Fast approximation algorithms for finding node-independent paths in networks

Douglas R. White

Department of Anthropology, University of California, Irvine, CA 92697

M. E. J. Newman

Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501

(Dated: June 29, 2001)

A network is robust to the extent that it is not vulnerable to disconnection by removal of nodes. The minimum number of nodes that need be removed to disconnect a pair of other nodes is called the connectivity of the pair. It can be proved that the connectivity is also equal to the number of node-independent paths between nodes, and hence we can quantify network robustness by calculating numbers of node-independent paths. Unfortunately, computing such numbers is known to be an NP-hard problem, taking exponentially long to run to completion. In this paper, we present an approximation algorithm which gives good lower bounds on numbers of node-independent paths between any pair of nodes on a directed or undirected graph in worst-case time which is linear in the graph size. A variant of the same algorithm can also calculate all the k-components of a graph in the same approximation. Our algorithm is found empirically to work with better than 99% accuracy on random graphs and for several real-world networks is 100% accurate. As a demonstration of the algorithm, we apply it to two large graphs for which the traditional NP-hard algorithm is entirely intractable—a network of collaborations between scientists and a network of business ties between biotechnology firms.

I. INTRODUCTION

The logical connections between the structural properties of a graph, such as its robustness to the removal of vertices, and properties of graph traversal, such as numbers of paths between vertices, have been widely studied and many rigorous results are known from graph theory (Harary 1969, Chartrand and Lesniak 1996). An example of particular interest in the study of social networks is that of node-independent paths on graphs—sets of paths between specified pairs of vertices that share no vertices in common other than their starting and ending points. Greater numbers of such paths between nodes provide graphs with greater cohesion. The number of node-independent paths between an initial vertex i and a final vertex f is also the minimum number of vertices which must be removed from the graph in order to disconnect i and f from one another, and is therefore a direct measure of the resilience of the graph to vertex deletion. Unfortunately, the calculation of numbers of node-independent paths between pairs of vertices is computationally difficult, taking time exponential in the size of the graph in the most general case. For all but the smallest or sparsest graphs, this makes exact counting of node-independent paths intractable. In this paper, we present some fast and accurate approximation algorithms which give good lower bounds on the numbers of node-independent paths in large graphs in reasonable time. Some definitions from the theory of graphs will help to define the concepts required for this approach (White and Harary 2001).

Consider a graph $\mathcal{G}$ that has n vertices and size m edges. Node-independent paths from an initial vertex i to a final vertex f in $\mathcal{G}$ are paths from i to f with no vertices in common other than i and f. For each pair of distinct vertices (i, f) in $\mathcal{G}$, let $K(i, f)$ be the maximum number of such node-independent paths. The local connectivity $\kappa(i, f)$ of two distinct and nonadjacent vertices i and f in $\mathcal{G}$ is the minimum number of vertices that must be removed (minimum separating cutset) to disconnect them. In particular, if $\kappa(i, f) = 0$ then i and f are disconnected in $\mathcal{G}$. For any two distinct and nonadjacent vertices (i, f) it is intuitively reasonable, but nontrivial to prove, that $K(i, f) = \kappa(i, f)$ (Menger 1927). Technically, the local connectivity of i and f is not defined if they are adjacent, but for simplicity we will define local connectivity in this case to be equal to the maximum number $K(i, f)$ of node-independent paths between i and f. This allows us to state that for any distinct (i, f) in $\mathcal{G}$: $K(i, f) = \kappa(i, f)$, and if $\mathcal{G}$ is a complete graph, then $K(i, f) = \kappa(i, f) = n - 1$.

If the connectivity $\kappa(i, f)$ is greater than or equal to some number k, then i and f are said to be k-connected. A maximal subset $\mathcal{S}$ of $\mathcal{G}$ such that all vertex pairs (i, f) are k-connected via paths within $\mathcal{S}$ is called a k-component. White and Harary (2001) call a k-component a cohesive block. A maximal subset $\mathcal{S}$ of vertices each pair of which is k-connected via paths which

are allowed to include vertices not in $\mathcal{S}$ we will call an extracohesive block.

Globally, a graph $\mathcal{G}$ is said to have connectivity $\kappa(\mathcal{G})$ if no removal of fewer than κ vertices will increase the number of components of $\mathcal{G}$. Again it is reasonable but nontrivial that $\mathcal{G}$ has connectivity $\kappa(\mathcal{G})$ if and only if between any two distinct vertices (i, f) of $\mathcal{G}$, there are at least κ node-independent paths (Menger 1927). Thus, if $K(\mathcal{G})$ is the pairwise minimum of $K(i, f)$ over all distinct (i, f) in $\mathcal{G}$, then for any graph $\mathcal{G}$, $\kappa(\mathcal{G}) = K(\mathcal{G})$.

In Section II we present an exact algorithm for calculating $K(i, f)$, the number of node-independent paths between a specified pair of vertices, that requires exhaustive back-tracking and computation time that scales exponentially with graph size. In Section III we give a family of approximation algorithms for computing a lower bound on $K(i, f)$ that can be used either for undirected or directed graphs. The computation time taken by the latter algorithms scales linearly with graph size and hence the algorithms are not limited to use with small graphs. For small graphs, the exact algorithm serves as a baseline for evaluating the accuracy of the approximation algorithms, and we perform such an evaluation in Section IV, using both artificial (computer generated) graphs, and graphs from real-world applications. In Section V we then apply the approximation algorithms to large graphs, for which the exact algorithm is computationally intractable. In Section VI we discuss briefly the application of our algorithm to the calculation of the k-components of a graph, and in Section VIII we give our conclusions.

II. AN EXACT ALGORITHM FOR NODE-INDEPENDENT PATHS

The standard algorithm for finding all node-independent paths from an initial vertex i to a final vertex f in a graph is the exhaustive back-tracking algorithm as follows.

1. With each vertex in the graph we associate a label which may take the value "available," meaning it is available to be incorporated into a path, "unavailable," meaning it is never to be used as part of a path, or an integer numerical value k, indicating that it is a part of the kth node-independent path found between i and f. Initially all vertices are labeled "available" except the initial vertex i, which is labeled "unavailable." We also keep a record of the current number n of node-independent paths on the graph, whose value is initially zero.

2. Within the set of vertices which are currently marked "available," we search for a new path from i to f (i.e., one which has not previously been considered as the nth path). If one exists, we increase the value of n by one and then mark all the vertices along the chosen path with the numerical value

$k = n$, to indicate that they have been used as part of the nth path. Then we repeat from step 2.

3. If no path of available vertices exists, we take all the vertices which belong to the nth path and mark them "available" again. Then we decrease n by one, and repeat from step 2.

4. When there are no more possible candidates for the first path from i to f, the algorithm ends. The number of node-independent paths on the graph is the highest value obtained by n during the run.

This algorithm can be used for either direct or undirected graphs. It will always correctly give the number of node-independent paths between i and f, but it is in general very slow. In the typical case the total number of paths between two vertices scales exponentially with graph size, and running time is thus also at least exponential in graph size. Improvements on the algorithm are possible. For example, the algorithm as we have described it separately considers configurations in which the same set of paths between i and f are present on the graph, but are just labeled in a different order. Removing this redundancy can speed the calculation by a substantial factor. However, overall scaling is still at least exponential in graph size.

In practice this limitation means that the algorithm can be used for graphs up to a few tens of vertices—more in the case of graphs with low average degree than ones with high average degree. For large graphs of hundreds or thousands of nodes, which have become increasingly common in recent years, the exhaustive algorithm is entirely impractical. Instead, therefore, we turn to approximation methods to estimate numbers of node-independent paths.

III. APPROXIMATION ALGORITHMS

For many problems whose exact numerical solution is slow, faster algorithms exist which will give approximate answers. In the best case, these algorithms give an absolute bound on the true answer to the problem, and we here present a one-parameter family of approximation algorithms for counting node-independent paths which all do exactly this. The one parameter in this family of algorithms allows us to tune the time taken by the algorithm, with the payoff being that quicker members of the family provide poorer bounds on the number of node-independent paths. Our algorithm runs in time linear in the number of vertices in the graph, with only the leading constant varying with the value of the parameter.

The fundamental idea behind our method is as follows. As before we find node-independent paths by selecting one path from the initial vertex of interest i to the final vertex f, marking the vertices along that path as taken, so that they cannot be used by any other path, and then looking for another path among the remaining vertices.

This process is repeated until no more paths from i to f exist. Instead of running through all possible paths from i to f however, as before, we note that longer paths are less likely to belong to large sets of node-independent paths than shorter ones. If we choose a very long and circuitous path from i to f as our first path, for example, then that choice removes from the graph a large number of vertices, which makes it hard to find many other node-independent paths among the remaining vertices. Thus our chances of finding many node-independent paths will on average be improved if we favor short paths from i to f. Our algorithm takes this idea to its most extreme conclusion, and considers only *shortest* paths from i to f. In its simplest form, our algorithm is as follows.

1. Initially label all vertices "available." Set $n = 0$.

2. Search for the shortest path of available vertices from i to f. If such a path exists, increase n by one, label all vertices along this path as belonging to the nth path and hence no longer available, and repeat from step 2.

3. If no path exists from i to j, the algorithm ends.

The final value of n is a strict lower bound on the number of node-independent paths from i to f. If the final value of n is equal to either d_i or d_f, the degrees of the initial and final vertices, then this bound is in fact the exact number of node-independent paths, since the smaller of the two degrees provides an upper bound on the number of paths.

Note that, even though we consider only shortest paths from i to f, this does not mean that all paths found are the same length, since the shortest path composed of "available" vertices is not necessarily the same length as the shortest path on the complete graph.

The shortest path between two vertices on any graph can be found in time $O(m)$ by breadth-first search, where m is the number of edges in the graph, and since $\min(d_i, d_f)$ is an upper bound on the number of times we have to search for the shortest path, worst-case total running time is $O(m \min(d_i, d_f))$.

In Fig. 1 we illustrate the application of this algorithm to pairs of vertices on three different undirected graphs. In the first graph, Fig. 1a, the algorithm works perfectly: it not only gives the correct result, it also tells us that the result is correct, since the number of paths found is equal to the degree of, in this case, both i and f. In the second graph, Fig. 1b, the algorithm again gives the correct result, but cannot tell that the result is correct, since the number of paths found is less than the degree of either i or f. In the third graph, Fig. 1c, the algorithm does not give the correct result, but still gives a correct lower bound on the number of node-independent paths.

The algorithm as we have described it is not yet complete. Some graphs have more than one shortest path of the same length between a specified pair of points. What do we do in this case? The most thorough approach would be to go through each shortest path in
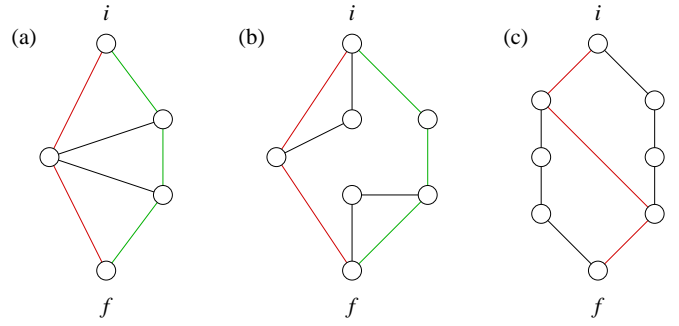

FIG. 1: Examples of the application of our algorithm to find node-independent paths between vertices i and f in three simple graphs. In each case the first path found by the algorithm is colored red and the second one (if it exists) green.

turn, marking their vertices as taken, and searching for all shortest paths between i and f in the remaining subset of vertices, and repeating until no more paths exist from i to f. This algorithm however is considerably slower than the one described above. In the typical case, the number of shortest paths between two points i and f is bounded by an increasing polynomial n^α in the number of vertices and, allowing for the $O(m)$ time taken by the breadth-first search procedure, the total running time of the algorithm is $O(n^\alpha m)$, which is slower than linear in system size, even for sparse graphs.

We can however recover linear performance by a slight modification of the algorithm. For given initial and final vertices i and f we calculate all shortest paths between them, then choose p of those paths to pursue further, or fewer if there were not p found. The value of p can be tuned to vary the running time of the algorithm. Each of the p paths chosen is eliminated from the graph in turn, and the shortest path calculation is repeated on the remaining subgraph, just as before. It is easy to see that the running time of this algorithm is bounded above by $O(p^{\min(d_i, d_f)} m)$, and in the extreme case where p is set to 1, is just $O(m)$, which is fast enough for even the largest graphs found in real-world applications.

And how do we choose the particular set of p shortest paths that we use in this algorithm? Many strategies are possible, but perhaps the simplest is to choose them at random making use of a suitable random number generator, and this is what we do in the present work. Repeated runs of the algorithm on the same data may give different results, although all runs will give correct lower bounds on the numbers of node-independent paths. In this case, clearly the highest of these lower bounds is our best bound on the actual number of paths. This behavior is typical of many stochastic approximation algorithms.

p	total incorrect	percentage error
1	732	3.85
2	166	0.87
3	102	0.54
4	93	0.49
5	86	0.45
6	88	0.46

TABLE I: Comparison of the results of the approximation algorithm, for various values of the parameter p, against exhaustive enumeration results for 100 random graphs $\mathcal{G}_{n,m}$ with $n = 20$ vertices and $m = 40$ edges. The second column indicates for how many of the 19 000 distinct pairs of vertices in the 100 graphs the number of node-independent paths was incorrectly calculated. The third column is the corresponding percentage of the time the approximation algorithm is in error.

IV. TESTS OF THE ALGORITHM

In this section we test our algorithm on a number of graphs which are small enough that we can also perform the exact exhaustive enumeration of node-independent paths described in Section II. This allows us to compare the results from the approximation algorithm with the known correct results for the same graph and hence determine how well the algorithm performs in a variety of cases.

We first test the algorithm on a set of computer-generated random graphs. These have the advantage that we can generate many of them with statistically similar properties, allowing us to gauge quantitatively how often the results given by our approximation algorithm differ from the exact answer. We then further test the algorithm on two real-world graphs, for both of which it turns out to work perfectly.

A   Random graphs

As our first test of the algorithm we have generated one hundred random graphs, of the type normally denoted $\mathcal{G}_{n,m}$, i.e., graphs with a fixed number of vertices n and a fixed number of edges m, with edges placed between pairs of vertices chosen uniformly and independently at random (Bollobas 1985). In this test we used $n = 20$ vertices and $m = 40$ edges, so that the average vertex degree is four. These graphs are small enough and sparse enough that exhaustive enumeration of node-independent paths between each of the $\frac{1}{2}n(n-1) = 190$ pairs of vertices can be performed in a just a few seconds. In Table I we show a comparison of this exhaustive enumeration with runs of the approximation algorithm.

The table shows that, out of the 19 000 distinct pairs of vertices in the one hundred graphs, 732 of them were accorded the wrong number of node-independent paths

by our approximation algorithm, when the parameter p was set to its minimum value of 1. This corresponds to 3.85% of all pairs. The other 18 268, the vast majority, were calculated correctly by the algorithm. The number of errors falls sharply as the value of p is increased, eventually leveling off at about 90 when p reaches 4, which corresponds to a percentage error of about 0.5%. For values of p higher than this, no improvement in seen in the number of pairs calculated correctly, which presumably indicates that the remaining incorrect pairs are of the type seen in the third panel of Fig. 1, for which no value of p, however large, will ever give the correct answer.

Of course, random graphs are not generic; they are a very special subset of all graphs. However, these results indicate that, under appropriate conditions, our algorithm can achieve an accuracy of better than 99%.

B   Zachary's karate club

For our second test of the algorithm, we use real world data, drawn from Zachary's well-known "karate club" study (Zachary 1975, 1977). During two years of ethnographic observation of 34 members of a karate club, a karate teacher (T, #1, Mr. Hi) and a club administrator (A, #34, John) in dispute about whether to improve the club's solvency by raising fees (teacher) or by holding down costs (administrator). This resulted in each calling meetings at which they hoped to pass self-serving resolutions by encouraging attendance by their own supporters. The formation of factions was visible to the ethnographer and evident in meeting attendance, which varied in factional proportions according to the convener. Ultimately the teacher was fired, set up a separate club, and the factional split became the basis for each person's choice of which of the new clubs they would join.

Zachary collected data on friendships between pairs of individuals within the club. He constructed networks in which friendships were weighted according to the number of contexts (karate and other classes, tournaments, bars and hangouts) in which the individuals in question met. Here we consider only the unweighted version of the friendship network, which has 34 nodes including the teacher and administrator.

Applying our approximation algorithm to the karate club data and comparing the results with exhaustive enumeration of node-independent paths, we find again that with $p = 1$, the algorithm makes a moderate number of errors—typically about 10 pairs of vertices out of 561 are accorded the wrong number of node-independent paths, about a 2% error. However, this falls off sharply as p is increased, and for $p > 4$ we find that the algorithm performs the calculation perfectly on almost all runs. All 561 path-counts are exactly the same as the exhaustive enumeration.

In the left panel of Fig. 2 we show a hierarchical clustering tree for the karate club data, where the distance
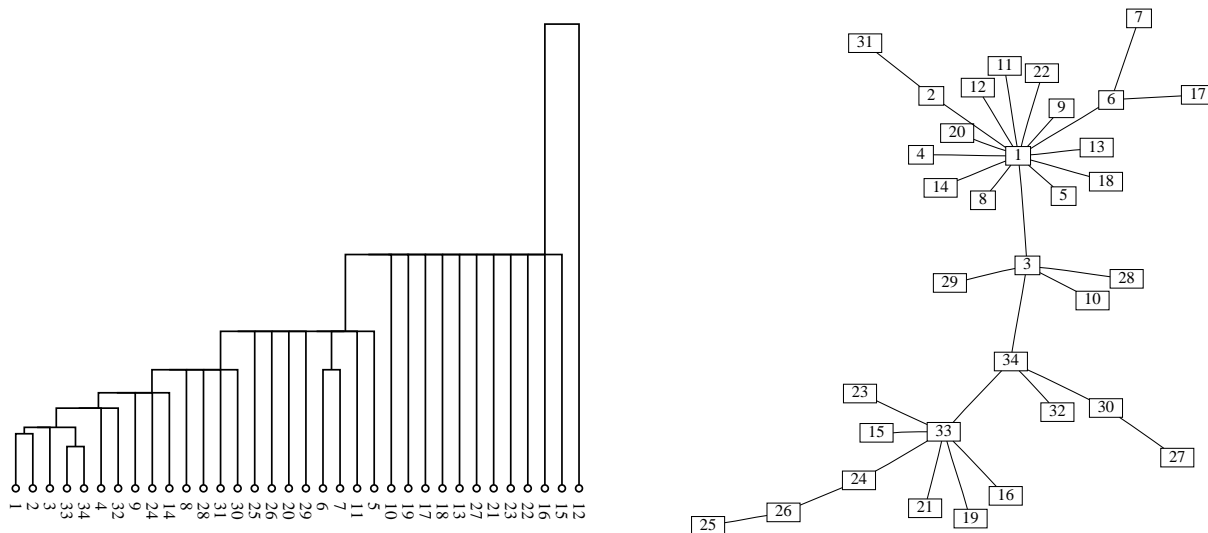
FIG. 2: Left: hierarchical clustering of the karate club dataset, based on a distance between pairs of vertices equal to the inverse of the number of node-independent paths between them. Right: minimum spanning tree for the same.

between vertices in the dataset is the reciprocal of the number of node-independent paths between them. As the figure shows, vertices 1, 2, 3, 33, and 34 are at the core of the club, with other club members belonging to the community through their connections with one of these. In the right panel of Fig. 2, we show the minimum spanning tree for the same calculation, which reveals indeed that the network splits roughly into two parts, one centered around vertices 1 and 3, and one around vertices 33 and 34. The minimum spanning tree of a component of n vertices within a graph is the set of $n-1$ edges which connects all vertices in the component while having the maximal weight, where the weight in this case is the number of node-independent paths. In cases where pairs of vertices tied for number of node-independent paths, we broke the tie in favor of the pair separated by the shortest geodesic distance.

This calculation is a good example of the speed of the algorithm also. The exact enumeration of all node-independent paths for this graph took six hours on a current workstation (*circa* 2001). With $p = 5$, our approximation algorithm took 28 seconds on the same computer to get identical answers.

C Taro exchange network

Our third test of the algorithm uses data from a network with very sparse and uniform links and much local structure. Among the Orokaiva of Papua New Guinea, Schwimmer (1973) collected data from the village of Sivepe on taro exchange between 20 households. At feasts, raw taro is given by men to start a social relationship or to transform an existing one into one where the giver attains "Big Man" status. Small but frequent gifts

of cooked taro by women express a desire to maintain intimate relations between households. We use Hage and Harary's construction of the taro graph, in which edges correspond to first or second choices of exchange partner or reciprocated third choices (Hage and Harary 1991). Applying our algorithm with $p = 5$ to this network, we again find that the numbers of node-independent paths calculated are in exact agreement with the exhaustive calculation for all pairs of vertices. Once again, the algorithm provides not just a lower bound, but a perfect enumeration of node-independent paths, in a fraction of the time taken by the exhaustive algorithm.

V. APPLICATIONS

In this section we give two examples of applications of our approximation algorithm to networks for which exact enumeration of node-independent paths is impossible, because the networks' size and density makes the exhaustive backtracking algorithm of Section II computationally intractable.

A Collaboration network

We have applied our algorithm to the collaboration network of 271 scientists at the Santa Fe Institute—an interesting case study since the institute focuses on interdisciplinary research. Actors in the network are scientists in residence at the Santa Fe Institute during any part of calendar year 1999 or 2000, and their collaborators. Two actors are considered to have a tie between them (a collaboration) if they coauthored one or more scientific papers together during the same period. The
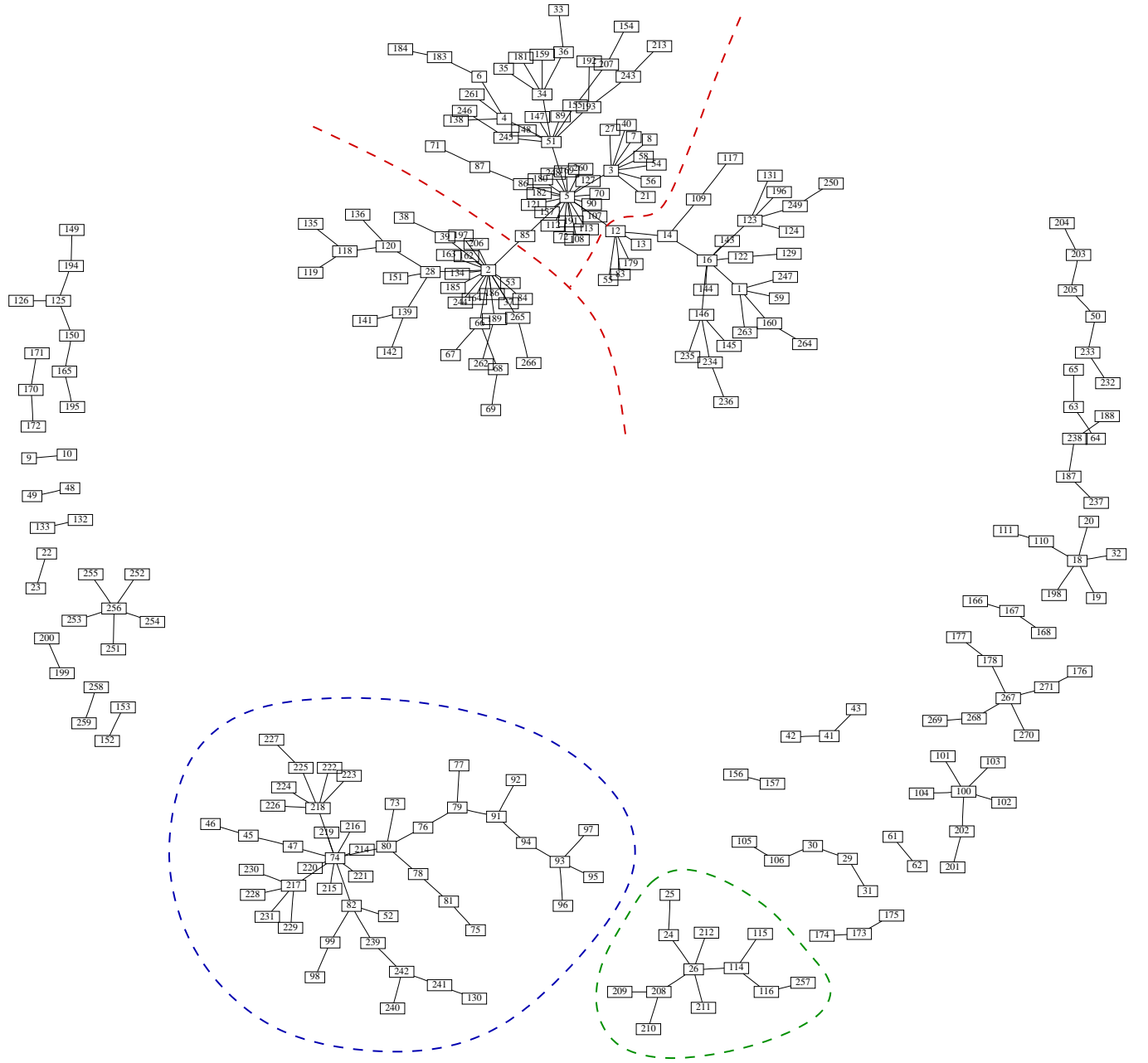
FIG. 3: Minimum spanning tree of the Santa Fe Institute collaboration newtork discussed in the text. Dotted lines denote known communities within the institute.

data were compiled by S. Knutson from publicly available bibliometric sources, and from the institute's technical reports.

With $p = 5$, our appoximation algorithm took about 70 seconds to calculate numbers of node-independent paths for all pairs of vertices. In Fig. 3 we show the resulting minimum spanning tree for the network. The structure visible in this figure reflects closely the known scientific organization of the institute. The largest component, shown at the top of the figure, comprises 118 vertices,

or about 44% of the total, and represents three subject areas, demarcated by the dotted lines. The uppermost of the three is a group of researchers working predominantly on the structure of RNA, and is spearheaded by the scientists represented by vertices 3, 4, 5, 34, and 51. Below that on the left is a group working on mathematical models in ecology, spearheaded by the scientist numbered 2. To the right is a group working in statistical physics, spearheaded by the scientists numbered 1, 12, and 16.

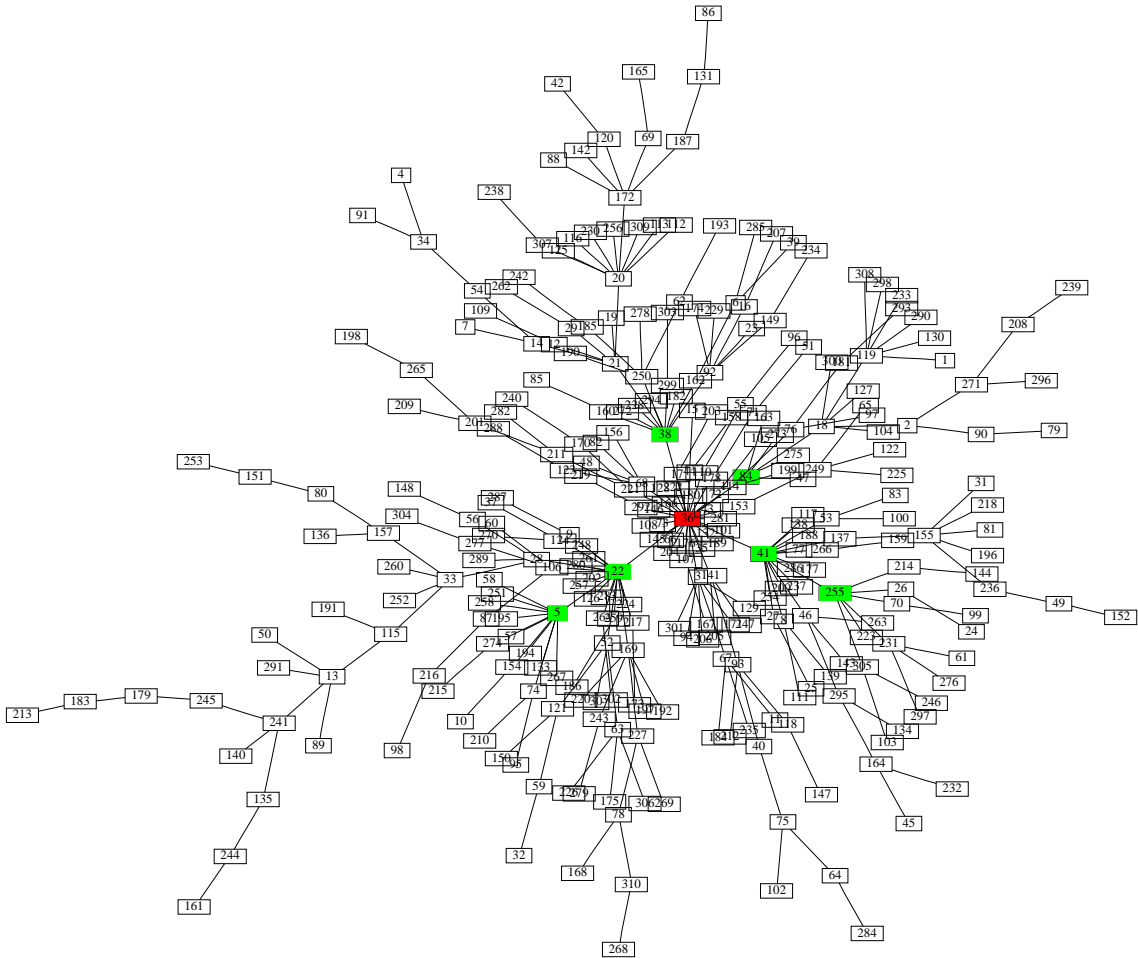The two next largest components of the graph also rep-

FIG. 4: Minimum spanning tree of the network of biotech companies discussed in the text.

resent known groupings within the institute. The second largest, shown at the bottom left of the figure is a group working on HIV, led by the researcher represented by vertex 74. The smaller component to the right of that is a group working on immunology, led by scientist 26. Thus it appears that the numbers of node-independent paths, and our approximation to them using our algorithm, are extracting a significant amount of structure from this particular network.

It may appear strange that the research community of an institute which is ostensibly interdisciplinary divides so clearly along lines of research topic. One might think that the point of interdisciplinary research is precisely to avoid such divisions. This however is fallacious: it is a mistake to assume that working in interdisciplinary research necessarily means you have to work in *all* disciplines. In fact, most researchers concentrate on only one or two areas. What makes the work interdisciplinary is that those areas are often not the areas in which the scientist in question received their original training. Of course, information about scientists' training is not contained in

the collaboration network; one would need other data, such as educational records, to detect interdisciplinary work. Nonetheless, the Santa Fe Institute is an interesting subject for study because there *are* collaborative ties between people with widely different interests, something which is rare in more traditional research environments.

B Biotechnology firms

Our second example application is to a network of formal inter-organizational relationships made by dedicated biotechnology firms (DBFs) focusing on research on medicines for humans. Collaborative ties of finance, R&D, commercialization, and licensing, connecting biotech firms, pharmaceuticals, finance and venture capital, universities, research institutes, and government agencies underwent considerable growth during the period 1988–1999. The biotech industry emerged from this period with an intensively networked form of organization (Powell 1996, Powell *et al.* 2001). The data used

here where compiled by W. Powell and K. Koput from the industry journal *BioScan*. Two firms are considered to have a symmetric tie if a contract is reported by a DBF. We have selected for analysis the 310 connected firms from among the 445 DBFs in the network. Figure 4 shows the minimal spanning tree.

The firm at the center of the figure is Genentech (36, red), one of the industry leaders, immediately surrounded by other major players (green) such Genzyme (41), Chiron (22) and, somewhat further away, Genetics Institute (38), CellTech (84), and then ArQule (255) and Amgen (5). This concentric structure of major stars with retinue surrounded by secondary stars and their retinue, and so on out to the margins of the graph, correlates well with descriptions of the cohesive core of the industry given by Powell *et al.* (2001). It also contrasts markedly with the collaboration network of the previous section, in which, rather than a concentric layered structure, we saw clear separate work-groups surrounding individual group leaders.

VI. FINDING K-COMPONENTS

We can use the algorithm described here to find subgraphs of nodes within a network in which all nodes are connected by at least k node-independent paths. Once we have the numbers of node-independent paths between all pairs finding such subgraphs is a trivial matter of creating the graph in which there is an edge connecting only those pairs with at least k paths on the original graph, and then finding all components. These subgraphs are the extracohesive blocks discussed in the introduction (White and Harary 2001). They are similar but not identical to the k-components of the graph. The difference is that a k-component is a subgraph in which all pairs of nodes are connected by at least k node-independent paths which each run entirely within the subgraph. The extra condition that the paths must run with the subgraph makes k-components harder to calculate than extracohesive blocks. However, our algorithm can be modified to calculate them also as follows. Every k-component $\mathcal{S}$ of a graph will be a subgraph of an extracohesive block $\mathcal{B}$ at level k. Hence, by taking the subgraph $\mathcal{B}$ and applying the algorithm again, a smaller subgraph is found for which node-independent paths are computed internally. By successive iterations of this procedure, k-components can be found. The success of our algorithm when applied to the test graphs of Section IV suggests that in fact these subgraphs should be a good approximation to the true k-component of the network.

VII. CONCLUSIONS

In this paper we have introduced a fast new algorithm for computing lower bounds on the numbers of node-independent paths between the nodes of a network. In practice, the algorithm appears to give excellent bounds on path counts, giving path counts which are in error less than 1% of the time on any of the graphs tested, and in many cases agreeing perfectly with exhaustive enumeration methods. The algorithm runs in time linear in the size of the network, a huge improvement over the exhaustive enumeration methods which take time exponential in network size. This makes possible the calculation of numbers of node-independent paths on much larger networks than have previously been feasible. We have given two applications of the algorithm to networks of moderate size (*circa* 300 nodes), for which it showed short computation times ($\lesssim$ 100 seconds) and produced useful results that appear to capture the cohesive structures of the networks. The algorithm also has potential applications in the calculation of k-components for networks.

REFERENCES

[1] B. Bollobás, *Random Graphs,* Academic Press, New York (1985).

[2] Chartrand, G. and L. Lesniak 1996 *Graphs and Digraphs,* 3rd edition, Chapman and Hall, London.

[3] Hage, Per and Frank Harary 1991 *Exchange in Oceania,* Clarendon Press, Oxford.

[4] Harary, F. 1969 *Graph Theory,* Addison-Wesley, Reading, MA.

[5] Menger, Karl 1927 "Zur allgemeinen Kurventheorie," *Fundamenta Mathematicae* 10:96–115.

[6] Powell, Walter W. 1996 "Inter-organizational collaboration in the biotechnology industry," *Journal of Institutional and Theoretical Economics* 120:197–215.

[7] Powell, Walter W., Douglas R. White, Kenneth W. Koput, and Jason Owen-Smith 2001 "Evolution of a science-based industry: Dynamic analyses and network visualization of biotechnology," Santa Fe Institute working paper.

[8] Schwimmer, Erik 1973 *Exchange in the social structure of the Orokaiva: traditional and emergent ideologies in the Northern District of Papua New Guinea,* St. Martin's Press, New York.

[9] White, Douglas R. and Frank Harary 2001 "The cohesiveness of blocks in social networks: Connectivity and conditional density," *Sociological Methodology,* in press.

[10] Zachary, Wayne W. 1975 "The cybernetics of conflict in a small group: An information flow model," unpublished masters thesis, Department of Anthropology, Temple University.

[11] —1977 "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research* 33:452–473.