Cooperative Computing for Autonomous Data Centers

Jonathan Berry *, Michael Collins[‡], Aaron Kearns[†], Cynthia A. Phillips*, Jared Saia[†] and Randy Smith*
*Sandia National Laboratories, Albuquerque, NM
Email: jberry@sandia.gov; caphill@sandia.gov; ransmit@sandia.gov
[†]Computer Science Department, University of New Mexico,
Farris Engineering Building, Albuquerque, NM 87131-1386
Email: aaron.kearns.username@gmail.com; saia@cs.unm.edu
[‡]Christopher Newport University, Newport News, VA, michael.collins@cnu.edu

***Abstract*—We present a new distributed model for graph computations motivated by limited information sharing. Two or more independent entities have collected large social graphs. They wish to compute the result of running graph algorithms on the entire set of relationships. Because the information is sensitive or economically valuable, they do not wish to simply combine the information in a single location. We consider two models for computing the solution to graph algorithms in this setting: 1) limited-sharing: the two entities can share only a polylogarithmic size subgraph; 2) low-trust: the entities must not reveal any information beyond the query answer, assuming they are all honest but curious. We believe this model captures realistic constraints on cooperating autonomous data centers.**

We have results for both models for s-t connectivity, one of the simplest graph problems that requires global information in the worst case. In the limited-sharing model, our results exploit social network structure. Standard communication complexity gives polynomial lower bounds on s-t connectivity for general graphs. However, if the graph for each data center has a giant component and these giant components intersect, then we can overcome this lower bound, computing s-t connectivity while exchanging $O(\log^2 n)$ bits for a constant number of data centers. We can also test the assumption that the giant components overlap using $O(\log^2 n)$ bits provided the (unknown) overlap is sufficiently large.

The second result is in the low trust model. We give a secure multi-party computation (MPC) algorithm that 1) does not make cryptographic assumptions when there are 3 or more entities; and 2) is efficient, especially when compared to the usual garbled circuit approach. The entities learn only the yes/no answer. No party learns anything about the others' graph, not even node names. This algorithm does not require any special graph structure. This secure MPC result for s-t connectivity is one of the first that involves a few parties computing on large inputs, instead of many parties computing on a few local values.

***Keywords*-graph algorithms; privacy; s-t connectivity; distributed computing models; social networks;**

I. Introduction

Consider two entities, Alice and Bob, who autonomously observe the world, collecting information on social relationships, which each represents as a social graph. Alice would like to combine her information with Bob's to answer a query about the full set of relationships. It is in Bob's best interest to cooperate, since he may need Alice's help in the future. But there are barriers to total information sharing, which we model in two ways: 1) limited-sharing: Alice and Bob can share only a polylogarithmic size subgraph; 2) low-trust: Alice and Bob must not reveal any information beyond the query answer, assuming they are both honest but curious.

We are motivated by recent trends in data collection over large social networks. Specifically, we consider a small number of autonomous data centers that are collecting data about a social network. Periodically, these centers may want to collaborate to solve a computational query. However, data is a critical resource, so the centers want to answer the query while sharing as little data as possible.[1]

Brickell and Shmatikov [1] were similarly motivated when they conducted related work using a different model. They provide several compelling examples involving commercial entities that must evaluate the consequences of a potential merger, or coordinate in some useful way without revealing private details. Networking companies would be interested in measuring the efficiency of joint infrastructure before committing to a merger, shipping companies would similarly need to know the effects of merged capacities on efficient routing, and social networking websites may wish to collaborate to compute more accurate statistical measures of their users' behavior without revealing private information.

Our cooperative computing problem has overlap with two mature research areas that deal with privacy: secure multiparty computation and differential privacy. In secure multiparty computation (MPC), a set of m parties, each of whom has a private input, want to compute an m-variate function over their inputs, without revealing

[1]We imagine that user data may be valued more by data centers than it is valued by individuals, since the data centers can monetize that data.

any information about their inputs (see e.g. [2] for a survey of MPC). A novelty of our problem when compared to most results in MPC is that the size of the inputs held by the parties are very large. In differential privacy, a single entity holds all the data, and the goal is to answer queries as accurately as possible, while minimizing the chance of leaking information about individual records in that data (see e.g. [3]). By contrast, in our problem, multiple entities hold the data. The entities seek to minimize not the chance of identifying individual records in the data, but rather the total amount of information revealed about their own data set. Also, they require that the query is answered exactly.

The term "data center" frequently refers to a distributed set of resources owned by a single entity such as Google, cloud systems, or providers of web services. Cooperation in such settings is a given, with research focusing on providing quality of service while minimizing energy or other costs. See for example these surveys [4], [5]. In our setting, the data centers are owned by autonomous, potentially competing parties who nevertheless wish to compute cooperatively in some cases while minimizing the loss of proprietary information.

A. Our Model

Our model assumes a small number of autonomous data centers. For simplicity of discussion here, we will assume two centers, but our results generalize to any small constant number of centers. Let G_a be Alice's graph and G_b be Bob's graph. Alice and Bob wish to perform computations on the graph $G_U = G_a \cup G_b$. Let n be the number of nodes in G_U. Alice and Bob build their graphs by observing a common world graph G. The fundamental observation is an edge representing a relationship between two people. Alice and Bob know nothing about each other's graphs. However, the nodes come from a shared namespace, so if Alice gives Bob an edge (or vice versa), he will recognize the nodes if he has seen them before, and therefore he knows where that edge fits into his graph.

The world graph G is a social network, and therefore has topological properties of a social network. In general, Alice and Bob can each sample from this graph according to arbitrary distributions. Thus G_a, G_b, and G_U do not necessarily inherit social network topological properties in the worst case. However, for the algorithms in Section III, we do assume that G_a and G_b both have a giant component, a "standard" social network property. See Section III-A for a more detailed discussion of this assumption.

B. s-t Connectivity

In this paper, as a first approach to cooperative computing, we focus only on the problem of s-t connectivity. In particular, we assume that Alice and Bob want to know if there is a path between node s and node t in G_U. Our motivation for focusing on this problem is two-fold. First, s-t connectivity is one of the simplest graph theoretic problems that can require non-trivial communication in the case where the edges of the graph are partitioned between Alice and Bob. Second, we believe that s-t connectivity is a problem of interest for autonomous data centers, both for direct queries and as a building block for more sophisticated graph theoretic problems such as clustering.

C. Our Results

Our primary contributions are the model of Section I-A, and algorithms for s-t connectivity in two variants of the model. First, a *limited-sharing algorithm* that exploits social network properties, such as a giant component, to achieve polylogarithmic communication complexity under certain conditions (thereby circumventing a communication lower bound of $\Omega(n \log n)$ for general networks). Second, two versions of a *low-trust MPC algorithm* that do not depend on cryptographic assumptions if there are at least three data centers. These MPC algorithms do not use heavy-weight techniques like oblivious RAM. We also avoid compiling either MPC algorithm into a single large circuit. We believe that the correctness of the limited-sharing algorithm is clear enough that prose descriptions are more appropriate than a sequence of lemmas.

II. Related Work

A. Communication Complexity

In a classical lower-bound result, Hajnal, Maass, and Turàn [6] show that the communication complexity of graph s-t connectivity, connectivity and bipartiteness all are $\Omega(n \log n)$ in a model that partitions the edges of a graph among two data centers with no intersection. These results had fundamental implications for VLSI design. In contrast, our model is inspired by human activity. We exploit the expected properties of social networks to circumvent this lower bound and to obtain polylogarithmic communication for s-t connectivity in social networks.

B. Cloud Computing

Modern cloud computing offers additional motivation for our model. Either of the entities represented by Alice and Bob might have large amounts of data stored in the same platform-as-a-service cloud resource (for example the Amazon EC2 cloud [7]). Alice and Bob still have the same concerns and will not share too much of their data. Their portions of the cloud are simply abstractions for geographically-distributed data centers, whether they reside on racks in the same building or not. Furthermore, their co-location on the same virtual resource may encourage the actual implementation of algorithms developed using our limited-sharing and/or low-trust models.

C. Secure Multiparty Computation (MPC)

Brickell and Shmatikov [1] consider protocols for secure two-party computation of all-pairs shortest paths and single-source shortest distance. These protocols of course reveal much more than the existence of an s-t path. In addition, their protocols assume that each graph is defined on the same set of nodes. That is, they assume that both parties know all the nodes. If the node sets were different to begin with, then this means that each party must tell the other parties its nodes. This might be equivalent to giving your customer list away to a competitor. Thus, in our work, we seek to avoid this assumption.

D. Giant component structure in social networks

Many real-world graphs including the Internet and many social networks have a giant component that contains almost all the nodes, and a small second-largest component. Easley and Kleinberg [8] describe informally why this is true for real human relationships. Researchers have found a giant component in networks representing social relationships. For example, Aiello *et al.* [9] looked at call graphs from long-distance phone calls in 1998 and showed they have a giant component with the next smallest component $O(\log n)$. Kang *et al.* [10] show empirically that the size of the second smallest component for LinkedIn and Wikipedia is bounded by the Dunbar number, which is roughly 150.

Many randomized graph generators also have a small second-largest component. For example, the simplest random graphs, Erdös-Rényi graphs [11], where each edge appears with probability p, has a giant component with a second-largest component of size $O(\log n)$ when $p = c/n$ for a constant $c > 1$. When $p \geq \log n/n$ the graph is almost certainly fully connected. We are interested primarily in the case where the average degree is bounded, which happens when $p = c/n$. Since the average degree of such a graph is approximately c, then $c > 1$ is the interesting case. Of course, real-world graphs look nothing like Erdös-Rényi graphs, but this is an interesting initial case theoretically.

Chung-Lu graphs, where each node v_i has a desired degree d_i, and edge (v_i, v_j) is included with probability proportional to $d_i d_j$, also have second largest components of size $O(\log n)$ when the desired degree distribution has a power law. Specifically, Aiello *et al.* [9] consider distributions where the number of nodes with degree d is e^α/d^β for parameters α and β. The parameter β is the power-law coefficient. Many graphs obey or approximately obey a power-law degree distribution (where the number of nodes of degree d is proportional to $d^{-\beta}$). Boginski *et al.* [12] argue that graphs that represent correlations among financial instruments ("market graphs") have power law degree distributions. Siganos *et al.* [13] argue that the AS-level Internet graph has power law structure. Aiello *et al.* [9] prove that for Chung-Lu graphs with power-law degree distributions with $\beta < 1$, the graph is almost surely connected. If $1 < \beta < 2$, there is a giant component and the second largest component is $O(1)$. For $2 < \beta < \beta_0 \approx 3.4785$, there is a giant component with smaller components $O(\log n)$. The special case of $\beta = 2$ has smaller small components. If $\beta > \beta_0$, there is almost surely no giant component. Chung and Lu also show (among other results) that when the average desired degree is at least 1.5, a Chung-Lu graph has a unique giant component where the total degree of nodes in the giant component is $\Omega(n)$. Furthermore, the next largest component has $O(\log n)$ nodes [14].

III. A Limited-Sharing s-t Connectivity Algorithm

In this section we give a low-communication algorithm for s-t connectivity for graphs G_U in which the second-largest connected component, called C_s, is small compared to the number of nodes in G_U. Although in some cases, the algorithm could be modified to return a path between nodes s and t, in general it only returns the boolean answer to the question "Are nodes s and t connected?"

The algorithm requires the following **assumptions**. We assume G_U has a single giant component and $|C_s| = O(\log n)$, where n is the number of nodes in G_U. The algorithm also works in our low-sharing model for $|C_s|$ polylogarithmic in n, that is where $|C_s| = O(\log^h n)$ for a constant h. We assume Alice and Bob have agreed to an upper bound γ on the number of shared node names. This algorithm is correct if and only if $\gamma \geq |C_s|$.

We discuss how G_U might inherit the giant component property property from G. We give an algorithm that exchanges a polylogarithmic number of bits. We discuss one randomized way that Bob and Alice can prove G_U has a giant component provided the giant component in G_a and the giant component in G_b overlap by a constant fraction.

A. Inheriting Giant Component Structure

In section II-D, we discussed evidence that social networks have a single giant component containing a large fraction of the graph, where the size of the second-largest component is polylogarithmic in the number of nodes. We assume the base "world" graph G has this property. Depending on the way each data center samples graph G, it's possible that $|G_a| + |G_b| \ll |G|$. However, we assume, since each represents the contents of a data center, that $|G_a|$ and $|G_b|$ are large. It is always possible for an adversary to select edges for G_a and G_b that do not reflect the overall component structure of the underlying graph G.

It is an interesting open problem to characterize the sampling distributions for Alice and Bob that preserve a giant component and a small second-largest component in subgraphs like $G_a \subseteq G$. For the simplest case, the subgraphs inherit these properties. Consider an Erdös-Rényi (ER) graph created with a probability $p = c/n$ for a constant $c > 1$. These are values for p that give the desired property in G. Create $G_i \subseteq G$ with n_i nodes by sampling the edges of G with probability p_i. This is equivalent to generating an ER graph with initial probability $p{*}p_i$, since an edge survives with that probability. If $p{*}p_i > c'/n$ for some new constant $c' > 1$, then each of the subgraphs has a giant component and the second largest component is be of size $O(\log n_i)$.

For our s-t connectivity algorithm, we assume that both Alice's graph G_a and Bob's graph G_b have giant components called $G_{a\ell}$ and $G_{b\ell}$ respectively. Since the graphs are social graphs, one could imagine them evolving the way other social networks do, and having a giant component based on the evidence in section II-D. If Alice and Bob are companies and the nodes are customers, one could imagine the customer base growing in part by word of mouth to friends and acquaintances, thus creating components that evolve as human social networks do.

The graph $G_U \equiv G_a \cup G_b$ is likely to have a giant component as well. If $G_{a\ell} \cap G_{b\ell} \neq \emptyset$, then it does, because the giant components of G_a and G_b, each containing most of G_a and G_b respectively join to contain most of G_U. If $G_{a\ell} \cap G_{b\ell} = \emptyset$, then the giant components may still be joined by a path through smaller components to form a giant component in G_U.

B. *Shell Expansion*

The base algorithm is a straightforward distributed generalization of breadth-first search (BFS). This process could have high communication complexity if always run to completion, but our algorithms truncates the search before violating the communication budget.

Alice, WLOG, begins a shell expansion from s by computing all nodes in the component of G_a that contains s. This is the first shell S_1. If this shell is disjoint from $G_{a\ell}$, she sends all of the nodes in S_1 to Bob. Bob finds all the nodes in S_1 that are also in G_b, contracts them to a single node V_1, and does a BFS from V_1 to find all the nodes in graph G_b connected to any node in S_1. These nodes are the second shell, S_2. If S_2 is disjoint from $G_{b\ell}$, Bob sends the nodes in $S_2 - S_1$ to Alice, who repeats the process to form the third shell and so on.

The process stops when $S_i = S_{i+1}$ for some i (no new nodes are found) or when sending the newly discovered nodes would violate the communication/sharing budget.

C. *The Algorithm*

As described in section III-A, we assume that graphs G_a and G_b have giant components $G_{a\ell}$ and $G_{b\ell}$ respectively. We assume that components $G_{a\ell}$ and $G_{b\ell}$ intersect so that graph $G_U \equiv G_a \cup G_b$ also has a giant component. We denote the second largest connected component by C_s. Recall Alice and Bob have agreed to an upper bound γ on the number of shared names. This algorithm is correct if and only if $\gamma > |C_s|$.

The algorithm does a shell expansion starting from both s and t. The expansion for each component stops when it reaches any of these stopping conditions:

1) The shell containing node s merges with the shell containing node t.
2) The component stops growing. That is, no new nodes are added during an iteration.
3) The component falls into local giant components $G_{a\ell}$ (which Alice determines) or $G_{b\ell}$ (which Bob determines).
4) The data center currently growing the component determines that the number of nodes in the component exceeds γ. In this case, the center stops before sending new elements to ensure the number of shared names is at most γ.

The connectivity answer depends upon the stopping condition for the shell expansions from nodes s and t. There are the following cases:

A) For stopping condition 1, the nodes are connected and the algorithm returns "yes."
B) If either shell, say the search from s, stops because it is completely found (stopping condition 2) and it does not contain the other node (t in this case), then the algorithm terminates the other shell expansion, if necessary, and returns "no."
C) Otherwise, both shells terminate with stopping conditions 3 or 4. In stopping condition 4, the shell has exceeded the size of the second-largest component, and therefore must be in the giant component of G_U. For stopping condition 3, the shell is also in the giant component of G_U since both $G_{a\ell}$ are $G_{b\ell}$ are in this giant component. Therefore the algorithm returns "yes."

The communication complexity of the algorithm is $O(\gamma \log n) = O(|C_s| \log n)$, assuming $O(\log n)$-bit vertex names and a good guess $\gamma = O(|C_s|)$.

The above discussion informally argues the following:

Claim III.1. If Alice and Bob's social graphs both have giant components, these giant components intersect, and they know a $\gamma > |C_s|$, such that $\gamma = O(|C_s|)$, where C_s is the second largest component of G_U, then simple BFS-like shell expansion can answer the s-t connectivity decision problem with $O(\gamma \log n)$ communication. When $|C_s| = O(\log n)$, as is typical for social networks, the communication is $O(\log^2 n)$.

Alice and Bob could also send the edges within the shells, which gives shortest paths when the algorithm says "yes" in case A. This is likely a rare case. Alice and Bob can agree to send the edges if and when this happens. This requires communication $O(|C_s|^2 \log n)$, which is still polylogarithmic.

If Alice and Bob know that $G_{a\ell}$ are $G_{b\ell}$ do *not* intersect, they can alter the algorithm. In this case, when a shell expansion is stopped for size without dropping into a $G_{a\ell}$ or $G_{b\ell}$, the algorithm cannot give an answer. Also, if one component drops into $G_{a\ell}$ and the other drops into $G_{b\ell}$, then the answer is "no."

This algorithm extends to $r > 2$ data centers by running the protocol through each data center in sequence. The requester sends its s component and t component to the next data center in order (with wraparound). If the requester has not seen s and/or t, these components may be just s and/or just t respectively. The next data center grows the component and passes the full set of nodes to the next center and so on. Thus if the requester is center i, one round of communication passes around the ring of centers in the order $i, i+1, \ldots, r-1, r, 1, 2, \ldots, i-1, i$. Each data center follows the protocol above, with the growing components passed around the ring of centers until the growth terminates with one of the above conditions. The center that detects the termination condition then sends a message to the requester. From the point of view of a single center, it is running the two-center protocol with a merged version of the other $r-1$ centers.

Each node is tagged with the ID of the center that added it to the component. For example, if center j adds node u to the component, it tags node u. Then center j removes node u from the circulating list of names when it comes back in the next round. Thus each node in the component is communicated to each center at most once. Therefore, the communication complexity is $O(r\gamma \log n) = O(\gamma \log n)$ for constant r.

D. Verifying G_U Has a Giant Component

Alice and Bob may wish to verify that G_U has a giant component. If they can determine that $G_{a\ell}$ and $G_{b\ell}$ intersect, then the answer is a clear "yes." Alice and Bob can use a randomized set intersection algorithm. Given two sets of objects, we wish to find an element in their intersection or determine that the intersection is empty. These are sets of vertex names for us, subsets of the vertices in G_U. We describe a randomized algorithm that finds a node in common with polylogarithmic communication provided the two sets intersect in a constant fraction of nodes.

In the worst case, with no extra information, if both centers have a subset of vertices drawn from the set $\{1, \ldots, n\}$, where n is the number of nodes in G_U, then determining if the sets are disjoint (i. e. the intersection is empty) requires $\Omega(n)$ bits of communication for a deterministic algorithm [15]. This is also true for randomized set intersection [16]. Håstad and Wigderson showed the complexity can depend on the set sizes rather than the size of the universe [17], but we use set intersection to determine intersections of giant components, which have size asymptotically equal to the universe size n. However, if the set intersection is not only non-empty but also large, a randomized algorithm can find an element in the intersection efficiently. The following analysis arguments are standard, but included for completeness.

Suppose that we know the intersection has size $\mathcal{I}$, where $\mathcal{I}$ is a function that grows with n. Consider the following randomized algorithm: Alice chooses $(cn \log n)/\mathcal{I}$ elements uniformly at random from her giant component, where c is a constant greater than 0. She sends these elements to Bob, who determines if any of these elements are in his giant component. With high probability, this algorithm successfully finds an element in the intersection.

To see this, notice that for a single draw, the probability of successfully picking an element in the intersection is $\mathcal{I}/n$. If the drawing is done without replacement, then the probability for picking an element in the intersection increases for subsequent draws. Thus we have

Prob[no point in intersection selected]

$$\begin{aligned} &\leq \left(1 - \frac{\mathcal{I}}{n}\right)^{\frac{cn \log n}{\mathcal{I}}} \\ &= \left(1 - \frac{1}{\frac{n}{\mathcal{I}}}\right)^{(\frac{n}{\mathcal{I}})c \log n} \\ &\leq e^{-c \log n} \\ &= \frac{1}{n^c}, \end{aligned}$$

where $\log n$ is the natural logarithm. Line (1) follows from the standard inequality:

$$\left(1 - \frac{1}{n}\right)^n \leq \frac{1}{e},$$

in turn derivable from the definition of e. By increasing the constant c or by multiplying by more than $\log n$, one can further decrease the probability of failing to find an element in the intersection, given that the intersection has $\mathcal{I}$ elements. If $\mathcal{I}$ is $\Omega(n)$, then the algorithm requires communication of only $O(\log n)$ names, or $O(\log^2 n)$ bits. The intersection can be a poly-logarithmic factor smaller than n asymptotically and the algorithm still succeeds with high probability using polylog n bits.

If we don't know the value of $\mathcal{I}$, we can find it by recursive doubling: start with a guess of $n/\mathcal{I} = \log(n)$ or some other small value. It's even fine to start with a guess that is a constant. Run the randomized algorithm. If it fails, double the guess, and send only new elements in the next round. Because the communication is $c \log n$ times a function that is growing geometrically, the communication complexity of the entire procedure is bounded asymptotically by the size of the last, longest run. Thus the communication is $O(n \log n)/\mathcal{I}$ even if we don't know $\mathcal{I}$ *a priori*.

IV. A LOW-TRUST s-t CONNECTIVITY ALGORITHM

In this section, we describe our low-trust algorithm for s-t connectivity. This algorithm reveals no information about the graphs held by Alice and Bob, except whether or not there is a path between s and t in the union of the two graphs. For ease of exposition, we first describe an algorithm that assumes shared (known) vertex names. We then extend this algorithm to our final algorithm, which assumes private vertex names and does not reveal names during the algorithm.

A. The Problem

As in section III, Alice and Bob have graphs G_a and G_b on the same universe of node names. They want to determine if there exists an s-t path in the graph G_U, where G_U has node set equal to the union of the nodes sets of G_a and G_b and has edge set equal to the union of edges of G_a and G_b.

We now do not restrict the amount of communication, but we require that the two parties learn nothing about each other's graphs, other than the existence or nonexistence of an s-t path in G_U.

We note that all the results in this section make no assumptions about the graph G_U. Thus, in contrast to our previous results, we now do not require G_U to have any special, social-network properties. However, some properties of social networks make the algorithm run provably faster. Although the model does not place an upper bound on computational complexity, in practice a faster correct algorithm is better.

B. Preliminaries

Our algorithm makes critical use of two tools: Secret Sharing and Secure Multiparty Computation of the MUX function.

Secret Sharing Let z be an arbitrary value in a finite field. Under secret sharing, it is possible to create k shares of z, such that 1) z can be constructed from all the shares; and 2) any subset of fewer than k shares reveals no information about z. For any such z, let $[z]_i$ denote the share of z held by party i in a secret sharing scheme. A simple example (from [18]) of secret sharing when $s = 2$ is to create a line with y-intercept equal to z and random slope, and to let $[z]_i$ be the y-value of the line when $x = i$. Under this scheme it is possible for parties to compute shares of $x + y$ from their shares of x and y without leaking any information; $[x + y]_i$ is just $[x]_i + [y]_i$. Thus we can view addition of shares as having essentially zero cost: it is an operation which each party can perform with a small amount of local computation and no communication.

Secure Multiparty Computation of MUX The parties also need a way to securely compute shares of $\text{MUX}(c, a, b)$ from shares of c, a and b, where

$$\text{MUX}(c, a, b) = \begin{cases} a, & c \neq 0, \\ b, & \text{otherwise.} \end{cases}$$

That is, each party i should learn $[\text{MUX}(c, a, b)]_i$, where at the start of the computation, each party i knows $[a]_i$, $[b]_i$ and $[c]_i$. In addition, party i should gain no additional information about a, b or c. There exist information-theoretically secure protocols to solve this problem when the number of parties is three or larger, see for example the algorithm of Ben-Or, Goldwasser and Wigderson [19]; for 2 parties, Yao's garbled circuit construction provides cryptographic security. Unlike addition, secure computation of shares of MUX is an expensive operation, requiring communication between the participants, so we seek to minimize the number of times it occurs.

C. An Algorithm for Public Vertex Names

We now describe our secure algorithm for the case where all vertex names are public knowledge. We assume that Alice and Bob first agree on the value of a prime N that is larger than the number of vertices squared. The algorithm performs all arithmetic in the finite field $\mathbb{Z}_N$, the integers modulo N. Recall that a finite field is required for the secret sharing implementation.

Alice first determines the connected components of her graph and assigns a different number (modulo N) to label each component. For each node v let x_v be the label assigned by Alice to the component that contains node v. Alice generates shares $[x_v]_a, [x_v]_b$ of each x_v (using linear secret sharing as described above) and gives all $[x_v]_b$ to Bob. Bob does similarly on his graph. Call Bob's labels y_v.

The general idea of the algorithm is that the parties iteratively update these vertex labels, marking a node v as reachable from s whenever v's connected component in either G_a or G_b intersects the set of nodes already marked as reachable from s. We mark v as reachable from s by relabeling $x_v \leftarrow x_s$ or $y_v \leftarrow y_s$ (or both), so any node v which is reachable (in G_U) from s eventually has either $x_v = x_s$ or $y_v = y_s$ (or both). All the updates are done on the shares, so the players do not know the updated labels; all that is revealed at the end is whether or not t has been relabeled as reachable from s.

To begin each player, for each node v, computes a share of P_v, a binary variable which is 0 if and only if there exists a node u such that $x_u = x_s$ and $y_u = y_v$. Algorithm 1 shows how to compute P_v.

Algorithm 1 OddStep

```
1: $P_v = 1$
2: **for** node $u$ **do**
3:     $P_v \leftarrow \text{MUX}((x_s - x_u + y_u - y_v), P_v, 0)$
4: **end for**
```

The iteration at node u sets $P_v = 0$ if $x_u = x_s$ and $y_u = y_v$. If such a node exists then node v is reachable from s in G_U, by traveling along Alice's edges from s to u and then along Bob's edges from u to v. We also need to guarantee the converse: $P_v \neq 0$ if no such u exists. To achieve that we must place some restrictions on Alice's and Bob's component labels. They must be drawn respectively from sets X, Y such that the difference of two elements of X is never equal to the difference of two elements of Y. We can accomplish this by having each $1 < x_u < M$ while each y_u is of the form tM for some $1 < t < M$, where $M^2 < N$ but M is larger than the number of nodes. By restricting the labels in this way, we avoid having to explicitly compute a logical AND (which would require another expensive secure computation), at the cost of making the vertex labels twice as large.

The next step is to update the values of the y_v to encode information about what is reachable from s; players modify their shares so that

$$y_v \leftarrow \text{MUX}(P_v, y_v, y_s) \quad (1)$$

The idea (see Figure 1) is to iterate this process, alternating between updating x_v and y_v, at each stage expanding the set of nodes that are labeled as reachable from s. At each iteration we interchange the roles of x and y, so for instance in the second iteration we perform Algorithm 2 to update x_v instead of y_v. After iterating "enough" times, the players combine their shares to reveal P_t. $P_t = 0$ if and only if there is an $s - t$ path. In the worst case, $2n$ iterations are required. If Alice and Bob each possess nothing but alternating edges on

Algorithm 2 EvenStep

```
1: $P_v = 1$
2: **for** node $u$ **do**
3:     $P_v \leftarrow \text{MUX}((y_s - y_u + x_u - x_v), P_v, 0)$
4: **end for**
```

a Hamiltonian path from s to t, the algorithm builds this path one edge at a time (this is in fact the situation illustrated in Figure 1). But if the diameter of G_U is known to be at most d, then $2d$ iterations suffices. For social networks, the maximum diameter is effectively a constant [8], [20].

If Alice and Bob are not particularly interested in connectivity once the shortest path is longer than a threshold d_t, they can stop the protocol after $2d_t$ iterations. The algorithm may also correctly answer "yes" even if the path is longer than d_t. This is because consecutive pieces of the path that lie completely in Alice's graph or Bob's graph do not require communication proportional to their length. For example, if the entire path is inside Alice's graph, she recognizes this after the initial connected components algorithm.

D. *Correctness of the Algorithm*

Correctness follows from the following two lemmas (applying them to the case $v = t$).

Lemma 1. If any iteration of the algorithm sets $y_v = y_s$ or $x_v = x_s$, then there is a path in G_U from s to v

Proof: Proof is by induction on number of iterations. By assumption the arrays are initialized so that $x_v = x_s$ only if there is a s-v path in G_a, hence in G_U. By observations noted above and the induction hypothesis, at each iteration $P_v = 0$ (and hence $y_v \leftarrow y_s$) only if there is a path in G_U from s to some node u and a path in G_b from u to v. An identical argument applies to y_v. ■

Lemma 2. If there is a path of length d in G_U from s to v, then after $2d - 1$ iterations, $y_v = y_s$ and after $2d$ iterations, $x_v = x_s$.

Proof: Proof is by induction on d. At iteration 0, before any communication, the arrays are initialized so that $x_v = x_s$ if and only if there is a path in graph G_a. Also, $y_v = y_s$ if and only if there is a path from s to v in graph G_b. Suppose node v has an edge (a path of length 1) to s in graph G_a. Then $x_v = x_s$ after the initialization. In step 1, line 3 of algorithm 1 will set $P_v = 0$, for instance when $u = v$. This then sets $y_v = y_s$ in the following update shown in (1). By a similar argument, if node v is connected to node s in graph G_b, then $y_v = y_s$ after initialization and step 2 sets $x_v = x_s$.

In any iteration, if y_v is set to y_s then y_w for all nodes w in v's current connected component (in G_b) are set to y_s. This is because the node u that sets $P_v = 0$, also sets $P_w = 0$ for each w known to be in v's component in G_b.

Suppose there is a path of length d between node s and node v in G_U. Let $su_1u_2 \cdots u_{d-1}v$ be such a path. By the induction hypothesis $y_{u_{d-1}} = y_s$ after step $2d-3$ and $x_{u_{d-1}} = x_s$ after step $2d-2$. If edge $u_{d-1}v$ is in graph G_b, then $y_{u_{d-1}} = y_v$ after the algorithm initialization. Then, but the argument in the previous paragraph, y_v was set to y_s when $y_{u_{d-1}}$ was set to y_s, no later than step $2d-3$. Otherwise, if edge $u_{d-1}v$ is in graph G_a, we have $x_{u_{d-1}} = x_v$ after the algorithm initialization. By the argument in the previous paragraph, we know x_v is set to x_s when $x_{u_{d-1}}$ is, no later than step $2d-2$ by the induction hypothesis. Consider step $2d-1$. Line 3 of algorithm 1 will set $P_v = 0$. Plugging in v for u in this line gives $x_s - x_v + y_v - y_v = 0$. This is because $x_s = x_v$ as we just argued. Therefore, $y_v = y_s$ after step $2d-1$. A similar argument shows $x_v = x_s$ after step $2d$. ■

Each iteration of the algorithm requires $O(n^2)$ MUX operations; however, for each u, all updates $P_v \leftarrow \text{MUX}((x_s - x_u + y_u - y_v), P_v, 0)$ can be done in parallel, so only $O(n)$ rounds of communication are required, with $O(n)$ communication in each round. Security follows from the security and composability of the secure circuit evaluation protocol. Thus we have

Theorem IV.1. Given G_a, G_b, s, t as above, if the diameter of $G_a \cup G_b$ is at most d, two parties can securely compute the existence of an s-t path in $G_a \cup G_b$ with $O(dn)$ communication rounds, $O(n^2)$ communication, and $O(dn^2)$ secure circuit evaluations.

E. Hiding Vertex Names

We now extend our algorithm to the case where the two parties do not have a shared, publicly known vertex set, and neither party should reveal anything about its vertex set (except for the common source/destination vertices s, t).

Instead of having variables x_u indexed by vertex names, we have two secret-shared arrays $x_i, \hat{x}_i$ where i is just an array index that goes from 0 to $n-1$ (where n is a reasonable upper bound on the maximum number of nodes in G_U; $\hat{x}_i$ is the name of the i^{th} vertex and x_i is the label of its connected component in Alice's graph. The other party (Bob) similarly has $y_i, \hat{y}_i$. A node u might exist in both graphs, but at unrelated positions; then $\hat{x}_i = \hat{y}_j = u$ with no particular relationship between i and j. Another node v might exist in only one graph; $\hat{x}_i = v$ while no $\hat{y}_j$ is equal to v. The parties do not want to reveal any information about which (or even how many) nodes are in their graphs. They pad their arrays with dummy nodes which are not adjacent to anything, making each array size exactly n. The special nodes s, t must be put at known positions in the array, which might as well be 0 and 1.

As stated before, we assume that the participants agree on some normalized naming convention, so that the same "individual" has exactly the same name for Alice and Bob if it does appear in both graphs. This might be a genuine practical difficulty in some situations, but would not be a problem if each node is tagged from some commonly used set of identifiers. The parties must also agree on some standard way to represent such names as elements of a finite field of integers modulo N.

Assuming this, we compute another secret-shared array y', which effectively permutes y to be aligned with x. If y_i refers to the same vertex as x_j, then $y'_j = y_i$; if $\hat{x}_j$ does not exist in Alice's graph, then $y'_j = 0$. This array is computed as follows:

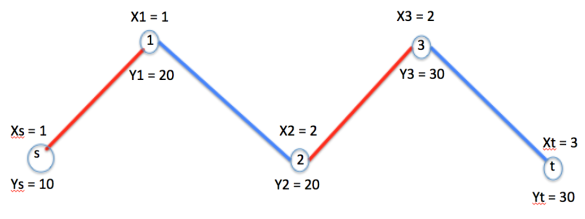
```
for $j$ do
    $y'_j \leftarrow 0$
    for $i$ do
        $y'_j \leftarrow y'_j + \text{MUX}(\hat{x}_j - \hat{y}_i,\ 0,\ y_i)$
    end for
end for
```

Then the parties compute shares of P_k as

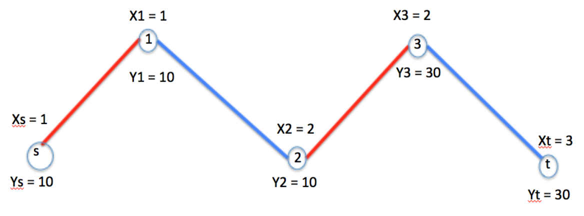
```
$P_k \leftarrow 1$
for $j$ do
    $P_k \leftarrow \text{MUX}(x_s - x_j + y'_j - y_k,\ P_k,\ 0)$
end for
```

We restrict the vertex labels as before and note that 0 is never used as a label, i.e. all x_i and y_i are non-zero. Thus if node $u = \hat{x}_j$ is in both graphs, y'_j is nonzero. It is the label of u in Bob's graph. Then we have $x_s = x_j$ if there is a path from s to u in Alice's graph, and $y'_j = y_k$ if (and only if) there is a path from u to $\hat{y}_k$ in Bob's graph, giving $P_k = 0$ as required. We have again guaranteed that $x_s - x_j$ and $y'_j - y_k$ cannot be additive inverses mod N unless they are both zero. Now Alice uses each P_k to update each y_k just as before; Alice does not need to know which (if any) of her vertices corresponds to Bob's y_k. Finally Alice has to recompute y', since y has changed. The parties compute an analogous x' array for the iterations in which the roles of x, y are reversed.

The proof of correctness carries over to the vertex-hiding version, since all previous properties of $x_s - x_j + y_j - y_k$ now apply to $x_s - x_j + y'_j - y_k$. Asymptotic communication complexity is the same since the shared computation of y' requires the same amount of communication as the shared computation of P.



(a)



(b)

Figure 1. An example of the low-trust s-t connectivity algorithm. Alice's edges are red, Bob's edges blue. (a) The first iteration gives $P_1 = 0$ (since $x_s - x_1 + y_1 - y_1 = 0$) and $P_2 = 0$ (since $x_s - x_1 + y_1 - y_2 = 0$), which sets $y_1 \leftarrow 10, y_2 \leftarrow 10$, marking these nodes as reachable from s. (b) The second iteration gives $P_2 = 0$ (since $y_s - y_1 + x_1 - x_2 = 0$) and $P_3 = 0$ (since $y_s - y_2 + x_2 - x_3 = 0$), which sets $x_2 \leftarrow 1, x_3 \leftarrow 1$, marking these nodes as reachable from s.

V. Acknowledgements

This work was funded by the Sandia National Laboratories LDRD (Laboratory-Directed Research and Development) program. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] J. Brickell and V. Shmatikov, "Privacy-preserving graph algorithms in the semi-honest model," in *Advances in Cryptology - ASIACRYPT 2005*, ser. Lecture Notes in Computer Science, B. Roy, Ed. Springer Berlin Heidelberg, 2005, vol. 3788, pp. 236–252.

[2] M. M. Prabhakaran and A. Sahai, *Secure Multi-Party Computation*. IOS press, 2013, vol. 10.

[3] C. Dwork, "Differential privacy: A survey of results," in *Theory and Applications of Models of Computation*. Springer, 2008, pp. 1–19.

[4] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center virtualization: a survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 909–927, 2013.

[5] A.-C. Orgerie, M. Dias de Assuncao, and L. Lefevre, "A survey on techniques for improving the energy efficiency of large scale distributed systems," *ACM Computing Surveys*, vol. 46, no. 4, 2014.

[6] A. Hajnal, W. Maass, and G. Turán, "On the communication complexity of graph properties," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 1988, pp. 186–191.

[7] "Amazon elastic compute cloud (ec2) – scalable cloud hosting," http://aws.amazon.com/ec2/, accessed October 17, 2014.

[8] E. David and K. Jon, *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. New York, NY, USA: Cambridge University Press, 2010.

[9] W. Aiello, F. Chung, and L. Lu, "A random graph model for power law graphs," *Experimental Mathematics*, vol. 10, no. 1, 2001.

[10] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: mining peta-scale graphs," *Knowledge and Information Systems*, vol. 27, no. 2, pp. 303–325, May 2011, special issue: best papers of the fifth international conference on advanced data mining and applications (ADMA 2009).

[11] P. Erdös and A. Rényi, "On the evolution of random graphs," *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, vol. 5, pp. 17–61, 1960.

[12] V. Boginski, S. Butenko, and P. Pardalos, "On structural properties of the market graph," in *Innovation in Financial and Economic Networks*, A. Nagurney, Ed. London: Edward Elgar Publishers, 2003, pp. 29–45.

[13] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos, "Power laws and the AS-level internet topology," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 514–524, 2003.

[14] F. Chung and L. Lu, "Connected components in random graphs with given expected degree sequences," *Annals of Combinatorics*, vol. 6, pp. 135–145, 2002.

[15] E. Kushilevitz and N. Nisan, *Communication complexity*. Cambridge University Press, 1997.

[16] B. Kalyanasundaram and G. Schnitger, "The probabilistic communication complexity of set intersection," *SIAM Journal on Discrete Mathematics*, vol. 5, no. 4, pp. 545–557, November 1992.

[17] J. Håstad and A. Wigderson, "The randomized complexity of set disjointness," *Theory of Computing*, vol. 3, pp. 211–219, 2007, online only journal.

[18] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[19] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 1988, pp. 1–10.

[20] U. Kang, M. McGlohon, L. Akoglu, and C. Faloutsos, "Patterns on the connected components of terabyte-scale graphs," in *ICDM*, 2010, pp. 875–880.