Identifiyng Inter-Genomic Repeats Using Fast, Approximate Betweenness Centrality

Jay Ghurye, Chris Hill, and Mihai Pop

University of Maryland, College Park
{alfred.hofmann,ursula.barth,ingrid.haas,frank.holzwarth,
anna.kramer,leonie.kunz,christine.reiss,nicole.sator,
erika.siebert-cole,peter.strasser,lncs}@springer.com
http://www.springer.com/lncs

Abstract. Assembling metagenomic data is a challenging task due to several reasons. One of the reasons that complicates assembly is the sequences shared between the genomes of multiple organisms. Such regions in genomes are called repeats. Repeats make assembly difficult because they link the unrelated sections of genomes. It has been shown that the number of reconstructions of genome grows exponentially with repeats. Existing approaches to identify repeats are inefficient and do not scale well as size of data increases. We propose an application of a core concept in social network analysis called betweenness centrality to metagenomic data to identify inter-genomic repeats. We study how approximation algorithms for betweenness centrality can be useful to find central nodes quickly and propose a novel approach to identify inter-genomic repeats accurately.

Keywords: Metagenomics, Algorithms, Centrality, Graph

1 Introduction

Metagenomics is a direct sequencing of DNA from all organisms in an environment without culturing. It is a culture-independent tool for studying environmental microorganisms. In addition to the information about taxonomic diversity (who is there), metagenomics gives insight into the physiology of the organisms present in the environment (what are they doing), through studying their genes. Two main goals to be achieved with metagenomics are finding new genes with desired biological activity (bioprospecting) and studying environmental microbes without the need to culture them.

Metagenomic data are considerably more complex. The assembly of metagenomic data is complicated due to several reasons. Few of them are: (i) widely different levels of representation for different organisms in a community; (ii) genomic variation between closely related organisms; (iii) conserved genomic regions shared by distantly related organims; and (iv) repetitive sequences within individual genomes. Genomic repeats are the major challenge when assembling isolated genomes and their effect is prominently seen in metagenomic datasets. It has been shown by Kingsford et al.[5] that the number of reconstructions grow exponentially with the number of repeats. Therefore it is impossible find one correct solution for assembly. There are several approaches such as [7] to find repeats in metagenomic data. The main drawback of these methods is that they do not scale well as number of nodes in assembly graph increase and prove very inefficient as number of nodes reaches to about a million. To tackle this problem, we explore some of the work in social network analysis, particularly in the area of node centrality. We make use of betweenness centrality to find out repeats in assembly graph.

2 Related Work

Betweenness Centrality

In graph theory and network analysis, indicators of centrality identify the most important vertices within a graph.Several metrics to measure centrality have been proposed. We make use of betweenness centrality. For a particular node, betweenness centrality is equal to the number of shortest paths from all vertices to all others that pass through that node. Formally, for a node v, it is defined by following expression:

$$g(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths passing through v.

Brandes[2] gives an exact algorithm for computing betweenness centrality of all the nodes that is based on solving a single source shortest path problem for each node. An SSSP computation from s produces a directed acyclic graph (DAG) encoding all shortest paths starting at s. By backward aggregation of counter values, the contributions of these paths to the betweenness counters can be computed in linear time. Brandes algorithm explains how to calculate σ_{st} on the fly during shortest path calculations: $\sigma_{ss} = 1$ and for $s \neq t, \sigma_{st} = \sum_{v \in pred(t)} \sigma_{sv}$ where $pred(t)$ is a multiset containing the immediate predecessors of t in the shortest path DAG. In a subsequent aggregation phase, the nodes are processed in reverse topological order, i.e. by non-decreasing distance from s. So we have:

$$\delta_s(v) = \sum_{w \in succ(v)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_s(w))$$

where $\delta_s(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$ and $succ(v)$ is an immediate successor of v in the shortest path DAG. This algorithm takes time $\Theta(mn)$ for unit edge weight graphs and $\Theta(mn + n^2 log(n))$ for weighted graphs. This algorithm does not scale well with large networks with millions of nodes and edges. Bader and Madduri[8] provide a massively parallel implementation of the exact algorithm. However this approach scales to about a few million nodes.

Betweenness Centrality Based on Sampling Nodes

Due to scalability limitations of exact betweenness centrality, several approximation algorithms for betweenness centrality have been proposed. Bader and Pich[1] provide an approximation algorithm by extrapolating from a subset of k starting nodes called pivots and using same aggregation strategy as exact beteeness algorithm. However, this algorithm overestimates the centrality of some unimportant nodes which lie near pivot. The approximation algorithm proposed by Geisberger et al. [3] comes over this drawback by changing the scheme for aggregating betweenness contributions so that nodes do not get profited just because they are near pivot. They define an estimator for betweenness centrality. This estimator is characterized by a length function $l : E \Rightarrow \mathbb{R}$ on the edges and a scaling function $f : [0, 1] \Rightarrow [0, 1]$. For a path $P = (e_1, e_2 ..., e_k)$ let $l(P) := \sum_{1 \leq i \leq k} l(e_i)$. Let n be the number of sample nodes chosen from a graph. The algorithm works as follows: In each iteration one of $2n$ shortest path searches is performed with uniform probability $1/2n$. An iteration can be a forward search in $G = (V, E)$ from a pivot $s \in V$ ($|V| = n$) or backward search from a pivot $t \in V$, that is a search from t in $(V, (v, u) : (u, v) \in E)$. For each shortest path of the form

$$P = (\overbrace{s, ..., v}^{Q}, .., t)$$

a scaled contribution is defined as follows:

$$\delta_P(v) := \begin{cases} \frac{f(l(Q)/l(P))}{\sigma_{st}} & \text{for a forward search} \\ \\ \frac{1-f(l(Q)/l(P))}{\sigma_{st}} & \text{for a backward search} \end{cases}$$

Overall, v receives contribution

$$\delta(v) := \delta_s(v) := \sum_{t \in V} \sum \{\delta_P(v) : P \in SP_{st}(v)\}$$

due to a forward search, and

$$\delta(v) := \delta_t(v) := \sum_{s \in V} \sum \{\delta_P(v) : P \in SP_{st}(v)\}$$

due to a backward search.

This algorithm is efficiently implemented using two techniques. First one is *linear scaling*, where the contribution of a node v depends linearly on the distance to the sample. Nearer the node is to the sample, lesser is its contribution counted. Linear scaling is implemented using a simple variant of aggregation scheme proposed in Brandes algorithm. Second technique is *bisection scaling* where the sample only contributes to the second half of a path. This means that for a vertex v, its contribution is considered only if it is at atleast half distance between s and t away from s in forward search or more than half a distance away from t in backward search.

Betweenness Centrality Based on Sampling Shortest Paths

There is another approximation algorithm was proposed by Matteo et al. [9] based on randomized sampling of shortest paths and offer probabilistic guarantees on the quality of approximation. This algorithm guarantees that all approximate values of betweenness for all vertices are within an additive factor $\epsilon \in (0, 1)$ from the real values with probability at least $1 - \delta$. To derive bounds on sample size, results from VC dimension theory[11] are used. The main advantage of this algorithm is that the sample size does not depend on the size of the graph, but only on the maximum number vertices in a shortest path which is called *vertex diameter*.

The Vapnik-Chernovenkis(VC) dimensions of a class of subset is defined on a set of points is a measure of expressiveness of such class. Given a probability distribution on the set of points, a finite bound on the VC-dimension of the class of subsets implies a bound on the number of random samples required to approximate the probability of each subset in the class with its empirical average. This technique is used to derive the expressiveness of the sample of shortest paths derived. For graph $G(V, E)$, let S_{all} be the union of T_v for all $v \in V$, where T_v is the set of all shortest path to which v is internal. VC dimension technique helps in choosing sample from S_{all} so that it gives probabilistic guarantees. It is shown that the upper bound on VC dimensions of S_{all} is given by:

$$VC(S_{all}) \leq \lfloor log_2(VD(G) - 2) \rfloor + 1$$

where $VD(G)$ is vertex diameter of G. Using this upper bound, sample size r can be estimated as follows:

$$r = \frac{c}{\epsilon^2}(\lfloor log_2(VD(G) - 2) \rfloor + 1 + ln\frac{1}{\delta}) \tag{1}$$

where c is a universal constant with value 0.5. It is guaranteed that betweenness centralities estimated using this sample size are within additive factor ϵ from real values with probability atleast $1 - \delta$.

Complete algorithm is sketched in algorithm 1. There are two crucial things in this algorithm. First is computing approximate vertex diameter. Calculating exact VD(G) will need to solve all pair shortest path problem. The complexity of this algorithm is $O(V^2 logV + VE)$ by Johnson's algorithm[4] and $\Theta(V^3)$ by Floyd-Warshall algorithm. This would be very slow. A faster linear time 2-approximation algorithm is proposed in [9] is used to calculate approximate vertex diameter.

Second is to sample shortest paths. They way a random shortest path from S_{uv} is inspired by Brandes algorithm[2]. Assume that u and v are connected. For each vertex w, let $P_u(w)$ be a subset of neighbors of w which are predecessors of w along the shortest path from u to w. Let p^* be the sampled shortest path that we build backwards starting from the endpoint v and by adding sampled predecessors before v. Initially $p^* = v$. Starting from v, we select it's predecessor $z \in P_u(v)$ using weighted random sampling. Each $z \in P_u(v)$ has probability

Algorithm 1 Algorithm to compute approximate betweenness centrality

```
1: **Input**:Graph $G(V, E)$ with $|V| = n, \epsilon, \delta \in (0, 1)$
2: **Output**: A set of approximate betweenness centrality for all nodes $v$ in $G$
3: **for** $v \in V$ **do**
4:     $b'(v) = 0$
5: $VD(G) \leftarrow$ getVertexDiameter($G$)
6: $r \leftarrow \frac{c}{\epsilon^2}(\lfloor log_2(VD(G) - 2)\rfloor + 1 + ln\frac{1}{\delta})$
7: **for** $i \leftarrow 1$ to $r$ **do**
8:     $(u, v) =$ SampleUniformVertexPair(V)
9:     $S_{uv} =$ ComputeAllShortestPath(u, v)
10:    **if** $S_{uv} \neq \{p_\phi\}$ **then**
11:        $t \leftarrow v$
12:        **while** $t \neq u$ **do**
13:            sample $z \in P_u(t)$ with probability $\sigma_{uz}/\sigma_{ut}$
14:            **if** $z \neq u$ **then**
15:                $b'(z) \leftarrow b'(z) + 1/r$
16:            $t \leftarrow z$
17: **return** $(v, b'(v)) : \forall v \in V$
```

$\sigma_{uz}/\sum_{w\in P_u(v)} \sigma_{uv} = \sigma_{uz}/\sigma_{uv}$. We add z to p^* and repeat this procedure until $z \neq u$. In the end, we have single shortest path sampled from S_{uv}.

Analysis of this algorithm shows that it offers probabilistic guarantees on the quality of all approximations of the betweenness centrality. Let $b(v)$ be the actual betweenness centrality of node v and $b'(v)$ be the betweenness centrality returned by algorithm 1. Then following lemma holds:

Lemma 1 *With probability atleast* $1 - \delta$, *all the approximations computed by the algorithm are within* ϵ *from their real value:*

$$Pr(\exists v \in V s.t. |b(v) - b'(v)| > \epsilon) < \delta$$

The runtime of this algorithm is dominated by the computation of shortest path at each step which takes time $O(|V| + |E|)$ if graph is unweighted (BFS) and $O(|E| + |V|log|V|)$ othewise (Dijkstra's algorithm). This is multiplied by r for r iterations to obtain final complexity.

3 Methods

De Bruijn Graph

De Bruijn strings are a type of 'comprehensive' strings. For a given alphabet $\Sigma = \{A, C, T, G\}$ and a length k, a de Bruijn string of order k contains as substrings of length k over Σ. These strings can be represented in a sophisticated way in term of a graph called de Bruijn graph [6]. A de Bruijn graph of order k is a graph that contains all strings of length $k - 1$ from a given alphabets as nodes and contains edges between two nodes if the corresponding strings overlap by exactly $k-2$ letters in prefix-suffix manner. In other words, each k-mer is represented by two nodes in a graph connected by an edge.

We modify the de Bruijn graph generated from genomes to identify repeats in a better way. We define *unipaths* as the path of nodes in which the outdegree of first node on the path is 1 and indegree is greater than 1, indegree of last node on the path is 1 and outdegree is greater than 1 and for all other nodes, both indegree and out degree is exactly 1. We compress all the unipaths in a graph to represent in a compact way. For all the internal nodes on unipaths, we represent them by a single node with label as the combined label of all nodes. figure1 shows an example of such compression of unipaths in de Bruijn graphs.

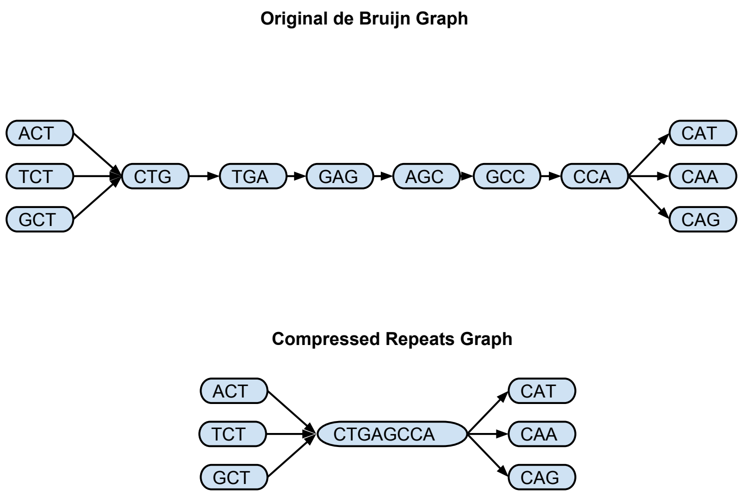

Fig. 1. Example of conversion of de Bruijn graph to Repeats graph

Networkit

We investigated the performance of approximate betweenness centrality algorithms on the metagenomic graphs to find inter-genome repeats. We used the implementation in Networkit[10] library. It provides implementation for various graph algorithms. We are particularly interested in the implementation of betweenness centrality algorithms. We make use of two approximate betweenness centrality algorithms provided by NetworKit.

NetworKit provides fast parallel implementation of approximate betweenness centrality algorithm by Matteo et al. [9]. The implementation first approximates diameter of of graph. After this, it calculates the number of shortest paths to be sampled. It processes shortest paths in the sample in parallel in multithreaded fashion and then aggregates the centrality scores for each shortest paths once every thread finishes processing of shortest path assigned to it.

We also investigate the performance of algorithm by Geisberger et al. [3]. NetworKit provides single threaded implementation of this algorithm. It first creates a sample of n nodes at random from a given graph. It then runs single source shortest path algorithm for each of those nodes. It calculates the betweenness scores for each node by the modified aggregation scheme proposed in the algorithm.

Here are few questions we want to investigate based on these algorithms:

- **Scalability**: How these algorithms scale on large metagenomic repeats graph? Does parallelized algorithm provides quick results or taking sample of fewer nodes provides quick results?
- **Accuracy**: Which of these algorithms provides better accuracy while determining inter-genomic repeats?
- **Sample Size**: For the algorithm by Geisberger et al. [3], how should a sample size be defined so that it would produce fast and accurate results? We try to come up with a heuristic to define sample size

Cutoff Criteria For Repeats

To determine which nodes in a graph should be marked as repeats, we define a criteria based on the value of centrality calculated by the algorithm. Let x be the mean of centralities of all the nodes and σ be the standard deviation. A node v is marked as repeat only if the following condition holds: $b(v) \geq x + c * \sigma$ where c is a constant and $b(v)$ is betweenness centrality of the node. We set value of c as 3 as it works out the best.

4 Results

We evaluated algorithms for betweenness centrality to find inter-genome repeats on real datasets described above. We use the notion of sensitivity and specificity to define the accuracy of repeat detection. We define following terms:

- **True positives:** Number of nodes marked as repeat that is found within at least 2 genomes
- **False positives:** Number of nodes marked as repeat that is only repetitive within a single genome
- **True negatives:** Number of nodes marked as non-repeat that is only repetitive within a single genome
- **False negatives:** Number of nodes marked as non-repeat that is found within at least 2 genomes

A good repeat detection should have both good sensitivity and specificity. Sensitivity reflects how many true repeats are detected. Detecting to few repeats can lead to assembly errors in scaffolding. Specificity reflects the false positives. Detecting too many repeats leads to a suboptimal assembly as these contigs do not fully participate in scaffolding.

5 Conclusion

In this work, we show how approximate betweenness centrality can be used to identify inter-genome repeats. We explored two algorithms, one based on sampling shortest paths in a graph and other based on sampling nodes in a graph. Application of both these algorithms to repeat detection problem provided significantly accurate and fast results. We explore how various parameters such as number of sampled nodes and filtering criteria to mark a node as repeat affects the accuracy of repeat detection. We are planning to use these techniques for repeat detection in our future metagenomic assembler.

References

1. D. A. Bader, S. Kintali, K. Madduri, and M. Mihail. Approximating betweenness centrality. In *Algorithms and Models for the Web-Graph*, pages 124–137. Springer, 2007.
2. U. Brandes. A faster algorithm for betweenness centrality*. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.
3. R. Geisberger, P. Sanders, and D. Schultes. Better approximation of betweenness centrality. In *ALENEX*, pages 90–100. SIAM, 2008.
4. D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)*, 24(1):1–13, 1977.
5. C. Kingsford, M. C. Schatz, and M. Pop. Assembly complexity of prokaryotic genomes using short reads. *BMC bioinformatics*, 11(1):21, 2010.
6. S. C. Kleene, N. de Bruijn, J. de Groot, and A. C. Zaanen. Introduction to metamathematics. 1952.
7. S. Koren, T. J. Treangen, and M. Pop. Bambus 2: scaffolding metagenomes. *Bioinformatics*, 27(21):2964–2971, 2011.
8. K. Madduri, D. Ediger, K. Jiang, D. A. Bader, and D. Chavarria-Miranda. A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE, 2009.
9. M. Riondato and E. M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 413–422. ACM, 2014.
10. C. L. Staudt, A. Sazonovs, and H. Meyerhenke. Networkit: An interactive tool suite for high-performance network analysis. *arXiv preprint arXiv:1403.3005*, 2014.
11. V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.