Efficient connection-finding with FinTracer

Michael Brand

November 23, 2023

Executive Summary

We described a new algorithm for finding source-destination pairs using only FinTracer machinery that works with close-to-optimal complexity on sparse graphs.

1 Introduction

The key operational difference between FinTracer, the algorithm at the core of the Purgles Platform solution, and its predecessor Asterisk is that while both ostensibly address the same problem, of examining which accounts in a target set B are connected to accounts in a source set A via edges of a directed graph G, Asterisk also provides the information of which specific element of A is connected to which specific elements of B.

FinTracer, by omitting this extra information, was able to use much faster algorithms, but the price is that for those wishing to find the extra connectivity information there is no direct solution.

In [**?**], the best algorithm proposed for this works in $\mathrm{O}(\min(A', B'))$ FinTracer queries, where $A' \subseteq A$ is the subset of the source set that is connected to B and $B' \subseteq B$ is the subset of the destination set that is connected to A.

Over a general G, this is best possible, as can be shown by a simple information-theoretic consideration. The full information of which of $|A|$ elements connects to which $|B|$ other elements requires in the general case $|A| \times |B|$ bits to communicate. Given that each execution of FinTracer provides at most $\max(|A|, |B|)$ bits, we can conclude that at least $\min(|A|, |B|)$ executions are required. In a general graph, $|A'|$ has the same order of magnitude as $|A|$ and $|B'|$ has the same order of magnitude as $|B|$, so this translates to the desired $\mathrm{O}(\min(|A'|, |B'|))$.

We will show, however, that one can do better in the realistic case, where the graph can be assumed to be sparse.

Also, unlike the previously-described method, which utilised the ability to perform a reversed FinTracer run (finding A', instead of finding B'), this method only uses forward runs.

For a forward-only algorithm, assuming no account in $|B|$ is connected to more than s accounts in $|A|$, the simple information bound on the number of queries required is $s \log |A|$. We will show an algorithm working in an expected $\mathrm{O}(s(\log |A| + \log |B|))$ queries. Unless $|B| \gg |A|$ (which is not the typical case in real use), this is best possible.[1]

2 A simplified solution

We begin with a slightly simplified case.

Consider the following algorithm.

We repeatedly invoke FinTracer, each time setting to a positive tag value only a subset of A. The elements of this subset are chosen randomly and independently, each chosen with probability $\alpha = \frac{1}{s+1}$. We continue, until for each account in B it is known exactly which accounts in A connect to it.

[1] Even when $|B| \gg |A|$, I suspect this algorithm is closer to optimal than the difference between the bounds would suggest. Both the calculation of the algorithm's complexity and the information bound are very rough bounds.

In order to estimate how many queries the algorithm will go through before terminating, let us consider what is required in order to fully resolve the connectivity matrix.

Let a be some account in A, let b be some account in B, and let S_b be the set of accounts in A connecting to b, noting that $|S_b| \leq s$.

One way to determine definitively that $a \notin S_b$ is to run FinTracer while setting the tag value for a as nonzero, but the tag value for all accounts in S_b to be zero. The output tag for b will be zero, proving the claim.

For any such specific a and b, the probability that a will not be definitively excluded from S_b after l queries is $\left(1-\alpha(1-\alpha)^{|S_b|}\right)^l$. This can be lower-bounded by $(1-\alpha(1-\alpha)^s)^l$.

On the other hand, suppose that $a \in S_b$. What would be required to prove this? If all accounts that are not in S_b have already been excluded, to definitively associate a with S_b requires executing FinTracer while setting that tag of a to nonzero, and the tag value for all other elements of S_b to zero.

For any such specific a and b, the probability that a will not be definitively included in S_b after l queries (assuming all accounts not in S_b are definitively excluded), is $\left(1-\alpha(1-\alpha)^{|S_b|-1}\right)^l$. This can also be lower-bounded by the same bound as before: $(1-\alpha(1-\alpha)^s)^l$.

The expected total number for all (a,b) pairs, such that after l queries a has neither been definitively excluded from S_b nor definitively included in S_b (if all non-S_b accounts are excluded) is therefore upper bounded by $|A||B|\,(1-\alpha(1-\alpha)^s)^l$.

Our choice of $\alpha = \frac{1}{s+1}$ brings this upper bound to a minimum, which for a large s is approximately $|A|B|\left(1-\frac{1}{es}\right)^l$, where e is Euler's constant.

This expectation, in turn, is an upper bound for the probability that the algorithm will not terminate after l rounds. Suppose we want to set this termination probability to some constant value p. The number of queries needed, assuming a large s, is approximately

$$l \approx -\frac{\log(|A||B|/p)}{\log\left(1-\frac{1}{es}\right)} \approx es\left(\log|A| + \log|B| - \log p\right),$$

where all logarithms, here and throughout, are natural logarithms unless otherwise specified.

This calculation shows that regardless what termination probability is required, the algorithm will always terminate in $\mathrm{O}\left(s(\log|A| + \log|B|)\right)$ queries with that probability.

It also bounds the expected number of rounds required for termination. To see this, set p to some constant, for example $1/3$. If a probabilistic algorithm has probability $1-p$ of terminating successfully, the expected number of runs until success is $\frac{1}{1-p}$. So, the expected number of FinTracer queries until termination is clearly bounded by $1.5l$. This multiplication by a constant does not impact complexity.

3 The full algorithm

The algorithm described above makes one simplifying assumption, namely that the value of s is known. This allows us to choose $\alpha = \frac{1}{s+1}$.

Let $\mathcal{A}_s$ be our simplified algorithm, with the subscript indicating the value of s used. It will choose each element of A with probability

$$\alpha_s = \frac{1}{s+1}$$

and will run for

$$l_s = es\left(\log|A| + \log|B| + \log 3\right)$$

rounds, ensuring that if the true $\max_{b\in B}|S_b|$ is at most s, the algorithm will succeed with probability at least $1-p = 2/3$.

Note 3.1. In implementing this algorithm, it is possible to either check for the termination conditions after every query or only at the end of l_s queries. Early termination can save on the number of queries required, but it is unlikely to impact their overall complexity. Moreover, such early termination requires much more frequent checks of the termination criteria, which, despite not requiring any communication or encryption, are not very lightweight. If we were to compute the *computational* complexity of the algorithm, rather than the complexity of the expected number of queries, such frequent checks would have increased that complexity. In practice, it is likely that the algorithm's communication portions will remain its heaviest bits by far even so, so early termination may be advisable.

The full algorithm is now as in Algorithm 1.

Algorithm 1 Pair finding algorithm

```
s ← 1
while Termination conditions not met do
    Run algorithm A_s.
    s ← 2s.
end while
```

Let us assume that the real $\max_{b\in B} |S_b|$ is some value s_0, which we'll round up, for convenience, to the nearest power of 2 (without this impacting complexity). Let $i_0 = \log_2 s_0$.

To compute the expected number of queries until termination, let

$$c = e\left(\log|A| + \log|B| + \log 3\right),$$

and note that, as proved before, the probability that after cs queries the algorithm $\mathcal{A}_s$ will not terminate successfully is upper-bounded by $1/3$, assuming $s_0 \le s$.

To measure the expected number of queries, we take the conservative assumption that when running $\mathcal{A}_s$ for any $s < s_0$, the algorithm will never be successful. This costs us in $\sum_{i=0}^{i_0-1} c2^i < cs_0$ queries.

After these initial queries, each run of $\mathcal{A}_s$ costs in an additional cs queries, but has at the very least probability of $1 - p = 2/3$ of reaching termination conditions (and quite likely much better).

The total expected number of queries is therefore bounded by

$$cs_0 + \sum_{i=i_0}^{\infty} c2^i p^{i-i_0} = cs_0 + 3cs_0 = 4cs_0.$$

We conclude that the full algorithm, working without prior knowledge of s_0, completes after an expected $\mathrm{O}\left(s(\log|A| + \log|B|)\right)$ queries, as claimed.