Noah Amsel
& Gabe Dolsten

12/15/18
LING 219

Top-Down Network Clustering Methods for Language Phylogeny

Introduction:

The central task of phylolinguistics is the construction of phylogenetic trees that describe the shared evolutionary history of a group of languages. Traditionally, this task has been performed qualitatively, relying on the judgement of expert linguists familiar with the similarities and differences of the languages in question. In the last several decades, increasing research attention has focused on quantitative methods of performing this analysis. In this effort, linguistics has benefited from a rich body of models and software originally developed for evolutionary biology. Just as biologists use computational methods to compare DNA sequences from different organisms, linguists have attempted to describe different languages with sets of numerical features. Commonly, these take the form of coded lexical data, where each feature corresponds with a word-form or cognate class. Taken together, the presence or absence of the various lexical items comprises the "genetic code" of the language from which a tree can be built.

Much recent work has used Bayesian methods to infer tree structure. These approaches require several layers of assumptions and parameter tuning. First, an underlying model of linguistic change must be established, including a substitution model and a clock model. Prior distributions must be provided for all parameters in the model, possibly including tree priors. A starting tree most be provided, a Monte Carlo simulation must be run to convergence, and finally the sampled trees must be aggregated into a meaningful summary tree. Recent work has demonstrated the power and applicability of this kind of analysis (Grollemund et al. 2015, Bouckaert et al. 2018). However, a need exists for simpler methods that do not require specifying parameters or highly specialized software. In this work, we do not consider Bayesian or Monte Carlo methods. Rather, we are interested in non-parametric, deterministic ways of making phylogenetic trees.

The baseline methods for producing trees are UPGMA (Unweighted Pair Group Method with Arithmetic mean), Neighbor Joining, and Maximum Parsimony. Maximum Parsimony searches for the tree with the fewest number of state transitions. This optimization is NP-Hard, making the method inapplicable to medium and large datasets. Hence, we will not consider it further. Neighbor Joining and UPGMA begin by computing pairwise distances between the languages under study. Initially, each language is placed in its own taxon. Pairs of taxa are sequentially merged together until only one taxon remains. Each intermediate taxon corresponds to an inner node in the tree. UPGMA and Neighbor Joining differ in how they choose which groups to merge. In UPGMA, the distance between two groups is defined as the average of the distances between all pairs of languages in each. The pair with the closest distance is merged. In Neighbor Joining, the distance to a newly-formed group is the average distance to its two subgroups minus the within-group distance. Two groups are merged if they have the smallest distance to one another, taking into account the average distances of each to all other groups.

Both UPGMA and Neighbor Joining are agglomerative or “bottom up” methods. An alternative approach to hierarchical clustering would begin with all languages in the same clade, and form a tree by successively splitting clades in two—this is called divisive or “top down” clustering. At each step, we seek to identify a “subfamily” or “cluster” of closely related nodes. Given a distance matrix, we can recast the problem in network theoretic terms. Form a network where each language corresponds to a node, and each pair of nodes is connected by an edge whose weight depends on the similarity between the two languages. In this setting, finding good divisions into subfamilies corresponds to the well-studied problem of community detection. Divisive methods have received less attention in phylogenetics. The popular split decomposition algorithm (Bandelt and Dress 1992) implemented in SplitsTree is a notable exception, however this method produces a "network" containing inconsistent splits, rather than a proper tree like UPGMA. Meanwhile, a growing body of research has studied the problem of community detection in graphs. We propose to adapt community detection methods to networks of languages in order to create linguistic trees.

The most popular way to quantify community structure in a network is to use the modularity score (Chen et al. 2014). In a good division of the network, the number (or total weight) of edges within each community should be higher than expected if the division were chosen randomly. The modularity score counts these "excess" edges. It is defined as

$$Q = \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2|E|} \right] \delta_{c_i,c_j}$$

Here A_{ij} is the weight of the edge between node i and j, k_i is the degree of node i (sum of all its edge weights) and $\delta_{c_i c_j}$is 1 if the nodes are in the same clusters and 0 otherwise. (The terms "community" and "cluster" are used interchangeably.) Given a graph, we seek an assignment of nodes to communities that maximizes this score (the most "modular" clustering). Unfortunately, solving this problem exactly is NP-hard (Chen 2014), but a variety of algorithms have been proposed to find an approximate solution. First is the fast-greedy algorithm (Clauset et al. 2004). Like UPGMA and Neighbor Net, it is agglomerative. For each pair of clusters, we calculate the increase in modularity from merging the two, and we merge the pair for which this increase is highest (hence, it is a greedy algorithm). The chief advantage of fast-greedy is speed; we use it primarily as a baseline. The leading eigenvectors method, described in in Newman 2006a and Newman 2006b, is a divisive algorithm. Newman rewrites the modularity score Q as the quadratic form induced by an appropriately constructed “modularity matrix”. To find a division of the network into two pieces with nearly maximal modularity, the signs of the coordinates of the first eigenvalue of this matrix are observed; vertex 10 is placed in the first community if the 10th coordinate is positive, and the second community if it is negative. The method is then applied recursively to the two communities that were identified. One attractive feature of this form of community detection is that it automatically knows when to stop dividing; when the signs are all positive or all negative, we should not split that community any further. The last method we consider is the Girvan-Newman or edge betweenness method, another divisive algorithm but one that does not rely on modularity (Girvan and Newman 2002). Rather, the intuition is that edges which cross between two communities tend to lie on the shortest path between many pairs of vertices, since there are only a few such edges linking nodes from different communities. This property of an edge can be quantified using the so-called betweenness score. At each iteration, the betweenness method removes the edge with the highest betweenness until, eventually, the graph is separated into two pieces. However, calculating the betweenness scores of all edges in the graph is extremely slow, making the method inapplicable to graphs of more than a few dozen nodes.

While distance-based metrics like UPGMA have been widely used, they have some important drawbacks. First, it is unclear which distance metric is best suited to the task. Secondly, reducing a coded dataset with potentially thousands of features to a pairwise distance matrix necessarily loses information. (The same problems apply to the modularity-based methods, which rely on a similarity score.) It would be preferable to group languages based on their underlying coded features, without relying on a distance or similarity metric. Again, we will recast the problem in network theoretic terms. Binary coded data can be represented as a bipartite graph in which one set of nodes corresponds to languages and the other set corresponds to features (cognate classes). If a language contains a certain feature, the two corresponding nodes are joined by an edge. Bipartite co-clustering is the task of simultaneously clustering the two types of nodes—languages and cognate classes—using only information about the other. Bipartite models have been used in other fields; for instance, Correa et al. (2018) use a bipartite network to perform word-sense disambiguation for Natural Language Processing. To our knowledge, this technique has not been applied to cross-lingual comparative data. Standard community detection algorithms can be applied to bipartite networks; however, several specialized algorithms have also been developed to take full advantage of the bipartite structure. We consider the spectral co-clustering method of Dhillon (2001), which uses eigenvectors of a normalized adjacency matrix to represent both kinds of nodes (languages and cognates) as vectors in the same continuous space. It then uses k-means to cluster them.

Methods:

We aim to adapt the methods from network theory described above to the task of hierarchical language clustering. We rely on the iGraph Python package (Csardi and Nepusz 2006) for working with graphs, which includes implementations of common graph clustering algorithms. Given binary coded cognate data, we create a distance matrix using the Hamming distance (normalized to account for missing data). We then construct a complete graph where edge weights correspond to similarity—that is, 1/distance. In iGraph, some clustering algorithms, like fast-greedy and edge-betweenness, produce fully resolved bifurcating trees as output. Leading eigenvectors recursively divides clusters—so it captures partial tree structure—but the function only outputs the final clustering. By digging a bit into the iGraph internals, we were able to recover the splits and produce a partial bifurcating tree. However, as stated above, the method only divides clusters up to a certain point. When no modularity increase can be found by subdividing a cluster, it leaves that cluster alone. Furthermore, modularity-based methods are known to suffer from a so-called resolution limit (Chen 2014). In some cases, small groups are left in the same community when they really should be separate communities. Thus, to create a more complete tree for medium and large language families, we apply the method recursively on the identified clusters. By excising the detected clusters from the rest of the network, further meaningful divisions within each clade may be detected. For very small groups, even this extension may fail to produce a fully resolved tree (where each language winds up in its own clade). We experiment with a final step in which a backup method—either fast-greedy or edge-betweenness—is used. Edge-betweenness is too slow to apply to the network as a whole, but for fewer than about 30 nodes it runs quickly and is a well-suited backup. In total, five methods were tested:

1. Fast-greedy
2. Newman Leading Eigenvectors (applied iteratively)
3. Newman, non-binary (The bifurcating dendrogram is not recovered. The algorithm decides how many communities to make and a multiforcation is formed. The method is then applied iteratively producing multiforcations at each step)
4. Newman with fast-greedy backup

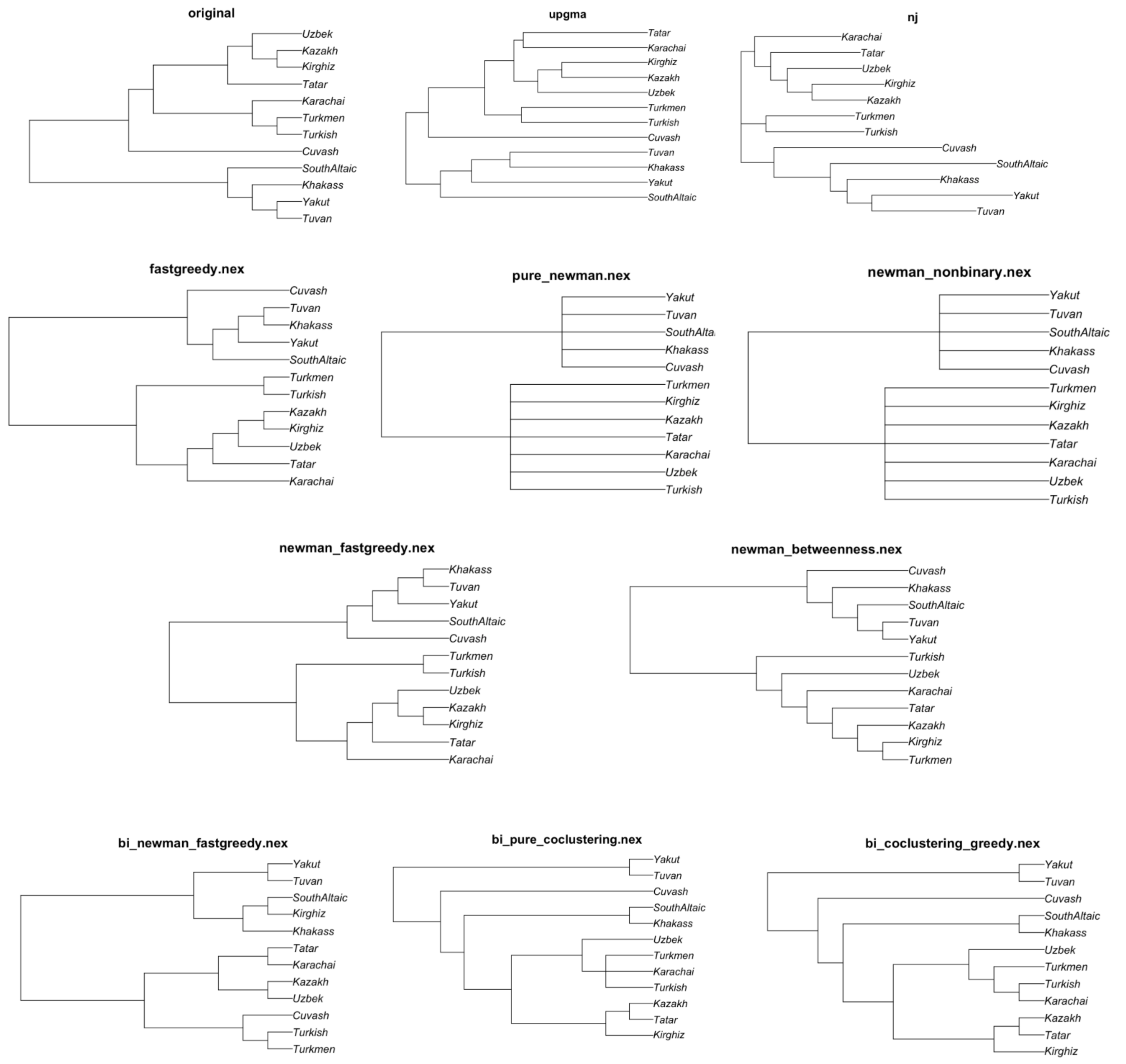
5. Newman with betweenness backup

For the bipartite model, we apply the one of the same methods—Newman with fast-greedy backup—used to analyze distance matrices to a bipartite graph of languages and cognate classes. We also use Dhillon's spectral coclustering method as implemented in the scikit-learn package. This method is not hierarchical, so to create a tree, we set the number of clusters to 2 and apply the method recursively on the results of the split. We make three trees from the bipartite graph:

1. Newman with fast-greedy backup
2. Spectral Co-clustering
3. Spectral Co-clustering with fast-greedy backup

Experiments:

We test our methods on three corpora of coded cognate data. The first is a small corpus of twelve Turkic languages, with 1,819 characteristics. We compare with a reference language network generated using SplitsTree. The second is a dataset of 424 Bantu languages with 3,859 characteristics (Gröllemund et al. 2015). We compare with a reference tree from the original publication created using Bayesian methods. Last is a dataset of 192 Pama-Nyungan languages with 5,593 characteristics (Bouckaert, Bowern, and Atkinson, 2018); again, we compare with the authors' tree. The data came pre-coded in a suitable binary format. All code and data is available on GitHub (https://github.com/NoahAmsel/Phylogenetic_Clustering).



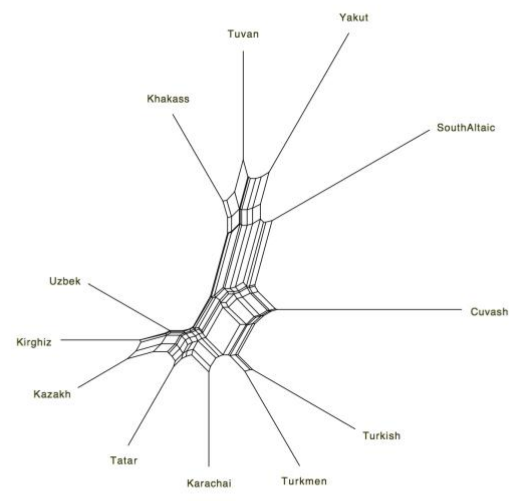


Figure 1: Trees of the Turkic language family. From top to bottom, left to right: Reference, UPGMA, Neighbor Joining, Fast-greedy, Newman's method with binary splits and no back up, Newman's method with multifurcations and no back up (these turn out to be the same, as the method finds no further binary splits), Newman's method with fast-greedy back up, Newman's method with edge betweenness backup, Newman's method with fast-greedy backup on the binary tree, Dhillon's spectral co-clustering method on the binary tree, Dhillon's spectral co-clustering method with a fast-greedy backup, and a SplitsTree network of the same data.

Results:

Figure 1 shows the trees created for the Turkic family. In addition to the reference network, we compare with Neighbor Joining and UPGMA trees.

Examining the binary and non-binary Newman trees, we can see that both methods recovered a good first split in the family but failed to detect subgroups beyond that. The fast-greedy backup method was able to recover clades pretty much identical to those in the neighbor joining tree, as was the fast-greedy method when used alone. The betweenness backup method produced a decent tree, but always separated out one node from all the others, which is not realistic. Except for Cuvash and Karchai, the Newman method on the bipartite graph also produced decent results. The two spectral clustering trees (one with backup and one without, as evidenced by the multifurcation) look much like each other but not much like the reference tree. Still, as a proof of concept that can be evaluated by eye, the Turkish family shows that these methods can definitely pick up phylogenetic signal.

Figure 2 below shows our results for the Bantu family. While there are too many taxa to analyze by eye, we can make some qualitative observations. Compared to the reference tree, the binary Newman method and especially the Bipartite (Co-clustering) method tend to make splits which divide the taxa into approximately equal-sized groups. For the Spectral Co-clustering method, this could have to do with the k-means algorithm, which may be unlikely to group small amounts of data as a totally separate family. Likewise, modularity based methods like Newman's may not see such a large increase in modularity from dividing out only a small number of nodes, and might "rush into" a more impactful, balanced division that really ought to have come later.

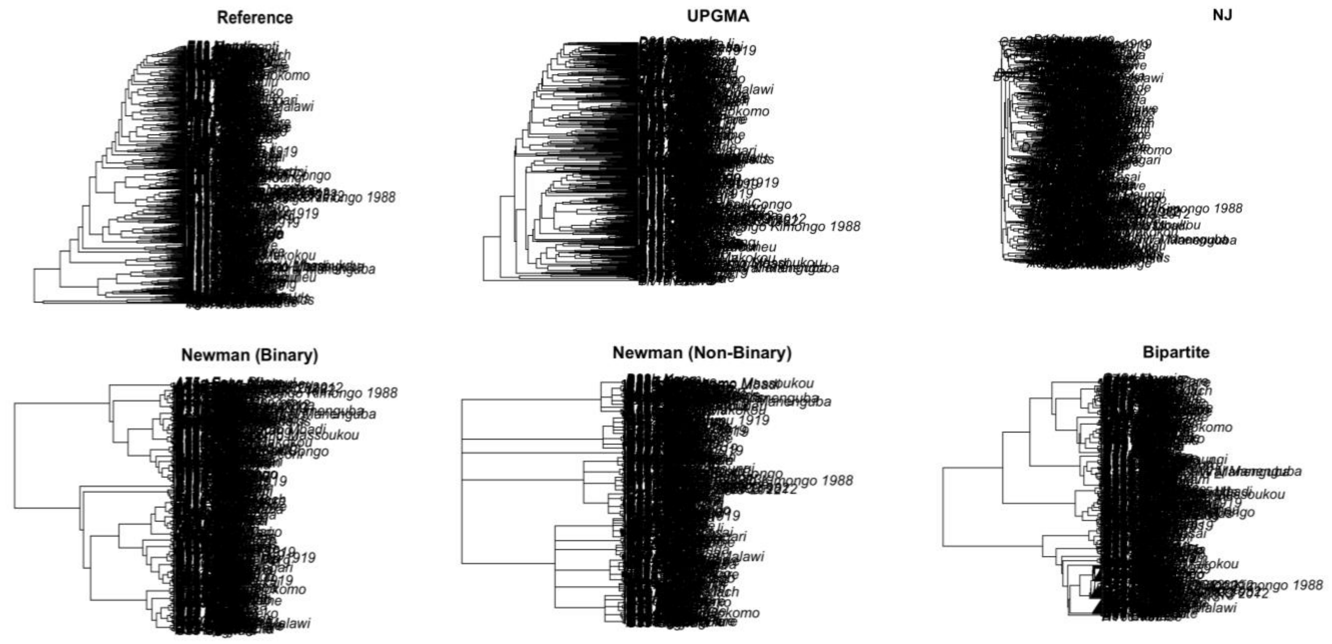


Figure 2. Bantu trees. Top row shows baselines, bottom row shows three of our methods

To quantify our comparisons, we computed topological distances between trees and reference trees using the R function dist.topo. This metric is based on counting the number of branches for which the division of nodes when the branch is cut differs between the two graphs. The results are shown in Table 1. A small topological distance to the reference tree indicates good performance. This distance is scale-dependent—that is, it increases when more taxa are considered. To put the different families on a common scale, we normalize each by dividing by the topological distance of a random tree generated for that family. The best score among our trees and the best score overall are shown in bold.

Table 1: Distance to reference tree for trees produced by Various Algorithms. Distances are normalized to the random tree.

Algorithm		Normalized Distance to Reference Tree		
		Pama-Nyungan	Bantu	Turkic
Top-down	Binary Newman with FastGreedy	0.87	0.79	0.67

	Binary Newman with Betweenness	0.60	0.62	**0.22**
	Non-binary Newman with FastGreedy	**0.55**	**0.55**	0.56
	Pure Newman	0.59	0.59	0.56
Bipartite	Coclustering with Greedy	0.96	0.91	0.78
	Bi Newman FastGreedy	0.98	0.88	0.89
	Bi Pure Coclustering	0.80	0.67	0.72
Bottom-Up	UPGMA	0.58	0.45	0.33
	NJ	**0.49**	**0.43**	**0.17**
Random	Random (pre-normalized shown in parens)	1.00 (381)	1.00 (844)	1.00 (18)

The algorithms perform quite consistently between families. In general, the Neighbor Joining tree performs best, and UPGMA is second. Of our algorithms, the Newman methods which include multifurcation do best, followed by the binary Newman methods, and the bipartite methods do worst. However, for the Pama-Nyugan family, the non-binary Newman method was able to outperform the baseline UPGMA method, showing that our best trees were reasonable compared to the baselines. For the Turkic dataset, one of our trees also performed better than UPGMA, but the small size of the dataset and the poor performance of that method generally indicate that this was likely a fluke. Since the topological distances were so much smaller for the Turkic dataset (in the teens rather than the hundreds), we should be cautious about comparing the scores with those of the larger families. It was puzzling that the non-binary Newman method—which included many multifurcations—was actually worse than the binary Newman which didn't. Under the hood, even the non-binary Newman splits were created by iterative binary splits, so one would think that recovering those splits would improve the quality of the tree. Further investigation must be made to determine why this is the case and to ensure that multifurcations are being properly penalized in the metric. While our methods do not outperform the baselines in general, we can see that they do much better than a random tree. While these results fall short of establishing the competitiveness of network based methods, they indicate that these methods are powerful and sensitive enough to warrant future study.

Even if we don't get a reliable fully-resolved tree using these methods, perhaps they can be useful in telling us which broad subfamilies are very well supported by the data. Agglomerative methods, after all, have no cutoff to stop before the tree is fully resolved, but divisive methods can tell us which clades are too ambiguous keep splitting. The fact that even the non-binary Newman method performed strongly could indicate that the communities it *did* find we fairly robust.

Several important questions are left to further work. One question we don't consider here is which similarity metric to use. We use 1/hamming distance = number features that aren't missing / number of mismatched features. When there is a lot of missing data, this metric can be skewed. Additionally, this metric seems to produce many edges of similar weight only a few very large weights. Perhaps the clustering methods would perform better if the distribution of weights were more evenly spread out. Given the good theoretical reasons to prefer a bipartite model, which avoids the need to calculate a distance matrix, it would be worth investigating alternative clustering algorithms for these graphs. Evidently, the general-purpose algorithms show middling performance on these graphs, and the co-clustering method (which was designed to find many clusters at once) is ill-suited to hierarchical clustering. One issue with the co-clustering method was that it tended to produce balanced splits of the languages, while real linguistic splits are often unbalanced. (Albanian and Italic are both considered top-level divisions of Indo-European, but the former contains only one living language while the later contains dozens.) The method we used applied to k-means to cluster the vector representations obtained for each language, but in principle a different clustering method that is less biased toward balanced splits. One could even use a preexisting hierarchical clustering method on these vectors to obtain a complete tree in one step (whereas we had to apply the method iteratively). Or perhaps a different hierarchical clustering algorithm could be developed specifically for bipartite graphs. A further advantage of the bipartite model is that in addition to producing a tree of the languages, it also identifies the cognates/classes associated with each clade. This feature would help validating the tree against historical records ("Was this set of words really present in the ancestor language?") and with identifying borrowing ("This word is in the language's vocabulary, but is clustered with a totally different clade. It must have been borrowed.").

Conclusion:

Community detection algorithms from network theory offer an alternative non-parametric approach to creating language trees. Top down algorithms for this task have generally received inadequate research attention. We demonstrated that, while it performs worse than agglomerative baselines, Newman's leading eigenvector method is able to capture phylogenetic signal. Bipartite models for language clustering have strong motivational basis, but further research is required to make this approach competitive.

Bibliography:

Bandelt, H.-J., & Dress, A. W. M. (1992). Split decomposition: A new and useful approach to phylogenetic analysis of distance data. Molecular Phylogenetics and Evolution, 1(3), 242–252. https://doi.org/10.1016/1055-7903(92)90021-8

Bouckaert, Remco R., Bowern, Claire, & Atkinson, Quentin D. (2018) The Original and Expansion of Pama-Nyungan Languages Across Australia. Nature Ecology & Evolution 2, 741-749

Chen, Mingming, Kuzmin, Konstantin & Szymanski, Boleslaw K. (2014) Community Detection via Maximization of Modularity and Its Variants. IEEE Transactions on Computational Social Systems 1(1). Retrieved from https://ieeexplore.ieee.org/document/6785984.

Clauset, A., Newman, M. E. J., & Moore, C. "Finding community structure in very large networks", Phys. Rev. E, vol. 70, pp. 066111, Dec. 2004.

Corra, E. A., Lopes, A. A., & Amancio, D. R. (2018). Word Sense Disambiguation Via Bipartite Representation of Complex Networks. Information Sciences—Informatics and Computer Science, Intelligent Systems, Applications: An International Journal,442(C). Retrieved from https://dl.acm.org/citation.cfm?id=3199425.

Csardi, Gabor & Nepusz, Tamas (2006) The iGraph Software Package for Complex Network Research. InterJournal, Complex Systems 1695. http://igraph.org.

Dhillon, Inderjit S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '01). ACM, New York, NY, USA, 269-274. DOI=http://dx.doi.org/10.1145/502512.502550

Girvan, M., & Newman, M. E. J. (2002). Community structure in social and biological networks. Proceedings of the National Academy of Sciences, 99(12), 7821. https://doi.org/10.1073/pnas.122653799

Grollemund, R., Branford, S., Bostoen, K., Meade, A., Venditti, C., & Pagel, M. (2015). Bantu expansion shows that habitat alters the route and pace of human dispersals. Proceedings of the National Academy of Sciences, 112(43), 13296. https://doi.org/10.1073/pnas.1503793112

Newman, M. E. J. (2006a) Modularity and Community Structure in Networks. PNAS 103(23). Retrieved from http://www.pnas.org/content/103/23/8577.

Newman, M. E. J. (2006b). Finding community structure in networks using the eigenvectors of matrices. Physical review. E, Statistical, nonlinear, and soft matter physics, 74 3 Pt 2, 036104. Retrieved from https://arxiv.org/abs/physics/0605087

Yang, Z., Algesheimer, R., & Tessone, C. J. (2016). A Comparative Analysis of Community Detection Algorithms on Artificial Networks. Scientific Reports, 6. Retrieved from https://www.nature.com/articles/srep30750.