resulting composition of all used permutations is drawn uniformly at random from all possible permutations and unknown to any minority of parties.

The Waksman network for N inputs uses $\mathcal{O}(N \log N)$ swapping gadgets on $\mathcal{O}(\log N)$ layers. To not swap single secret-shared values but rows and columns of dimension N, we can replace each swapping gadget by N swapping gadgets used in parallel. This yields a complexity of $\mathcal{O}(N^2 \log N \cdot p)$ arithmetic invocations and $\mathcal{O}(\log N \cdot p)$ rounds. In addition, $\mathcal{O}(N \log N \cdot p)$ control bits need to be secret-shared in parallel. As they are multiplied by values in the arithmetic domain, we secret-share them in the same domain. Thus, the complexity remains as previously stated.

After permuting the adjacency matrix according to some permutation π, we use Algorithm 4 to deterministically compute a matching $\mathfrak{M}'$ on $\pi(G)$. It remains to show how to derive matching $\mathfrak{M} = \pi^{-1}(\mathfrak{M}')$ on G.

Recall that $\mathfrak{M}'$ is encoded by $[\text{partner}'_u]$ for $u \in V$ s.t. $\text{partner}'_u = v$ if there is a $v \in V$ with $\{u, v\} \in \mathfrak{M}'$ and $\text{partner}'_u = N$ otherwise. We permute $[\text{partner}'_u]_{u\in V}$ by π^{-1} using the inverse of the previously locally chosen permutations to obtain $[\text{partner}''_u]_{u\in V}$. Furthermore, we apply π to $[id_u]_{u\in V}$ where we initialize $[id_u] \leftarrow [u]$ for $u \in V$. Both steps can be executed in parallel and cost $\mathcal{O}(N \log N \cdot p)$ arithmetic invocations and $\mathcal{O}(\log N \cdot p)$ rounds. Then, we pad the result $[id'_u]_{u\in V}$ by $[id'_N] \leftarrow [N]$. Finally, we set $[\text{partner}_u] \leftarrow \text{DEMUX}^{N+1}([\text{partner}''_u]) \cdot [id'_i]_{0\leq i\leq N}$ for all $u \in V$ in parallel. This step costs $\mathcal{O}(N^2)$ arithmetic invocations, $\mathcal{O}(N^2)$ binary invocations, $\mathcal{O}(\log \log N)$ rounds, $\mathcal{O}(N)$ edaBits and $\mathcal{O}(N^2)$ daBits using naïve dot product calculation by $N + 1$ arithmetic multiplications each. Note that a more efficient protocol for dot products as given for Shamir's secret-sharing [CH10] decreases the number of arithmetic invocations to $\mathcal{O}(N)$.

It remains to prove that the result correctly represents $\pi^{-1}(\mathfrak{M}')$. Let $\{u, v\} \in \pi^{-1}(\mathfrak{M}')$ implying that $\{\pi(u), \pi(v)\} \in \mathfrak{M}'$. Thus, $\text{partner}''_u = \text{partner}'_{\pi(u)} = \pi(v) < N$. Then:

$$[\text{partner}_u] = \text{DEMUX}^{N+1}([\text{partner}''_u]) \cdot [id'_i]_{0\leq i\leq N} = [id'_{\text{partner}''_u}] = [id'_{\pi(v)}] = [\pi^{-1}(\pi(v)] = v.$$

Now, let u not be matched, i.e., $\text{partner}''_u = \text{partner}'_{\pi(u)} = N$. Then:

$$[\text{partner}_u] = [id'_{\text{partner}''_u}] = [id'_N] = [N].$$

Using node shuffling only increases the number of arithmetic invocations from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^3 + N^2 \log N \cdot p)$ and the required rounds from $\mathcal{O}(N \log N \cdot \log b)$ to $\mathcal{O}(N \log N \cdot \log b + \log N \cdot p)$ compared to the deterministic protocol. Note that in an outsourcing setting, the number of parties p is usually small so that the increase of complexity is insignificant. For high p, other shuffling approaches like selecting a random value per item to shuffle and securely sort according to such random values [MSZ15] may be more efficient due to not requiring to shuffle in $\mathcal{O}(p)$ phases where different parties input their permutations.

5 Evaluation

Next we evaluate the run time and network traffic for our MWM approximation protocol in its random edge selection variant (cf. Section 4.3) and its node shuffling variant (cf. Section 4.4). After selecting a linear secret-sharing scheme and corresponding protocol (Section 5.1) and the benchmark setting (Section 5.2), we present our findings in Section 5.3.

5.1 Underlying Primitives

We implemented our protocol variants using the MPC framework MP-SPDZ [Kel20] version 0.3.2.[5] As underlying MPC protocol we focused on the three-party case, and therefore used the three-party ($p = 3$) protocol by Araki et al. [Ara+16]. Their work provides a linear secret-sharing schemes which supports arithmetic or binary operations in the domains $\mathbb{F}_q$ or $\mathbb{Z}_{2^k}$, and $\mathbb{F}_2$, respectively. We used $\mathbb{F}_q$ as our arithmetic domain together with the provided binary domain. In addition, we give a comparison to using Shamir's secret-sharing scheme [Sha79] for $p = 3$ or more parties in Appendix A.

5.2 Benchmark Setting

We executed the protocol variants in a LXC container with access to 4 CPU cores and 200 GiB of RAM. The host CPU is an AMD EPYC 7702P with a total of 64 cores, a base clock of 2.0 GHz, and hyperthreading enabled. We consider a LAN Setting with a bandwidth of 1 GBit/s and a round-trip time of 1 ms. The required network behavior is achieved with the tool `tc` (traffic control).

We measured the run times and generated traffic for both of our randomized protocol variants, i.e., the ones for random edge selection (cf. Section 4.3) and node shuffling (cf. Section 4.4). We did not run the deterministic base protocol (cf. Section 4.2) alone as it is used as sub-protocol of our node shuffling variant where we measured its share of the total run time. Each protocol was executed 10 times for different numbers of nodes N: Random edge selection was measured for up to $N = 300$, and node shuffling for up to $N = 400$. We took the arithmetic mean as final run times. As number of bits b and statistical security parameter σ, we used the MP-SPDZ default values, i.e., $b = 64$ and $\sigma = 40$. Not that this is sufficient regarding the security of the random edge selection variant with $b_0 = 64$ and $N \leq 300$, i.e., $\max\{b_0, \lceil \sigma/2 + 4\log_2 N \rceil\} = \max\{64, 53\} = 64$ (cf. Proposition 2).

Note that loops in MP-SPDZ are entirely unrolled at compile-time if they are fully parallelized. This leads to immense RAM utilization when compiling our protocol. Ultimately, our restriction to 200 GiB of RAM prevented us from benchmarking our protocol for more than $N = 300$ with random edge selection or more than $N = 400$ with node shuffling.

For the random edge selection protocol, we let the input of the protocol be an already generated adjacency matrix. The node shuffling variant was specifically motivated by nodes representing certain entities that want to get matched. Therefore, we benchmarked it together with an exemplary computation of a corresponding adjacency matrix: For each entity or node, we use an integer vector of dimension 50 as input. Then, for each pair of nodes we compute the squared Euclidean distance between their vectors X and Y, i.e., $(X - Y) \cdot (X - Y)$. In the adjacency matrix, we define two nodes as *compatible* and connect them by an edge, if their squared distance is below a certain threshold, and otherwise not. In this case, the weight is set to some fixed offset minus their squared distance where the offset is sufficiently high s.t. the resulting weight is positive. Thus, we implemented a basic compatibility measure that allows matching nodes with distance below a certain threshold and prefers to match nodes with lower distance.

[5] `https://github.com/data61/MP-SPDZ/releases/tag/v0.3.2`

5.3 Results

In Figure 2 the average run times of both protocol variants are given. Table 1 lists run times and communication for specific numbers of nodes N.

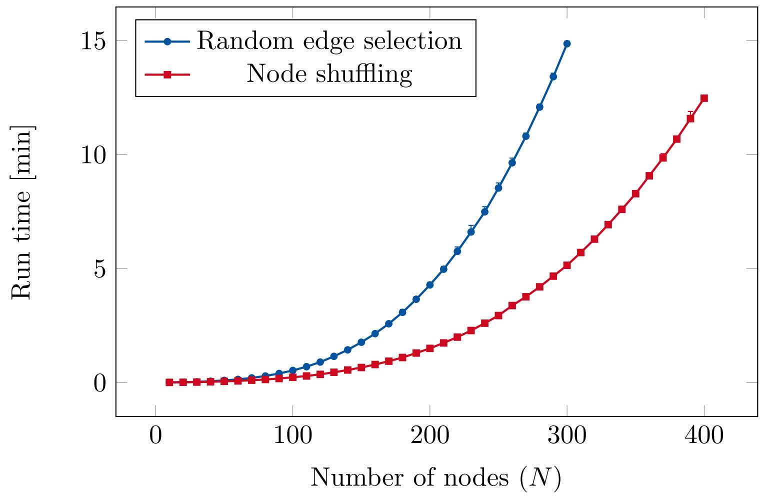

Figure 2: Arithmetic means, minima and maxima of the protocol variants' run times over 10 iterations. Minima and maxima are hardly visible due to low deviations.

Table 1: Arithmetic means of the protocol variants' run times and global communication over 10 iterations.

N	Random Edge Selection		Node Shuffling	
	Time	Comm.	Time	Comm.
50	5.9 s	85.3 MB	3.3 s	41.1 MB
100	32.0 s	635.9 MB	14.0 s	274.3 MB
150	1.8 min	2.1 GB	39.8 s	909.4 MB
200	4.3 min	5.1 GB	1.5 min	2.1 GB
250	8.5 min	9.9 GB	2.9 min	4.0 GB
300	14.9 min	17.1 GB	5.1 min	7.0 GB
350	-	-	8.3 min	11.1 GB
400	-	-	12.5 min	16.4 GB

The average run times for random edge selection are ≈32.0 s for $N = 100$, and ≈14.9 min for $N = 300$. For node shuffling, they are ≈14.0 s for $N = 100$, ≈5.1 min for $N = 300$, and ≈12.5 min for $N = 400$. The maximal measured deviation over all iterations is ≈6.8%. The overall run times of the node shuffling variant are significantly lower than the run times of random edge selection. For $N = 300$, the node shuffling variant is ≈2.9 times faster than random edge selection.

The total traffic generated by all three parties together for random edge selection is ≈635.9 MB for $N = 100$, and ≈17.1 GB for $N = 300$. The communication for node shuffling amounts to ≈274.3 MB for $N = 100$, ≈7.0 GB for $N = 300$, and ≈16.4 GB for $N = 400$. Thus, random edge selection also requires significantly more communication than node shuffling. This results in an overhead of ≈2.4× for $N = 300$ compared to the node shuffling variant.

Both variants run in under 15 minutes for up to 300 with run time and traffic scaling polynomially bounded which matches our previous theoretical complexity analysis. Thus, we can deduce that they scale well for an increasing number of nodes. Depending on the use case, especially the node shuffling variant could be feasible for thousands of nodes.

Recall that our node shuffling variant satisfies invariance under node permutation but not random edge selection while our random edge selection variant satisfies both (cf. Section 4.4.2). Our results show that this stronger guarantee comes at a significant cost. While the random edge selection variant still yields low run times in our benchmarks, use cases that do not necessarily require random edge selection can significantly profit from using the node shuffling variant instead.

5.3.1 Microbenchmarks

We also measured the share that each phase of our protocol variants has in the overall run time. The results for node shuffling are given in Figure 3 where we partition the total run time into the times required to generate the example adjacency matrix as described previously in Section 5.2, to shuffle the adjacency matrix, to compute a matching, to reverse the shuffling on the resulting matching on the shuffled graph, and remaining operations like inputs and outputs.

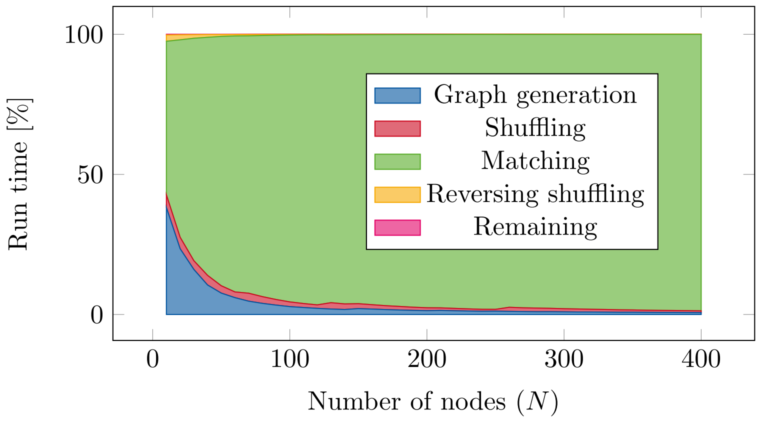

Figure 3: Split of the run time for the node shuffling variant.

Generating the adjacency matrix takes ≈38.2% of the total run time for $N = 10$, drops below 2% for $N = 160$ and is at only ≈0.7% for $N = 400$. In absolute values, its run time increases to ≈5.0 s for $N = 400$. Shuffling the adjacency matrix costs ≈4.5% of the total run time for $N = 10$ decreasing down to ≈0.6% for $N = 400$ corresponding to ≈4.8 s. Note that this phase's share of the run time does not decrease smoothly. This is caused by MP-SPDZ internally padding the number of items to shuffle to powers of 2 which does not change the asymptotic complexity.[6] Reversing the shuffling never takes more than 0.2 s and the remaining operations take at most ≈0.02 s which is an insignificant fraction of the total run time.

The matching phase itself clearly dominates the run time for increasing N taking ≈54.8% of it for $N = 10$ but already more than 90% for $N \geq 60$ and more than 95% for $N \geq 100$. Its share increases up to ≈98.7% for $N = 400$. This is due to it having the highest asymptotic complexity w.r.t. N. Note that the phase exactly is the deterministic base protocol from Section 4.2. This especially shows that achieving invariance under node permutation comes at nearly no cost except for very small graphs which is the reason why we do not give a detailed evaluation of the deterministic case alone.

For random edge selection, the results are given in Figure 4. The only phase besides the matching phase and insignificant input and output phases is the generation of random numbers for all potential edges. This takes less than 0.6 s for each tested $N \leq 300$ and thus is insignificant. In particular, ≈99.9% of the total run time are given by the matching phase for $N = 300$. Recall that this phase

[6]This padding is done as the original Waksman network [Wak68] works on powers of 2. Note that there is an optimized version where a padding is not required [BD02].

is only extended by comparing the random numbers in addition to edge weights. This tells that the overhead induced by the additional comparisons and corresponding operations indeed yields a significant increase in run time.

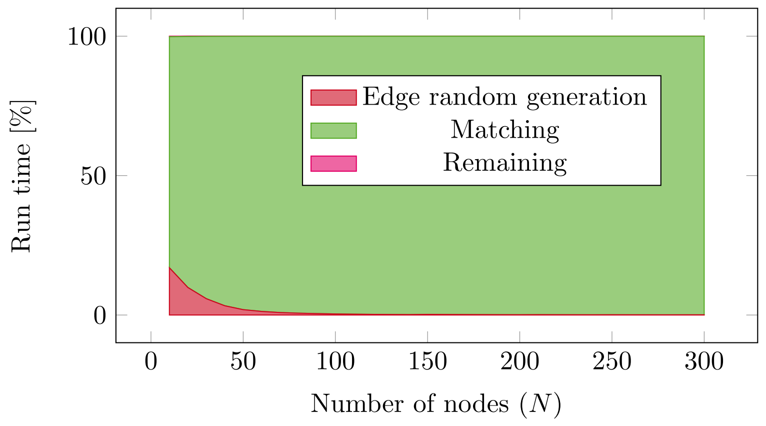

Figure 4: Split of the run time for the random edge selection variant.

6 Conclusion and Future Work

In this paper, we have presented two scalable variants of a protocol for secure maximum weight matching 1/2−approximation. Both variants of our protocol have different qualities regarding unbiasedness and efficiency satisfying different demands that may arise from specific use cases. The faster variant's complexity is $\mathcal{O}(N \log N)$ rounds and $\mathcal{O}(N^3)$ communication for N nodes yielding a run time of ≈12.5 minutes for $N = 400$. The slower variant offers an additional notion of unbiasedness costing an increase by a factor of $\mathcal{O}(\log N)$ regarding rounds and communication which yields a run time of ≈14.9 minutes for $N = 300$.

Regarding future work, we see three main directions. First, our protocol may be extended to a protocol that offers security even against an active adversary. Second, new protocols can be developed using other approximations that offer better quality guarantees or even exact algorithms as their basis. Here it is of special interest how much performance one must pay in order to get better results, i.e., investigating the trade-off between quality of results and run time. Third, while our approximation guarantees results to be at least half as good as optimal ones, their concrete quality can be significantly higher depending on the properties of the input graphs. Thus, an empirical study of the resulting matchings' quality could be of interest when using our protocol for a specific use case.

Acknowledgments

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) within SFB 1119 CROSSING/236615297 as well as GRK 2050 Privacy & Trust/251805230, and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within ATHENE. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Resarch Foundation) - project number (419340256). This work is partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 2236/1.

References

[AC17] Balamurugan Anandan and Chris Clifton. "Secure Minimum Weighted Bipartite Matching". In: *2017 IEEE Conference on Dependable and Secure Computing*. Aug. 2017, pp. 60–67. DOI: 10.1109/DESEC.2017.8073798.

[AL17] Gilad Asharov and Yehuda Lindell. "A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation". en. In: *Journal of Cryptology* 30.1 (Jan. 2017), pp. 58–151. DOI: 10.1007/s00145-015-9214-4.

[Ara+16] Toshinori Araki et al. "High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Oct. 2016, pp. 805–817. DOI: 10.1145/2976749.2978331.

[Ara+21] Toshinori Araki et al. "Secure Graph Analysis at Scale". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Nov. 2021, pp. 610–629. DOI: 10.1145/3460120.3484560.

[Avi83] David Avis. "A Survey of Heuristics for the Weighted Matching Problem". en. In: *Networks* 13.4 (1983), pp. 475–493. DOI: 10.1002/net.3230130404.

[BD02] B. Beauquier and E. Darrot. "On Arbitrary Size Waksman Networks and their Vulnerability". In: *Parallel Processing Letters* 12.03n04 (Sept. 2002), pp. 287–296. DOI: 10.1142/S0129626402000999.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation". In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. STOC '88. Jan. 1988, pp. 1–10. DOI: 10.1145/62212.62213.

[BMW22] Malte Breuer, Ulrike Meyer, and Susanne Wetzel. "Privacy-Preserving Maximum Matching on General Graphs and its Application to Enable Privacy-Preserving Kidney Exchange". In: *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy*. CODASPY '22. Apr. 2022, pp. 53–64. DOI: 10.1145/3508398.3511509.

[BS15] Marina Blanton and Siddharth Saraph. "Oblivious Maximum Bipartite Matching Size Algorithm with Applications to Secure Fingerprint Identification". en. In: *Computer Security – ESORICS 2015*. Lecture Notes in Computer Science. 2015, pp. 384–406. DOI: 10.1007/978-3-319-24174-6_20.

[Can00] Ran Canetti. "Security and Composition of Multiparty Cryptographic Protocols". en. In: *Journal of Cryptology* 13.1 (Jan. 2000), pp. 143–202. DOI: 10.1007/s001459910006.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgard. "Multiparty Unconditionally Secure Protocols". In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. STOC '88. Jan. 1988, pp. 11–19. DOI: 10.1145/62212.62214.

[CH10] Octavian Catrina and Sebastiaan de Hoogh. "Improved Primitives for Secure Multiparty Integer Computation". en. In: *Security and Cryptography for Networks*. Lecture Notes in Computer Science. 2010, pp. 182–199. DOI: 10.1007/978-3-642-15317-4_13.

[Cra+18] Ronald Cramer et al. "SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority". en. In: *Advances in Cryptology – CRYPTO 2018*. Lecture Notes in Computer Science. 2018, pp. 769–798. DOI: 10.1007/978-3-319-96881-0_26.

[Dam+06] Ivan Damgård et al. "Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation". en. In: *Theory of Cryptography*. Lecture Notes in Computer Science. Springer, 2006, pp. 285–304. DOI: 10.1007/11681878_15.

[Dam+19] Ivan Damgård et al. "New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning". In: *2019 IEEE Symposium on Security and Privacy (SP)*. May 2019, pp. 1102–1120. DOI: 10.1109/SP.2019.00078.

[DE16] Jack Doerner, David Evans, and abhi shelat abhi. "Secure Stable Matching at Scale". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Oct. 2016, pp. 1602–1613. DOI: 10.1145/2976749.2978373.

[DH03] Doratha E Drake and Stefan Hougardy. "A Simple Approximation Algorithm for the Weighted Matching Problem". en. In: *Information Processing Letters* 85.4 (Feb. 2003), pp. 211–213. DOI: 10.1016/S0020-0190(02)00393-9.

[DH05] Doratha E. Drake Vinkemeier and Stefan Hougardy. "A Linear-Time Approximation Algorithm for Weighted Matchings in Graphs". In: *ACM Transactions on Algorithms* 1.1 (July 2005), pp. 107–122. DOI: 10.1145/1077464.1077472.

[Die17] Reinhard Diestel. *Graph Theory*. en. 5th ed. Vol. 173. Graduate Texts in Mathematics. 2017. DOI: 10.1007/978-3-662-53622-3.

[DP14] Ran Duan and Seth Pettie. "Linear-Time Approximation for Maximum Weight Matching". In: *Journal of the ACM* 61.1 (Jan. 2014), 1:1–1:23. DOI: 10.1145/2529989.

[Edm65] Jack Edmonds. "Maximum Matching and a Polyhedron With 0,1-Vertices". en. In: *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics* 69B (Jan. 1965), p. 125. DOI: 10.6028/jres.069B.013.

[Esc+20] Daniel Escudero et al. "Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits". en. In: *Advances in Cryptology – CRYPTO 2020*. Lecture Notes in Computer Science. 2020, pp. 823–852. DOI: 10.1007/978-3-030-56880-1_29.

[Gab90] Harold N. Gabow. "Data Structures for Weighted Matching and Nearest Common Ancestors With Linking". In: *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*. SODA '90. Jan. 1990, pp. 434–443.

[Gol06] Philippe Golle. "A Private Stable Matching Algorithm". en. In: *Financial Cryptography and Data Security*. Lecture Notes in Computer Science. 2006, pp. 65–80. DOI: 10.1007/11889663_5.

[GT91] Harold N. Gabow and Robert E. Tarjan. "Faster Scaling Algorithms for General Graph Matching Problems". In: *Journal of the ACM* 38.4 (Oct. 1991), pp. 815–853. DOI: 10.1145/115234.115366.

[Hal35] P. Hall. "On Representatives of Subsets". en. In: *Journal of the London Mathematical Society* s1-10.1 (1935), pp. 26–30. DOI: 10.1112/jlms/s1-10.37.26.

[HH10] Sven Hanke and Stefan Hougardy. *New Approximation Algorithms for the Weighted Matching Problem*. en. Report No. 101010, Research Institute for Discrete Mathematics, University Of Bonn. 2010.

[HK12] Chien-Chung Huang and Telikepalli Kavitha. "Efficient Algorithms for Maximum Weight Matchings in General Graphs With Small Edge Weights". In: *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete algorithms*. SODA '12. Jan. 2012, pp. 1400–1412.

[Kel20] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. Oct. 2020, pp. 1575–1590.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. en. 2nd ed. CRC Press, Nov. 2014.

[Knu96] Donald Knuth. *Stable Marriage and Its Relation to Other Combinatorial Problems: An Introduction to the Mathematical Analysis of Algorithms*. en. Vol. 10. CRM Proceedings and Lecture Notes. Oct. 1996. DOI: `10.1090/crmp/010`.

[Lau+12] John Launchbury et al. "Efficient Lookup-Table Protocol in Secure Multiparty Computation". In: *ACM SIGPLAN Notices* 47.9 (Sept. 2012), pp. 189–200. DOI: `10.1145/2398856.2364556`.

[Lin17] Yehuda Lindell. "How to Simulate It – A Tutorial on the Simulation Proof Technique". en. In: *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*. Ed. by Yehuda Lindell. Information Security and Cryptography. 2017, pp. 277–346. DOI: `10.1007/978-3-319-57048-8_6`.

[MSZ15] Mahnush Movahedi, Jared Saia, and Mahdi Zamani. "Secure Multi-Party Shuffling". en. In: *Structural Information and Communication Complexity*. Lecture Notes in Computer Science. 2015, pp. 459–473. DOI: `10.1007/978-3-319-25258-2_32`.

[Pet12] S. Pettie. "A Simple Reduction From Maximum Weight Matching to Maximum Cardinality Matching". en. In: *Information Processing Letters* 112.23 (Dec. 2012), pp. 893–898. DOI: `10.1016/j.ipl.2012.08.010`.

[Pre99] Robert Preis. "Linear Time 1/2-Approximation Algorithm for Maximum Weighted Matching in General Graphs". en. In: *STACS 99*. Ed. by Christoph Meinel and Sophie Tison. Lecture Notes in Computer Science. 1999, pp. 259–269. DOI: `10.1007/3-540-49116-3_24`.

[PS04] Seth Pettie and Peter Sanders. "A Simpler Linear Time $2/3 - \epsilon$ Approximation for Maximum Weight Matching". en. In: *Information Processing Letters* 91.6 (Sept. 2004), pp. 271–276. DOI: `10.1016/j.ipl.2004.05.007`.

[Ria+17] M. Sadegh Riazi et al. "Toward Practical Secure Stable Matching". en. In: *Proceedings on Privacy Enhancing Technologies* 2017.1 (Jan. 2017), pp. 62–78. DOI: `10.1515/popets-2017-0005`.

[RW19] Dragos Rotaru and Tim Wood. "MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security". en. In: *Progress in Cryptology – INDOCRYPT 2019*. Lecture Notes in Computer Science. 2019, pp. 227–249. DOI: `10.1007/978-3-030-35423-7_12`.

[Sha79] Adi Shamir. "How to Share a Secret". In: *Communications of the ACM* 22.11 (Nov. 1979), pp. 612–613. DOI: `10.1145/359168.359176`.

[Wak68] Abraham Waksman. "A Permutation Network". In: *Journal of the ACM* 15.1 (Jan. 1968), pp. 159–163. DOI: `10.1145/321439.321449`.

[WMW17a] Stefan Wüller, Ulrike Meyer, and Susanne Wetzel. "Privacy-Preserving Multi-Party Bartering Secure Against Active Adversaries". In: *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. Aug. 2017, pp. 205–20509. DOI: `10.1109/PST.2017.00032`.

[WMW17b] Stefan Wüller, Ulrike Meyer, and Susanne Wetzel. "Towards Privacy-Preserving Multiparty Bartering". en. In: *Financial Cryptography and Data Security*. Lecture Notes in Computer Science. 2017, pp. 19–34. DOI: `10.1007/978-3-319-70278-0_2`.

[Wül+17] Stefan Wüller et al. "Using Secure Graph Algorithms for the Privacy-Preserving Identification of Optimal Bartering Opportunities". In: *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*. WPES '17. Oct. 2017, pp. 123–132. DOI: `10.1145/3139550.3139557`.

[WW18] Weicheng Wang and Shengling Wang. "Privacy Preservation for Dating Applications". en. In: *Procedia Computer Science*. 2017 International Conference on Identification, Information and Knowledge in the Internet of Things 129 (Jan. 2018), pp. 263–269. DOI: `10.1016/j.procs.2018.03.074`.

[Yao82] Andrew C. Yao. “Protocols for Secure Computations”. In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. Nov. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.

[Yi+16] Xun Yi et al. “Practical Privacy-Preserving User Profile Matching in Social Networks”. In: *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. May 2016, pp. 373–384. DOI: 10.1109/ICDE.2016.7498255.

[Zah+16] Samee Zahur et al. “Revisiting Square-Root ORAM: Efficient Random Access in Multiparty Computation”. In: *2016 IEEE Symposium on Security and Privacy (SP)*. ISSN: 2375-1207. May 2016, pp. 218–234. DOI: 10.1109/SP.2016.21.

A Evaluation Using Shamir’s Secret-Sharing

Shamir’s secret-sharing scheme [Sha79] is another linear secret-sharing scheme that can be used to as a basis for a passively secure protocol in the honest majority setting [BGW88; AL17]. While the three-party protocol by Araki et al. [Ara+16] is newer than the original BGW-protocol [BGW88] based on Shamir’s secret-sharing scheme, the BGW-protocol [BGW88] allows to use an arbitrary number of parties p. As MP-SPDZ [Kel20] allows to run the same high-level protocol on different low-level protocols, we are able to give a short comparison between both low-level protocols. This allows to assess the cost of switching to Shamir’s secret-sharing scheme which is not restricted to three parties.

For the arithmetic domain, we again use $\mathbb{F}_q$. While Shamir’s secret-sharing scheme requires a field that has cardinality larger than the number of parties, MP-SPDZ allows using $\mathbb{F}_{2^8}$ to embed the binary domain in. We additionally run our protocol using Shamir’s secret-sharing for node shuffling with $N \in \{100, 200, 300\}$, and three and ten parties. For the ten party case, we give our LXC container access to 11 CPU cores. Again, we use ten iterations for each case. This allows to assess the overhead resulting from using Shamir’s secret-sharing scheme in the three-party case and the additional overhead from using it for more parties. We only run benchmarks for the node shuffling variant of our protocol. The measured run times and traffic in comparison to the previous measured run times (cf. Section 5.3) are given in Table 2.

Table 2: Arithmetic means of the protocol’s node shuffling variant’s run times and global communication over 10 iterations using different underlying protocols and numbers of parties p.

N	$p = 3$ parties				$p = 10$ parties	
	Araki et al. [Ara+16]		Shamir [Sha79]/BGW [BGW88]		Shamir/BGW	
	Time	Comm.	Time	Comm.	Time	Comm.
100	14.0 s	274.3 MB	34.2 s	769.6 MB	3.4 min	34.0 GB
200	1.5 min	2.1 GB	4.6 min	5.8 GB	25.3 min	255.3 GB
300	5.1 min	7.0 GB	16.3 min	19.7 GB	85.3 min	859.2 GB

The results show that using Shamir’s secret-sharing scheme for $p = 3$ parties is significantly less efficient than using the protocol by Araki et al [Ara+16]. The run time increases by a factor of up to 3.2× for $N = 300$. The communication increases by a factor of more than 2.8× for all N.

Increasing the number of parties for Shamir’s secret-sharing scheme to $p = 10$ yields for $N = 300$ 5.2× the run time and 43.6× the communication of using the same secret-sharing scheme for $p = 3$. For lower N, these factors are higher. Compared to using the protocol by Araki et al [Ara+16], this is an overhead of 16.7× regarding run time and 122.7× regarding communication for $N = 300$. Thus, using Shamir’s secret-sharing scheme to increase the number of parties is possible but comes at cost of significant overheads compared to the three-party case based on the work of Araki et al [Ara+16].