Bounded Arboricity to Determine the Local Structure of Sparse Graphs*

Gaurav Goel[1,2] and Jens Gustedt[1,3]

[1] INRIA Lorraine, France
[2] IIT Delhi, India
[3] LORIA, France

Abstract. A known approach of detecting dense subgraphs (*communities*) in large sparse graphs involves first computing the *probability vectors* for *short random walks* on the graph, and then using these probability vectors to detect the communities, see Latapy and Pons [2005]. In this paper we focus on the first part of such an approach *i.e.* the computation of the probability vectors for the random walks, and propose a more efficient algorithm for computing these vectors in time complexity that is linear in the size of the output, in case the input graphs are restricted to a family of graphs of bounded arboricity. Such classes of graphs cover a large number of cases of interest, *e.g* all minor closed graph classes (planar graphs, graphs of bounded treewidth etc) and random graphs within the preferential attachment model, see Barabási and Albert [1999]. Our approach is extensible to other models of computation (PRAM, BSP or out-of-core computation) and also w.h.p. stays within the same complexity bounds for Erdős Renyi graphs.

1 Introduction and Overview

Consider a few real world large sparse graphs — the World Wide Web (WWW) graph where the vertices are HTML pages connected by links (edges) pointing from one page to another, the social acquaintance network where the vertices represent people and the edges represent the association between them, the graph representing the citation pattern of scientific publications with the vertices being the publications and the edges being the links to the articles cited in a publication, or for that matter the collaboration graph of movie actors with the vertices representing actors and edges joining actors which have worked together in at least one movie. All of these graphs and most other real world sparse graphs have a unique property — they have a low *arboricity*. Arboricity can be defined as follows:

Definition 1. *For a graph G the* arboricity $A(G)$ *is the smallest integer k for which there exists forests $T_1, \ldots, T_k$ which are subgraphs of G, such that their union is G.*

In fact, besides well-known examples of real world sparse graphs, all minor closed graph families, see *e.g.* Mader [1967], and all random graphs that are generated by the

* Research made possible by an internship grant for the first author at INRIA Lorraine, jointly accorded by the Lorraine Region, INRIA and the French ministery of foreign affairs.

F.V. Fomin (Ed.): WG 2006, LNCS 4271, pp. 159–167, 2006.

model discussed by Barabási and Albert [1999], also have a low value for arboricity, namely for each of these classes $\mathcal{C}$ in question there exists a constant $\delta_{\mathcal{C}}$ that bounds this parameter from above.

The method discussed in this paper is much inspired from previous work that explicitly or implicitly uses the property of bounded arboricity graphs to achieve algorithms of linear complexity, see *e.g.* in Cheriton and Tarjan [1976], Kannan et al. [1992], Bodlaender [1996], Gustedt [1998]. It divides the graph into portions based on the degrees of vertices, computes the required quantities only in one of the portions, passes the results to the remaining graph and finally recurses the procedure in this remaining graph.

One of the main tricks of such algorithms is that they may consider the bound on the arboricity $\delta = \delta_{\mathcal{C}}$ as being fixed and that this constant then appears only in the form of some function $f(\delta)$ in the complexity, independent of the input size, and, in our case, also of the output size.

1.1 Problem Definition

The problem of detecting dense subgraphs (*communities*) in large sparse graphs is inherent to many real world domains like social networking or internet computing. A known approach of detecting these communities involves first computing the *probability vectors* for *random walks* on the graph for a *fixed* number d of steps, and then using these probability vectors to detect the communities, see Latapy and Pons [2005]. Their algorithm takes $O(dnm)$ time where n and m are the number of vertices and edges in the graph. We focus on the first part of their approach i.e. computation of the probability vectors for the random walks, and propose a more efficient algorithm (than matrix multiplication) for computing these vectors in time complexity that is linear in the size of the output, in case the input graphs are restricted to a family of graphs for which the arboricity bounded by some constant δ.

The fact that the number d can be considered constant will be used extensively by the algorithms that we will propose. The complete expression of the complexity would be of the form $f(\delta, d) \cdot (N + M)$, for some function f of two parameters. But in this extended abstract we will not have the room for providing all the details. So far, the values for d that have been shown of practical use have been fairly small, usually smaller than 10.

Input: An undirected graph $G = (V, E)$ with average degree of z and a bounded arboricity $A(G) \leq \delta$ for some fixed constant δ. Once a random walk is placed at a particular vertex v, the probability of choosing any of the outgoing edges in the next step is the same p for all edges and the probability of staying at the same vertex in the next step, the *stationary probability* at v, is $q_v = (1 - (p * deg_v))$ where deg_v is the degree of vertex v.

Size of Input: $O(n + m)$ where $n = |V|$ is the number of vertices and $m = |E|$ is the number of edges in the given graph. Let this be denoted by N.

Task: Compute a set of vectors $P_1(v), \ldots, P_d(v)$ for every vertex v which contains the probabilities of reaching all other vertices in exactly $i = 1, \ldots, d$ steps if a random walk is started from v. For computational efficiency, each probability vector is maintained as a list of tuples, each containing a vertex number and the non

zero probability of reaching that vertex in i steps from v through a random walk. Vertices which cannot be reached in i steps from v are not present in this list.

It is easy to see that the total size of the probability vectors that we want to compute may be much larger than the input size. This is due to the fact even if the average degree of our graph will be bounded, there may be vertices with a very high degree. All neighbors of such a high degree vertex will see each other with a random walk of distance two, and so their probability vectors will have at least the size of this neighborhood. So we may not expect algorithms to solve our problem in time that is proportional to the size of the input, we have to consider output complexity as well.

Size of Output: The total number of probability vectors obtained are $d \cdot n$ and the length of each one of them may be as large as n. Let the total size of the output be denoted by M. We will always assume that $N \leq M$.

1.2 Our Approach and Result

Our algorithm divides the given graph (of bounded arboricity) into a *core* and a *periphery* based on the average degree of the vertices. Computations are first done on the periphery, then the information is passed on to the core. The core in itself is a graph of bounded arboricity and thus the procedure is recursed on the core. Once the innermost core is reached then the direction of flow of information changes and the information starts flowing from the core to the periphery till it reaches the outermost periphery. The process of exchanging information between core and periphery is repeated a number of times which in our particular case is a function of the length of the random walks d, to collect all probability vectors for those paths that cross several times between core an periphery. This approach will make it possible to compute the probability vectors in time complexity that is linear in the number of non-zero entries of these vectors i.e. in $O(M)$ time which is a major improvement over the existing algorithm.

Moreover, we will always be able to charge all computations to vertices that have bounded degree in the graph under investigation. Thereby it is possible to parallelize our algorithm efficiently for the PRAM (Fortune and Wyllie [1978]), BSP (Valiant [1990]) or PRO (Gebremedhin et al. [2002]) models of parallel or distributed computation. By arguments as given by Dehne et al. [1997] and Gustedt [2003] such a paralellization may then also be extended to out-of-core computations. The later is particularly interesting for practical problems, since the actual hurdle for large scale computations on massif graphs are memory and not time constraints.

It is also possible to extend our results to other classes of graphs, namely graphs that are randomly chosen according to the Erdős-Renyi model G_p. Almost certainly these have a low average degree, too, and with high probability the recursive procedure that we propose will only fail on some very small iterated 'core' graph where all vertices have high degree. Since the problem on such a small core could be solved directly without a major impact on the total complexity, these graphs are tractable by our approach w.h.p, and in particular we obtain linear complexity on average. But due to space restrictions we will no be able to discuss this in detail.

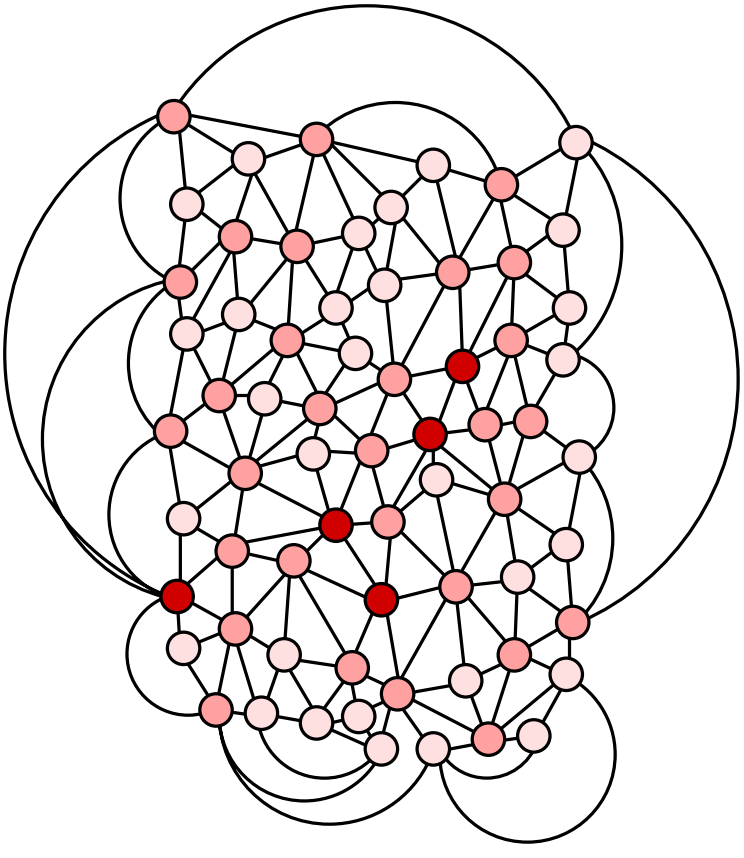

Fig. 1. A planar graph with two levels of periphery. The first level consists of all vertices of degree 5 or less (light colored). In the remaining graph, the *core*, the second level of periphery are then the vertices that have at most degree 5 in that graph (medium colored).

2 Basic Facts

To obtain a lower bound for the arboricity of a graph we may simply compute the quotient of the number of edges and the size of any of its spanning trees.

Definition 2. *Let $G = (V, E)$ be a connected graph. The* tree-density *of G is given by*

$$\tilde{A}(G) = \begin{cases} \left\lceil \frac{|E|}{|V|-1} \right\rceil & |V| > 1 \\ |V| & \textit{otherwise.} \end{cases} \tag{1}$$

If G is not connected $\tilde{A}(G)$ is the maximum tree-density of its connected components.

Observation 1. *Let $G = (V, E)$ be a graph and $G' = (V', E') \subseteq G$ be a subgraph. Then $\tilde{A}(G') \leq A(G)$.*

Proof: Suppose $A(G) \leq \delta$ and let $(F_1, \ldots, F_\delta)$ be a tree-partition of G. Now let $(F'_1, \ldots, F'_\delta)$ be the forests that correspond to G', *i.e.*, set $F'_i = F_i \cap G'$ for all $i = 1, \ldots, \delta$. Each of the F'_i contains at most $|V'| - 1$ edges and altogether these forest have at most $\delta(|V'| - 1)$ edges. □

Surprisingly the converse of the above observation is also true, *i.e.*, the tree-density of each subgraph also gives a lower bound on A. The following deep theorem holds

Theorem 1 (Nash-Williams [1961])

$$\textit{Let } G \textit{ be a finite graph. Then } A(G) = \max_{G' \subseteq G} \tilde{A}(G').$$

For an easily achievable converse we loose a factor of 2 in the estimation.

Observation 2. *Let $G = (V, E)$ be a graph such that each subgraph $G' \subseteq G$ has $\Delta(G') \leq \delta$ then $A(G) \leq \lfloor \delta \rfloor$.*

Proof: We proceed by induction on the number of vertices in G. Let v be a vertex of minimum degree in G and $G' = G \setminus \{v\}$. Clearly $deg_G(v) \leq \lfloor \delta \rfloor$. We have $\Delta(G') \leq \delta$ by assumption so by the induction hypotheses there is a partition of G' into forests $(F'_1, \ldots, F'_{\lfloor \delta \rfloor})$. Now we may obviously extend $(F'_1, \ldots, F'_{\lfloor \delta \rfloor})$ to a partition $(F_1, \ldots, F_{\lfloor \delta \rfloor})$ of G by making v a leaf in each of the forests. □

Another fact that we will need later concerns not the structure of our graphs but the structure of the probability vectors. The *symmetric* probability distribution which we assume deviates a bit from what is found in literature but it ensures the following useful property for the graph:

Lemma 3. *If in G the probability of going from a vertex v_1 to v_m through a m step random walk is ρ, then the probability of going from v_m to v_1 through a m step random walk is also ρ.*

Proof: Consider a m step path $v_1, v_2, \ldots, v_m$, not necessarily all distinct. The total probability of this path being chosen during a random walk of length m depends on the number of steps m' in the given path that join two distinct vertices. These steps contribute a factor of $p^{m'}$. The stationary phases of the path contribute by the number of times a particular vertex is repeated on the path. The order in which such a vertex v appears on the path is not important for the probability, it only contributes with a factor of $q_v^{m_v - 1}$, where q_v is the stationary probability of v and m_v is its multiplicity on the path.

So the inverse of the path is chosen with the same probability when starting a walk from v_m. Since this holds for all path of length m the claim holds. □

3 The Algorithm

Algorithm 1 shows a generic recursive procedure for which in the following we will instantiate the remaining parts to then compute the probability vectors we are interested in. Observe that this algorithm doesnot even request that the graph has an arboricity within some limits; in view of Theorem 1 it could even be used to prove or disprove a given bound of the arboricity. But for the sake of simplicity we will assume that the input graph has an arboricity that is bounded by some fixed constant δ.

Algorithm 1. Recursive shelling to solve problem $\mathcal{P}(G, \mathcal{D})$.

Input: A connected Graph $G = (V, E)$ with eventually some local problem specific data $\mathcal{D}(G)$ attached to vertices and/or edges.
Output: $G = (V, E)$ together with the local information for $\mathcal{P}(G)$ attached to vertices and/or edges.

if *G has degree bounded by 2δ* **then**
direct Solve $\mathcal{P}$ directly for G and **return** $\mathcal{P}(G)$;
else
Let $z = \lceil \Delta_G \rceil$;
split Let $V_0 \subseteq V$, the *periphery*, be the vertices of degree z or less;
Let $V_1 = V \setminus V_0$, the *core*;
$G_i = (V_i, E_i) = V|V_i$, for $i = 0, 1$;
recurse Recurse on the connected components of G_0 and G_1 to obtain $\mathcal{P}_0^0$ and $\mathcal{P}_1^0$;
Let $E_2 = E \setminus (E_0 \cup E_1)$;
loop **foreach** $i = 1, \ldots, k$ **do**
pull Via E_2, pull $\mathcal{P}_1^{2i-2}$ to G_0 and compute $\mathcal{P}_0^{2i-1}$ and $\mathcal{P}_0^{2i}$;
push Via E_2, push $\mathcal{P}_0^{2i-1}$ to G_1 to obtain $\mathcal{P}_1^{2i-1}$ and $\mathcal{P}_1^{2i}$;
combine Combine $\mathcal{P}_0^0, \ldots, \mathcal{P}_0^{2k}, \mathcal{P}_1^0, \ldots, \mathcal{P}_1^{2k}$ into $\mathcal{P}(G)$;
return $\mathcal{P}(G)$;

Steps split and recurse: Dividing the graph in to a core and a periphery. The main trick of the algorithm is to divide the graph in two parts, low degree vertices form the *periphery* and high degree vertices the *core*. Note that z is bounded by the arboricity $A(G)$. The core G_1 is a subgraph of G and thus $A(G_1) \leq A(G) \leq \delta$. By definition, recursing on G_0 will immediately run into the case direct.

Recursion returns the probability vectors for the paths that lie entirely inside the core or the periphery. Then, for the final result we have to take into account all paths of length at least d that exist in the entire graph and which cross the boundary between the core and the periphery, *i.e.* that use edges in E_2. In Algorithm 1, $\mathcal{P}_0^i$ and $\mathcal{P}_1^i$ denote the probability vectors of paths restricted to use exactly i edges from E_2.

Step direct: Computations inside degree bounded graphs. Initially each vertex has a probability vector consisting of its neighbors and the probability of reaching them in 1 step of a random walk. All paths of maximal length d starting in a particular vertex v can be enumerated in time $(2\delta)^d$ and the probabilities of all those paths can be maintained in the same complexity. Then, for all target vertices w that can be reached from v in that way, the corresponding probability can be computed. Since $(2\delta)^d$ is considered to be constant, all this computation is proportional to the size of the probability vector.

In total, this means that the computation is linear in the size of the output.

Steps pull: Computations in the periphery. Suppose that we have stored all probabilities for paths that contain exactly $2i - 2$ edges from E_2. Any path that contains $2i - 1$ such edges has at least one endpoint in the periphery. For all vertices in the periphery

we may collect all vertices that can be reached with exactly one edge in E_2 and compute the corresponding probabilities.

Now consider paths that contain $2i$ edges from E_2 and that have both endpoints v and v' in the periphery G_0. They can be divided into three parts: paths from v and v' into the core and a path that uses $2i-2$ edges of E_2 and for which the probability is registered in $\mathcal{P}_1^{2i-1}$.

Step push: Passing of information — periphery to core. With Lemma 3, for every vertex in the periphery look at its d probability vectors, search for probabilities of reaching vertices in the core which has a neighbor in the periphery, and reverse this information, *i.e.*, if a vertex v_j in the periphery has the knowledge that it can reach another vertex v_k in the core (which has a neighbor in the periphery) in m steps of random walk with probability p, then give this information to v_k, telling it that it can reach v_j in m steps of random walk with probability p. This step will generate probability vectors $\mathcal{P}_1^{2i-1}$ in the core.

Similar considerations for the case of paths using an even number of edges in E_2 and that start and end in G_1 lead to the computation of $\mathcal{P}_1^{2i}$. For complexity considerations it is important to note that the corresponding computations can be charged to nodes in the periphery. If a vertex v in the periphery knows that it can reach vertex v' in the core by crossing E_2 exactly once it may use the information in $\mathcal{P}_0^{2i-1}(v)$ to provide information for $\mathcal{P}_1^{2i}(v')$. To avoid duplication, this *stitching* of information needs to be done only by those vertices in the periphery which have a neighbor in the core, *i.e.* the endpoints of edges in E_2 that are in the periphery.

3.1 Correctness and Complexity

It is now easy to see that the sets of paths that are used for the different vectors $\mathcal{P}_{0,1}^{0,\ldots,2k}$ are mutual disjoint and thus their probabilities add up to give the total probabilities. Therefore correctness of the algorithm follows immediately.

For the complexity, observe that the **foreach**-loop is executed $O(d)$ times and that all push and pull computations are proportional to modifications on the probability vectors that they produce. Since δ and d are considered to be constant, all calls (without accounting for recursion) are linear in the output they produce. Observe here that not necessarily all vertices of the core are even touched by these updates. They are only touched and new non-zero probabilities are added to their vectors, if the merge of the periphery and core information discovers vertices that have not been reachable inside core or periphery alone.

So, when summing up over all calls the time is proportional to the produced output.

It remains to show that the recursion and its touching of vertices and edges does not worsen the running time. The recursive calls on the periphery G_0 always have depth 1, since by definition G_0 only has vertices of low degree. So the only call that could lead to a deeper recursion (and higher complexity) is the one for the core G_1. V_1 are the vertices in V that have a degree that is above $2\delta \geq 2$ and so we have that $|V_1| \leq |V|/2$. Thereby the total sum of the edges of the graphs that occur during recursion is bounded by $\sum_i 2^{-i}\delta|V| \leq 2\delta|V|$, and thus linear in our setting.

4 Conclusion and Further Work

We proposed an *output sensitive* algorithm for computing probability vectors for short random walks on graphs with bounded arboricity. Thereby it applies to a large number of graph classes which are of theoretical and practical interest. The good complexity behavior of our algorithm also extends to other computational models such as PRAM, BSP or PRO and also to some classes random graphs w.h.p.

The approach we discussed gives an alternative and more efficient way to compute these probability vectors, but it doesnot change the community structure that emerge from the approach discussed by Latapy and Pons [2005]. However, the second phase of using clustering algorithms that was described by them is very time consuming, too, so it might be appropriate to deviate from that approach, even by allowing for algorithms that would give a community structure that is defined slightly differently. This part of the problem is still open. A good approach could be to use Union-Find strategies which have linear complexity under some conditions (see Fiorio and Gustedt [1996]). Any such approach would require an oracle on the basis of which smaller communities would be joined to form bigger ones till a community structure emerges that is satisfactory according to some predefined criteria. We hope to do some work in this direction also.

Bibliography

W. Aiello, F. Chung, and L. Lu. Random evolution in massive graphs. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 510–519, 2001. URL `citeseer.ifi.unizh.ch/article/aiello01random.html`.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

Hans Leo Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(1305-1317), 1996.

David Cheriton and Robert Endre Tarjan. Finding minimum spanning trees. *SIAM J. Computing*, 5:724–742, 1976.

F. K. H. A. Dehne, W. Dittrich, and D. Hutchinson. Efficient external memory algorithms by simulating coarse-grained parallel algorithms. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 106–115, 1997.

Christophe Fiorio and Jens Gustedt. Two linear time union-find strategies for image processing. *Theoretical Computer Science*, 154:165–181, 1996.

Steven Fortune and James Wyllie. Parallelism in random access machines. In *10th ACM Symposium on Theory of Computing*, pages 114–118, May 1978.

Assefaw Hadish Gebremedhin, Isabelle Guérin Lassous, Jens Gustedt, and Jan Arne Telle. PRO: a model for parallel resource-optimal computation. In *16th Annual International Symposium on High Performance Computing Systems and Applications*, pages 106–113. IEEE, The Institute of Electrical and Electronics Engineers, 2002.

Jens Gustedt. Minimum spanning trees for minor-closed graph classes in parallel. In *Symposium on Theoretical Aspects of Computer Science*, pages 421–431, 1998.

Jens Gustedt. Towards realistic implementations of external memory algorithms using a coarse grained paradigm. In *International Conference on Computational Science and its Applications (ICCSA 2003), part II*, number 2668 in LNCS, pages 269–278. Springer, 2003.

S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM Journal On Discrete Mathematics*, 5:596–603, November 1992. URL `citeseer.ifi.unizh.ch/kannan92implicit.html`.

Matthieu Latapy and Pascal Pons. Computing communities in large networks using random walks. In *ISCIS'05*, pages 284–293, 2005.

W. Mader. Homomorphieeigenschaften und mittlere Kantendichte von Graphen. *Math. Ann.*, 174:265–268, 1967.

Crispin St. John Alvah Nash-Williams. Edge-disjoint spanning trees of finite graphs. *J. London Math. Soc. (2)*, 36:445–450, 1961.

L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8): 103–111, 1990.