An Efficient Ear Decomposition Algorithm

Debarshi Dutta[1], Kishore Kothapalli[1], G. Ramakrishna [2], Sai Charan Regunta [2], and Sai Harsh Tondomker [2]

[1]International Institute of Information Technology, Hyderabad. India.
[2]Indian Institute of Information Technology Chittoor, Sri City, India.

An ear decomposition of a graph G is a partition of the edge set of G into a sequence of edge-disjoint paths, such that only the end vertices of each path appear in earlier paths. For a graph on n vertices and m edges, the state-of-art algorithm for obtaining an ear decomposition by Schmidt takes $O(m + n)$ time. We design and implement a new algorithm to obtain an ear decomposition for a biconnected graph, whose running time $O(m+n)$. In practice, however, our experiments reveal that, the proposed algorithm runs at least 2 times faster than Schmidt's algorithm. The speedup increases as the graph gets denser.

1 Introduction

Obtaining an ear decomposition of a graph is an important problem in the context of graph algorithms. An ear decomposition of a graph is used in several other graph algorithms such as testing connectivity, s-t-numbering and planarity-testing [6]. Also, an ear decomposition has been used as a paradigm to obtain parallel algorithms for various problems. Whitney has introduced the notion of ear decomposition to characterize biconnected graphs [7]. An *ear* of a graph G is a maximal path whose internal vertices have degree 2. An *ear decomposition* of a graph $G = (V, E)$ is a partition of E into a sequence $(P_0, P_1, \cdots, P_k)$, such that (i) P_0 is a cycle, (ii) for each $i \geq 1$, P_i is an ear of $P_0 \cup \ldots \cup P_i$. An ear decomposition $(P_0, P_1, \ldots, P_k)$ of G is an *open ear decomposition* if the end points of all the ears P_i, $i \geq 1$, are distinct.

Past Work. The concept of ear decomposition is very well studied both structurally and algorithmically. An ear decomposition is used to characterize biconnected graphs [7]. Further, the notion of nested ear decomposition and nice ear decompositions are developed to characterize series-parallel graphs and polygonal 2-trees, respectively [2, 5]. For a graph on n vertices and m edges, algorithms to obtain an ear decomposition in $O(m + n)$ time are known for a long time. Lovász has designed an algorithm for the first time to obtain an ear decomposition in parallel framework [4]. The state-of-art serial algorithm to construct an ear decomposition is proposed by Schmidt [6]. His algorithm is based on depth first search spanning tree and is simpler to visualize. The algorithm of Schmidt also runs in time $4m + O(n)$.

Motivation and Our Contribution. To the best of our knowledge, there are no studies on computing ear decomposition from a practical perspective in serial computing. The state of art algorithm by Schmidt is simple and elegant [6]. However, this algorithm has not been explored in practice. The main objective of this work is to obtain an ear decomposition algorithm that works well both in theory and practice. In this paper, we present an $O(m + n)$ algorithm that offers an improvement to the algorithm of Schmidt in the practical setting. In particular, we show that, a large number of edges of a graph are *redundant* in the process of obtaining an ear decomposition. Removing such redundant edges in a prior pre-processing step can often result in practical improvement. Our characterization of redundant edges (trivial ear) is based on a similar notion that is identified in the context of biconnectivity [1]. In practice, our algorithm runs at least 2 times faster than Schmidt's algorithm on graphs having $\Omega(n \log n)$ edges.

2 Algorithm for Ear Decomposition

We use standard graph terminology from [7]. For a graph G, let $n = |V(G)|$ and $m = |E(G)|$ denote the number of vertices and edges in G, respectively. A graph is *biconnected* (*2-vertex connected*) if it does not contain a cut-vertex. An edge e in a biconnected graph is *non-essential* if the graph remains biconnected after the removal of e. For $i \geq 2$, an ear P_i in an ear decomposition is a *trivial ear*, if the number of edges in P_i is one. In the rest of the paper, G denotes an unweighted and undirected biconnected graph.

The main idea in Schmidt's algorithm for computing an ear decomposition is to obtain a depth first search tree T and process all the non-tree edges (edges in $G - T$) to construct $m - n + 1$ ears. The time required to perform these two steps is $4m + O(n)$, where at most $2m + O(n)$ is required in the individual steps.

Algorithm 1 shows an overview of our approach to compute an ear decomposition. At a higher-level, the main idea in our approach is to filter many non-essential edges and compute an ear decomposition on the rest of the graph.

Algorithm 1: An algorithm to find an ear decomposition of a biconnected graph G

```
1 Construct a breadth first search (BFS) spanning tree $T$ of $G$ ;
2 Construct a spanning forest $F$ from $G'$, where $G' = G - T$ ;
3 Find an ear-decomposition $\mathcal{P}$ of $T \cup F$ using Schmidt's algorithm ;
4 return the sequence of ears in $\mathcal{P}$ and the edges in $G''$ as trivial ears, where $G'' = G' - F$;
```

The proof of correctness of Algorithm 1 is shown in Theorem 3, using Lemma 1 and Lemma 2.

Lemma 1 ([7])**.** *A graph G is biconnected (2-vertex connected) if and only if G has an open ear decomposition.*

Lemma 2 ([1])**.** *Let T be a* BFS *spanning tree of G and F be a spanning forest in $G - T$. Then, the edges of each connected component of $G - T$ are in one biconnected component. The number of biconnected components in G and $T \cup F$ is same.*

Theorem 3. *For a biconnected graph G, let T be a* BFS*-spanning tree of G and F be a spanning forest in $G - T$. Then there is an ear decomposition $\mathcal{P}$ of G in which every edge in $G - (T \cup F)$ is a trivial ear.*

Proof. From Lemma 2, $T \cup F$ is biconnected. Then, by Lemma 1, there is an open ear decomposition $\mathcal{P}'$ of $T \cup F$. As a result, $\mathcal{P}'$ with each edge in $G - (T \cup F)$ being a trivial ear, becomes an open ear decomposition of G with the mentioned property. □

In Algorithm 1, Line 1 and Line 3 take $2m + O(n)$ time and $O(n)$ time, respectively. Constructing a spanning forest F from $G - T$ efficiently in Line 2 is a more involved task. If we use breadth first or depth first traversals, then Line 2 consumes $2m + O(n)$ time, the total running time is $4m + O(n)$. This matches with Schmidt's algorithm, and hence this is not a promising idea. If we use disjoint-set-forest datastructure with union-by-rank and path-compression, the Line 2 consumes $m\alpha(n) + O(n)$ time, the total run time is $2m + \alpha(n)m + O(n)$. A linked-list based disjoint-set data-structure requires $m + O(n \log n)$ time to compute the task in Line 2. However, as the data-locality is very poor in linked-lists, our experiments reveal that this idea does not beat Schmidt's algorithm in practice.

To make our idea work, we introduce a randomized algorithm Algorithm 2, shown below for the implementation of Line 2 in Algorithm 1.

```
Algorithm 2: Algorithm to implement Line 2 in Algorithm 1
1 $X = \{v \in V(G) \mid degree(v) \text{ in } T = degree(v) \text{ in } G\}$, $F = \emptyset$ ;
2 for each vertex $u \in V(G) - X$ do
3     append each edge incident at $u$ to $F$ with probability $\frac{\log n}{n}$ ;
```

The key idea in the randomized algorithm is to choose a sparse spanning connected subgraph of $G - X$, instead of a spanning forest of G. For each vertex $u \in V(G)$, each edge incident at u is sampled with probability $\frac{\log n}{n}$ and all the sampled edges in the random process form a spanning connected subgraph of $G - X$. This insight is inferred from our experiments and support the correctness of Algorithm 2. Let the number of vertices and edges in the input graph H be n vertices and $\Omega(n \log n)$ edges, respectively. If H' is a spanning subgraph of H that is constructed by sampling each edge of H with probability $\frac{\log n}{n}$, then we claim that H' is connected with high probability. The proof of this claim is open and Lemma 4 would be helpful to derive the proof. The expected number of edges in F is in $O(n \log n)$. If we assume that each edge can be sampled in constant time, then Algorithm 2 runs in $m+O(n \log n)$ time, and hence the run time of Algorithm 1 is $3m + O(n \log n)$. This run time is further reduced to $2m + O(n \log n)$ time in next section. Since m is $\Omega(n \log n)$, our algorithm runs in $O(m + n)$ time. However, our algorithm performs well in practice as m increases beyond $\Omega(n \log n)$.

Lemma 4. *[3] A random graph $G(n, p)$ on n vertices and edge probability p is connected with very high probability, if $p = \frac{\ln n}{n}$.*

3 Implementation Details and Experiments

In this section, we describe the lower level details of our algorithm. Without loss of generality, let us assume that the vertices in the input graph are numbered from 1 to n. We use the *compressed sparse row* (CSR) representation to store the input graph and as well as the intermediate graphs. A CSR of a graph G consists of two arrays, namely a vertex array $V[\ ]$ and an edge array $E[\ ]$. For each i, $1 \leq i \leq n$, $V[i]$ denotes an index in $E[\ ]$ such that all the neighbors of i are stored in $E[\ ]$ at locations $V[i], \ldots, V[i+1] - 1$.

We recall the construction of spanning forest F in Algorithm 2. Each edge incident at a vertex u is sampled with probability $\frac{\log n}{n}$. Then, the expected number of edges in F that are incident on u for a dense graph is at most $\log n$. In practice, this step consumes more time as every edge needs to be processed. One way to perform this step is as follows. For sampling k edges that are incident on a vertex i, we first choose k random indices whose range is $V[i], \ldots, V[i+1]-1$, and move these edges from G to F. If the number of edges incident at a vertex are less than k, then we move all of them from G to F. The run-time of Algorithm 2 as per this idea is $O(n \log n)$, and thus the run-time of our main algorithm is $2m + O(n \log n)$. The edges that are not sampled from CSR of G will be remained as trivial ears.

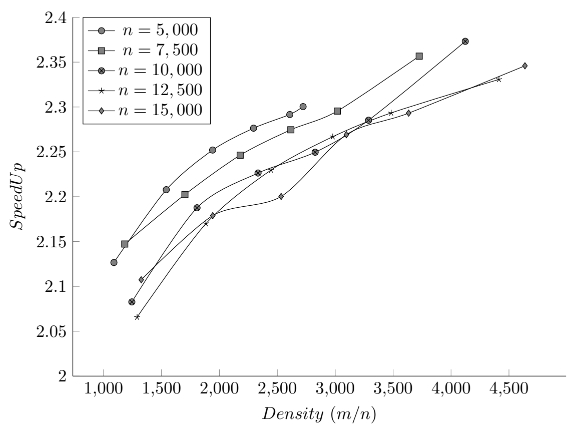

Figure 1: Density Vs SpeedUp

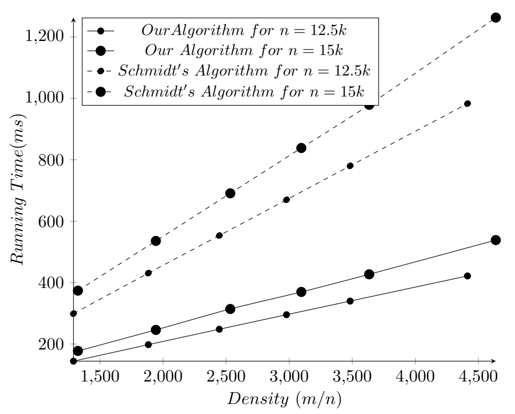

Figure 2: Schmidt's Vs Our Algorithm

Experimental Results

We implement our algorithms in C using `gcc` 4.8.5 compiler. We use Intel(R) Xeon(R) CPU E5-2650 v3 processor, which is based on x86_64 architecture. The frequency of the processor that we use in our experiments is 2.30GHz. The details on the number of vertices (n) and density (m/n) of the graphs considered in our experiments, which are constructed using GT-generator, are given in the left-side figure shown below. Our experiments reveal that the speedup is proportionate to the density of a graph. In other words, as the density of a graph increases, the speedup of our algorithm increases further and this is shown in Figure 1. The run-time of our algorithm against Schmidt's algorithm for $n = 15K$ and $n = 12.5K$, and for various densities are shown in Figure 2. In our experiments, the runtime plot against the density for $n = 5K$, $n = 7.5K$ and $n = 10K$ follow the similar trend shown in Figure 2. We conclude that, our algorithm runs at least 2 times faster than Schmidt's algorithm in practice.

Remark. Our algorithm can be used to obtain an approximated minimal biconnected graph. Finding the trade off between the quality of the solution and runtime by our algorithm against the state-of-art approximation algorithms to find a minimal biconnected graph is an interesting study.

References

[1] G. Cong and D. A. Bader. An experimental study of parallel biconnected components algorithms on symmetric multiprocessors (smps). *Inter. Par. and Dist. Proc. Symp.*, 2005.

[2] D. Eppstein. Parallel recognition of series-parallel graphs. *Inf. Comput.*, 98(1):41–55, 1992.

[3] P. Erdős and A Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, pages 17–61, 1960.

[4] L. Lovász. Computing ears and branchings in parallel. *Found. of Comp. Sci, 464–467*, 1985.

[5] N.S. Narayanaswamy and G. Ramakrishna. On minimum average stretch spanning trees in polygonal 2-trees. *Theor. Comput. Sci.*, 575(C):56–70, 2015.

[6] J. M. Schmidt. A simple test on 2-vertex- and 2-edge-connectivity. *Info. Proc. Lett.*, 113(7):241–244, 2013.

[7] Douglas B. West. *Introduction to graph theory - second edition.* Prentice Hall, 2001.