Detecting Missing and Spurious Edges in Large, Dense Networks Using Parallel Computing

Samuel Coolidge, sam.r.coolidge@gmail.com
Dan Simon, des480@nyu.edu
Dennis Shasha, shasha@cims.nyu.edu
Technical Report NYU/CIMS/TR2016-986

Abstract

Certain pairs of drugs can cause death from their interaction. Knowledge of such interactions is held in drug interaction networks. The problem is that such networks may miss interactions that should be present and may include interactions that should be absent. Clearly, such information is valuable. Drug interaction networks are not unique in this regard. The same holds for protein-protein interaction networks, ecological networks, and many others. Improving the quality of such networks often requires a ground truth analysis (e.g. more experiments) but Roger Guimerá, Marta Sales-Prado, and their colleagues have shown in several papers that a structural analysis of networks can lead to predictions of missing and spurious edges that can improve those networks. Our contribution in this paper and the accompanying software is to create a program implementing their algorithmic ideas that is parallelizable and easy to modify for researchers who wish to try out new ideas. Our software can be found at https://github.com/samcoolidge/network.

Motivation and Problem

In *Missing and Spurious Interactions and the Reconstruction of Complex Networks,* authors Roger Guimerá and Marta Sales-Prado present a general mathematical framework for identifying errors in a network. Their implementation of this framework yields excellent results when compared to previous attempts at network reconstruction. The main limitations of their program are (i) it is slow when reconstructing networks that are substantially large or dense and (ii) it is hard to understand**.** This report describes a program that provides an improvement in efficiency by parallelizing the most laborious portions of the reconstruction process and has been engineered to be easier to modify.

Intuitive Theory

The Guimera-Sales-Prado algorithm relies only on the assumption that any network conforms to the structure of a stochastic block model. In a stochastic block model the nodes are separated into partitions (hereafter called groups), and the probability of an edge between two nodes is determined strictly by the groups to which these nodes belong. Therefore, a network with n nodes is defined by three data structures:

1) A scalar value k denoting the number of groups in the network

2) An $n \times 1$ vector v where each entry v_i denotes the group associated with node i

3) A $k \times k$ probability matrix M, where M_{ij} contains the probability that a node in group i is connected to a node in group j

The justification for using the stochastic block model is that it holds empirically in real network connections. For example, nodes in many networks are often organized into communities in which interaction within a group is significantly more common than interaction between groups. Family structure in a social network of animals is an obvious example. Nodes in a network may also have specified roles that can increase the prevalence of connections between roles. For example, in a protein interaction network, some proteins are more likely to have lasting physical contact with proteins of a different type than their own. This inter-community connection also holds for drug interaction networks.

Complex networks typically have many groupings that determine the links between nodes. For example, a human social network may depend on partitions defined by birthplace, education, ethnicity, and age. These partitions may be very different from one another but all contribute to the likelihood of a network link and all are captured in some block model. The general approach of the algorithm is to estimate the node partitions (i.e. the groups) of a given network by sampling from the space of stochastic block models. We can then estimate the reliability of a link (i.e. a graph edge) given our understanding of the underlying group structures.

Guimera-Sales-Prado Algorithm and Our Implementation

To estimate the link reliabilities in a network we use the following procedure. A network of n nodes is defined to be an $n \times n$ adjacency matrix with $A_{ij} = 1$ if there is a edge between two nodes and $A_{ij} = 0$ if there is no such edge. Given our data we assume that there is a “true” network A' that differs from the observed network A^O. The input for the algorithm is this matrix A^O, and we estimate the probability that each possible link in the observed network exists in the true network. The reliability of a link between nodes i and j is derived from the definition of a stochastic block model and given by the formula:

$$(1) \qquad R_{ij} = \frac{1}{Z} \sum_{p \in P} \left(\frac{l^O_{\sigma_i \sigma_j} + 1}{r_{\sigma_i \sigma_j} + 2} \right) \exp\left[-H(p)\right]$$

Intuitively, this is a sum over each partition p in the space of all possible partitions P. Each σ_i represents the group of node i and $l^O_{\alpha\beta}$ is the number of links in the observed network between groups α and β.The term $r_{\alpha\beta}$ is the maximum number of links between groups α and β and, $H(p)$ is an “entropy” function of the partition given by:

$$(2) \qquad H(p) = \sum_{\alpha \le \beta} \left[\ln\left(r_{\alpha\beta} + 1\right) + \ln \binom{r_{\alpha\beta}}{l^O_{\alpha\beta}} \right]$$

and

$$Z = \exp\left[-H(p)\right]$$

Each partition provides the probability of a link between two nodes as determined by the proportion of existing links to possible links between the groups of these nodes. The reliability is calculated as an average of these probabilities weighted by the relevance of the partition.

In practice it is impossible to compute a sum over all possible partitions as the number of partitions exceeds 1×10^9 even for networks as small as 15 nodes. Much of the algorithm consists of finding only the partitions that contribute significantly to the sum. A Markov chain Monte Carlo method, specifically a Metropolis-Hastings algorithm, is used to find the relevant partitions. The N nodes are placed into one of N groups and then one at a time are randomly moved to a different group. At each step the change in $H(p)$ is calculated, and if the H-value decreases then the swap of groups is accepted. If the value of $H(p)$ increases then the swap is accepted with probability $\exp\left[-H(p)\right]$. The algorithm consists of three steps. First we find a “decorrelation factor” that will determine the number of node swaps needed to find uncorrelated partitions. Then we find an equilibrium partition as $H(p)$ decreases from its initial value to its equilibrium value, and finally we use this equilibrium partition to take a sample from the partition space.

Decorrelation Factor

First we must find the number of node swaps that must be attempted in order to create partitions that are uncorrelated. The accuracy of the algorithm relies on the fact that each partition provides unique information when calculating link reliability. To do this we create a starting partition with one of n nodes in each of n groups. We then perform a total of $n * x_1$ node swaps, where $x_1 = \left\lfloor \frac{n}{20} \right\rfloor$ and record the normalized mutual information y_1 of the resulting partition **.** We also perform $n * x_2$ node swaps, where $x_2 = n * \left\lfloor \frac{n}{4} \right\rfloor$ and record normalized mutual information y_2. Then we solve the system:

$$y_1 - a = (1 - a) * \exp\left(\frac{-x_1}{b}\right)$$

$$y_2 - a = (1 - a) * \exp\left(\frac{-x_2}{b}\right)$$

The value of b is recorded, and we repeat this process 10 times and average the results while discarding any outlier iterations. The result is a "decorrelation factor" d that will be a critical input for our Metropolis-Hastings function. This method is used as a heuristic by Guimerá Et al. to find a partitions that have suitably low mutual information. From this point forward most of the work of the algorithm is done using a function called factor_MC_step() that will execute the Metropolis Hastings procedure on an input partition, attempting $n * d$ node swaps to ensure each new partition is sufficiently different from those previous.

Equilibrium Partition

To find an equilibrium partition we create small samples of partitions until the entropy (given in equation (2)) of these samples reaches equilibrium. Specifically, we execute our factor_MC_step() function 20 times and record the H-value of each resulting partition. For every sample of 20 H-values we calculate the mean and standard deviation and compare them to previous samples. If the difference between two sample means is less than their pooled standard error, we consider one of five equilibration checks to be complete. If five samples in a row satisfy the equilibrium check when compared to the first sample that satisfied our check, we can be confident that entropy function has reached an equilibrium as opposed to some local minimum. We take the partition that yielded the final H-value and call it our equilibrium partition. If any new sample has a significantly different mean H-value, we compare to this new sample and all previous successful checks are ignored.

Sampling

Once we have an equilibrium partition, we can generate a large sample of partitions with which we calculate our link reliabilities. We execute our factor_MC_step() function with our equilibrium partition as an input. The result is a new uncorrelated partition that is then fed back into factor_MC_step(). This loop is completed 10,000 times to generate a sample. We calculate the reliabilities of each link in the network using equation (1), which is a sum over the sample of partitions. The reliabilities of links present in the observed network are sorted from lowest to highest, indicating that the links at the top of the list are likely to be spurious. Conversely, the reliabilities of the links missing from the observed network are sorted from highest to lowest, indicating that those at the top are likely to be present in the true network. The output of the algorithm is these lists which pair each link with their respective reliability.

Parallel Design

The objective of our reimplementation of this network reconstruction algorithm is to improve the algorithm's execution time through parallel computing. The experiments in this paper uses 100 processors, though improvements are noticeable with as few as 2-3 CPUs and efficiency would continue to increase with additional processing power. The most time-consuming portions of the algorithm are finding the equilibrium partition and then constructing a sample of 10,000 uncorrelated partitions.

Equilibration in Parallel

Using a single processor, we collect a sample of 100 equilibration times for two smaller networks. The data is shown below

Karate Club Social Network – 34 Nodes
```
Minimum Time:  0.155271053314 seconds
Maximum Time:  2.13078403473 seconds
Mean Time:  0.573533740044 seconds
Standard Deviation:  0.384631167387 seconds
```

Air Transportation Network – 133 Nodes
```
Minimum Time:  9.968334198 seconds
Maximum Time:  43.5213611126 seconds
Mean Time:  21.4101707292 seconds
Standard Deviation:  7.64734197381 seconds
```

The variation in equilibration times is detrimental to the efficiency of the sequential code, but can be leveraged by parallel processing. Our algorithm initiates 100 threads that each performs a Metropolis-Hastings algorithm attempting to find an equilibrium partition. Whichever thread finishes first is chosen and the others discarded. The accuracy is unaffected by choosing the fastest thread because the equilibrium conditions described in the previous sections are sufficient to ensure a high quality partition. This portion of the algorithm is sped up on average by a factor of 2-4 depending on the network, and reduces the standard deviation.

Sampling in Parallel

The most significant speed improvements in the new algorithm are achieved during the creation of a sample of 10,000 partitions. Instead of sampling sequentially from the equilibrium partition, we make copies of the starting partition equal to the number of processors p we are running in parallel. Then each processor performs a sequential sampling of size 10,000 / p. These threads complete simultaneously as uncorrelated partitions are found in a constant amount of time since the time of our factor_MC_step() function depends only on the number of nodes swaps executed. Furthermore, since we know all partitions are uncorrelated, accuracy is not effected by running in parallel. All communication between nodes and processors is done with MPI, specifically MPI4py.

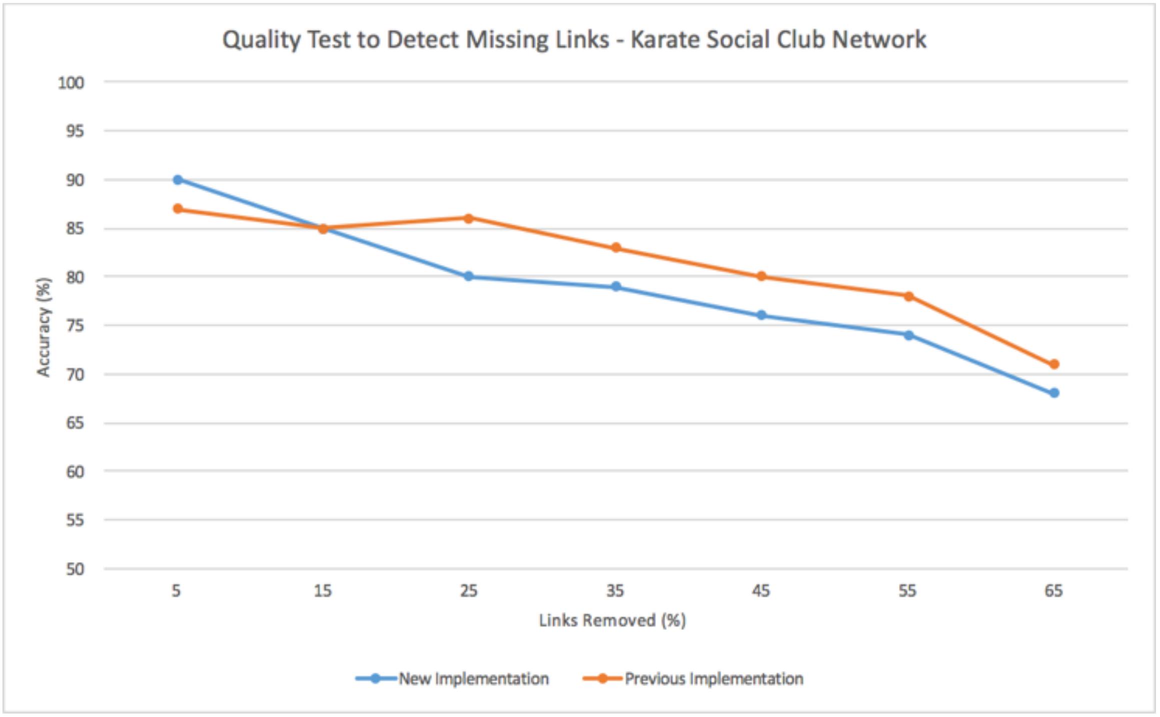
Dependencies

In addition to speed improvements, we believe that our implementation of the reconstruction algorithm is accessible and easy to modify. This is essential so that more research and improvements can be made to this algorithm. Our program relies only on the following Python packages: NumPy, SciPy, MPI4py and Cython. The function that executes the Metropolis-Hastings algorithm and calculates the entropy of partitions is by far the most complex and laborious part of our algorithm. This code has been simplified and is written with the Cython extension. It is accessible for Python users but still compiles and runs in C so no efficiency is sacrificed.

Experiments

Quality Comparison

While our network reconstruction algorithm is conceptually the same as that of Guimerá and Sales-Prado, the data structures and computations differ. Therefore, it is necessary to compare the qualitative results of each. We test the quality of our new implementation of the algorithm on two networks and compare it to the previous implementation.



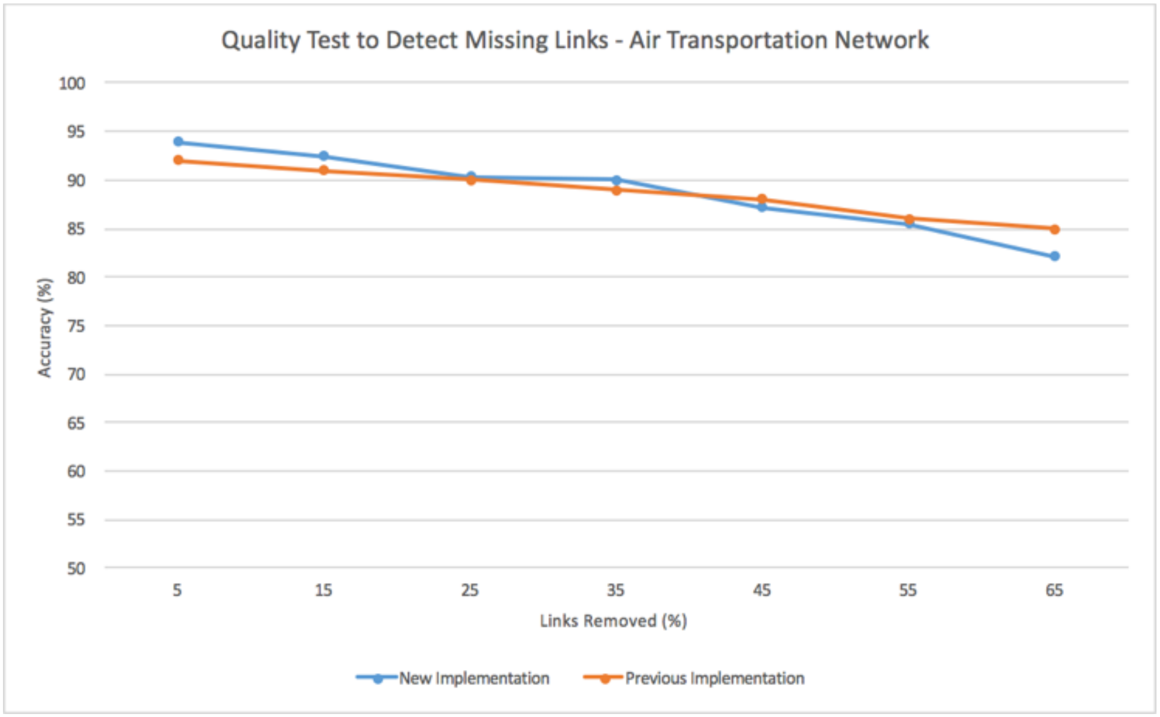
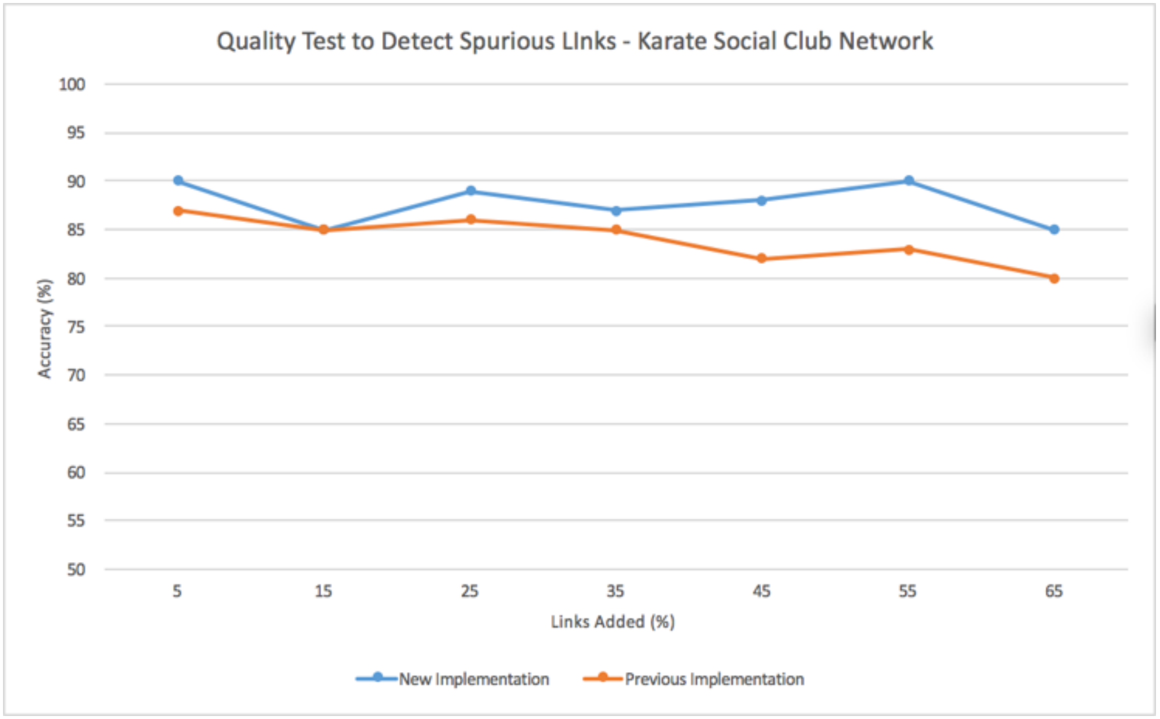


Fig. 1. We test the quality of missing link reliabilities by removing a percentage of links from a true network to simulate data error. We then calculate the probability that a true negative ($A^{o}_{ij} = 0$ and $A'_{ij} = 0$) has a lower link reliability than a false negative ($A^{o}_{ij} = 0$ and $A'_{ij} = 1$) and call this the accuracy of the reconstruction. Networks with with a percentage of removed links ranging from 5% to 65% and the corresponding accuracy of the reconstruction are displayed.



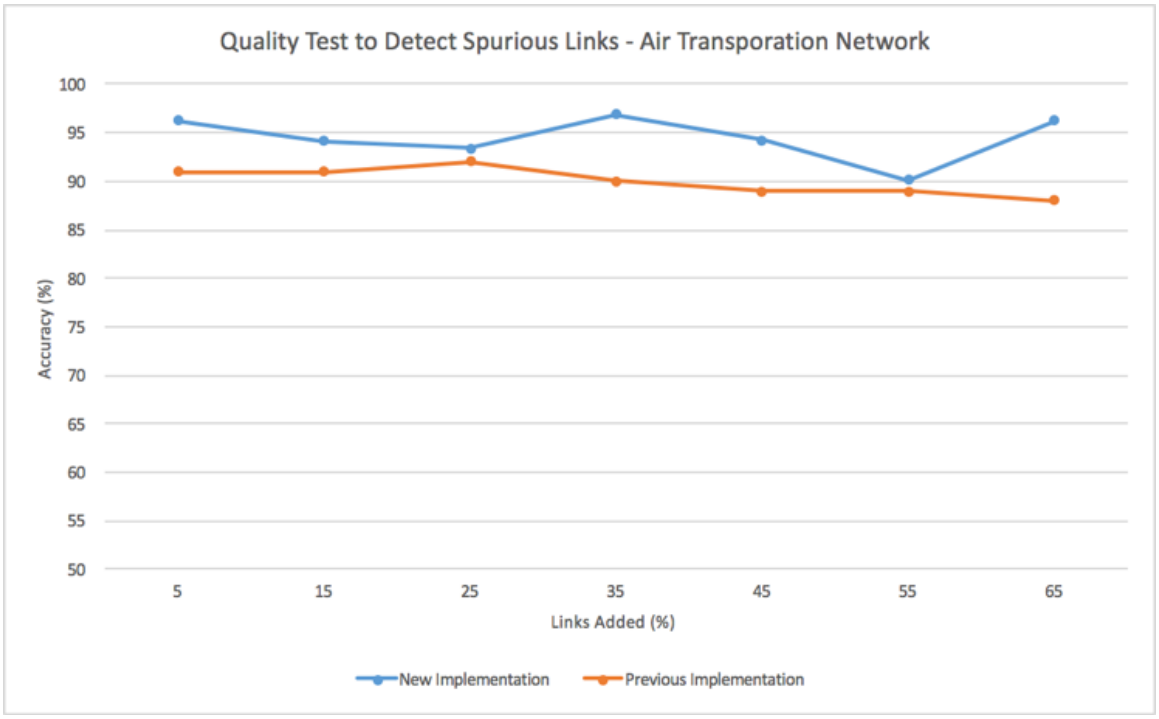


Fig. 2. We test quality of spurious link reliabilities by adding a percentage of links to a true network. We then calculate the probability that a true positive ($A^{O}_{ij} = 1$ and $A'_{ij} = 1$) has a higher link reliability than a false positive ($A^{O}_{ij} = 1$ and $A'_{ij} = 0$) and call this the accuracy of the reconstruction. Networks with with a percentage of added links ranging from 5% to 65% and the corresponding accuracy of the reconstruction are displayed.

Thus, the two implementations enjoy roughly the same quality. The new implementation is marginally more effective at detecting both missing and spurious links in the air transportation network, but marginally less effective at detecting missing links in the karate social club network.

Timing Comparison

To measure execution time improvements, we conducted timing tests on networks conforming to a stochastic block model. The networks were randomly generated while varying the parameters of network size and network density.

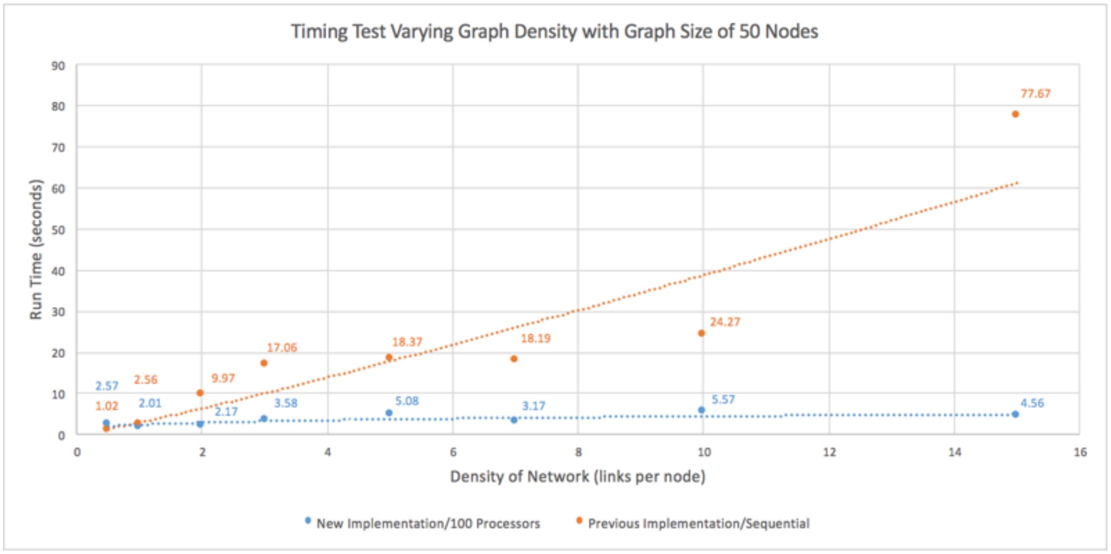


Fig. 3. Run time is measured while varying the density of a network of size 50 nodes. Density is measured as the ratio of links to the number of nodes. We compare the previous sequential implementation of the algorithm to the new parallel implementation (100 processors). Specific run time values and a trend-line are displayed. Parallelism helps more the denser the network.

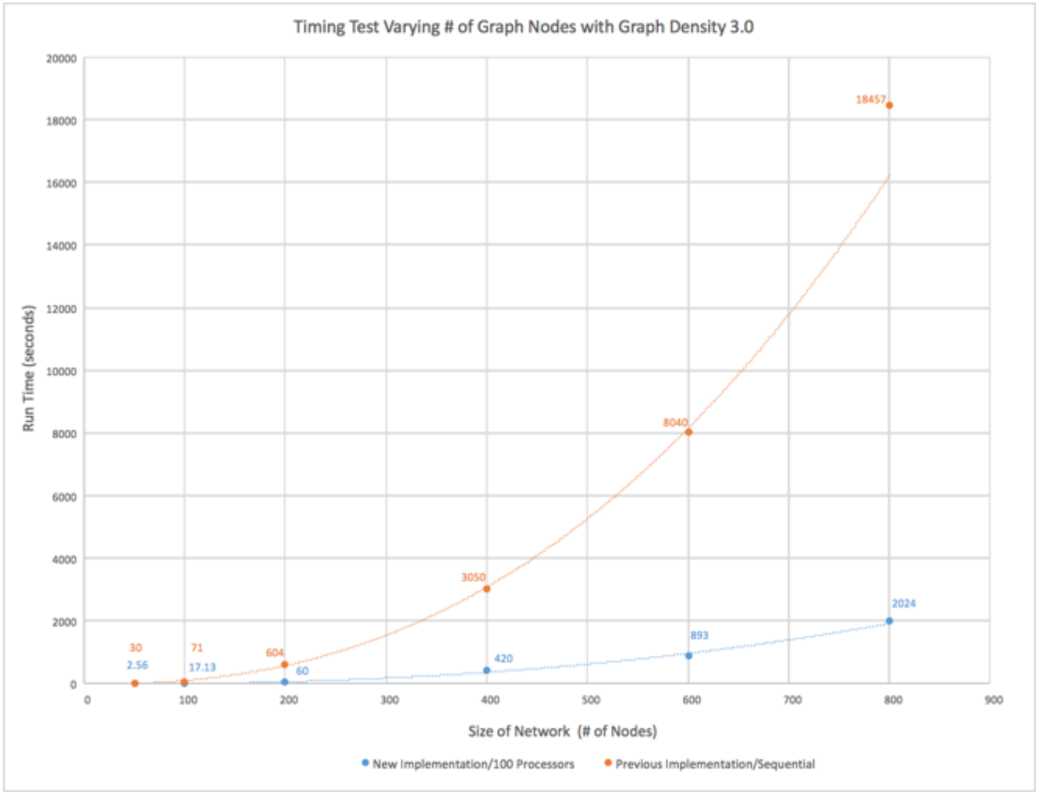


Fig. 4. Run time is measured while varying the size (# of nodes) of a network with density = 3.0. We compare the previous sequential implementation of the algorithm to the new parallel implementation (100 processors). Specific run time values and a trend-line are displayed. Parallelism helps more for larger networks.

Parallelization offers a negligible execution time improvement in small networks, especially those with very low density. However, parallelism shows significant benefits for large, dense networks. High density networks with a few thousand nodes would take days or weeks to reconstruct when using the previous implementation. The new code using 100 processors could cut down this run time by a factor of 10-20, expanding the set of networks to which this algorithm practically applies.

Processor Scaling

Finally, we investigate to what extent additional processing power improves efficiency. Run time can be modeled by the following function of the number of processors (p):

$$Run\ Time = \frac{Sequential\ Sampling\ Time}{p} + min\{Sequential\ Equilibration\ Times\}$$

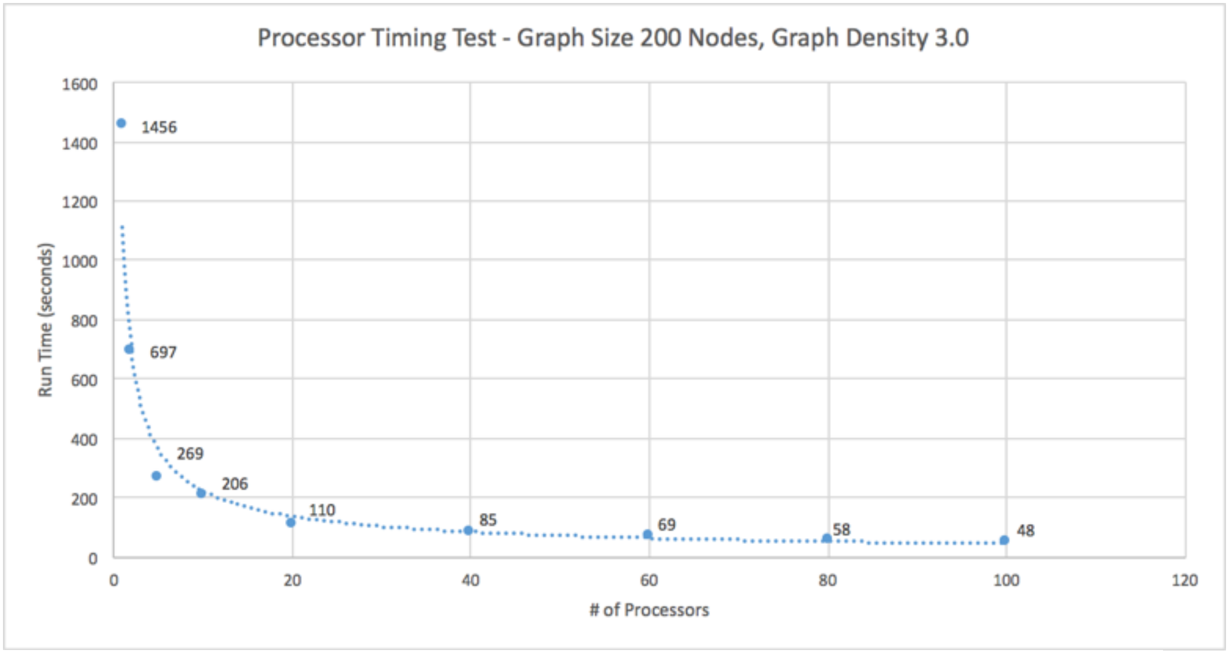


Fig. 5. Run time is measured while varying the number of processors used in parallel. Specific run time values and a trend-line are displayed.

Any variation caused is a result of different decorrelation steps or equilibration times. As additional processors are used during the sampling procedure, the minimum equilibration time becomes a larger portion of total run time. In this trial, minimum equilibration time is approximately equal to parallelized sampling time when 60 processors are used. Thus additional processors past 60 cannot do better than cut run time in half. Perhaps future research can be conducted on further improving the efficiency of the equilibration period.

Conclusion

Analysis of network structure can be useful in a variety of research and practical contexts, from biology and medicine to human interaction and sociology. However, network errors can arise both because of data collection failure or incomplete information. A computational approach that improves the reliability of network allows for the improvement in the quality of inferences made from a network by fixing links that are likely to be missing or spurious. It also can provide direction for data collection in incomplete networks by finding the missing links that are the most likely to exist. We have implemented a parallel version of the Guimera-Sales-Prado algorithm that shows substantial speedup over the original sequential algorithm in several contexts. The software is available at https://github.com/samcoolidge/network

Acknowledgements

We would like to thank Roger Guimerá for his helpful comments.
Support for this work has come partly from U.S. National Science Foundation grants MCB-1158273, IOS-1339362, and MCB-1412232. We greatly appreciate this support.