OPEX: Optimized Eccentricity Computation in Graphs

K. Henderson

November 15, 2011
OptEx: Optimized Eccentricity Computation in Graphs

Keith Henderson
Lawrence Livermore National Lab
keith@llnl.gov

Abstract

Real-world graphs have many properties of interest, but often these properties are expensive to compute. We focus on eccentricity, radius and diameter in this work. These properties are useful measures of the global connectivity patterns in a graph. Unfortunately, computing eccentricity for all nodes is \( O(n^2) \) for a graph with \( n \) nodes. We present OptEx, a novel combination of optimizations which improves computation time of these properties by orders of magnitude in real-world experiments on graphs of many different sizes. We run OptEx on graphs with up to millions of links. OptEx gives either exact results or bounded approximations, unlike its competitors which give probabilistic approximations or sacrifice node-level information (eccentricity) to compute graph-level information (diameter).

Keywords

Graph algorithms

1. Introduction

Given a connected, undirected, unweighted graph \( G = (V, E) \), a number of global and local properties of interest are often required for a variety of analysis techniques, including node classification, graph comparison, anomaly detection, and visualization. One of these properties at the node level is eccentricity, which measures the longest geodesic (shortest) path from a given vertex to any other vertex in the graph. The global counterparts of eccentricity are radius and diameter, which minimize and maximize eccentricity over \( V \), respectively.

Graph diameter has been studied in depth, both as a computational challenge [2] and to characterize real-world graphs [4]. Existing algorithms suffer from either scalability issues, probabilistic approximation, or failure to compute all node eccentricities in favor of computing global diameter. We present OptEx (Optimized Eccentricity), a fast algorithm for either exact or bounded approximate eccentricity, radius, and diameter. In the worst case, OptEx is \( O(|V|^2) \), as is true of any exact eccentricity algorithm. However, in several real-world graphs we demonstrate that OptEx runs orders of magnitude faster than existing algorithms. Additionally, using \( O(|V|) \) time before running OptEx we can often tell whether or not the optimizations will reduce runtime significantly.

The outline of the paper is as follows: Section 1 provides an introduction to this work. Section 2 presents an overview of the related work. Section 3 reviews definitions and introduces some theorems that will underpin our algorithms. Section 4 describes our proposed method. Section 5 presents experimental results on 4 real-world graphs. Lastly, Section 6 provides some concluding remarks.

2. Background and Related Work

There are three relevant competitors to OptEx: (1) the baseline approach, Johnson’s algorithm for computing all-pairs-shortest-paths, (2) HADI, a parallelizable algorithm for probabilistic approximation of eccentricity, radius, and diameter, and (3) the Iterating Bounds algorithm, a sampling approach that only approximates diameter.

Recall that eccentricity of a node is the length of the longest geodesic path from that node to any other node in \( V \), diameter is the maximum eccentricity among nodes in \( V \), and radius is the minimum eccentricity among nodes in \( V \). Let \( n = |V| \) and \( m = |E| \), and \( d \) be the graph diameter.

Johnson’s Algorithm. The simplest approach to computing eccentricity of a node \( u \in V \) is to compute geodesic paths \( u \sim v \) to each node \( v \in V \) and record the maximum path length. Johnson’s algorithm computes shortest paths between all pairs of nodes in \( O(n^2 \log n + nm) \) time. However, since we are dealing with unweighted graphs, the runtime can be decreased to \( O(n^2 + nm) \). This approach is exact, and in the worst case we can do no better for exact eccentricities. However, the method does not scale up to real-world graphs with millions of nodes and edges.

HADI. HADI [2] is a parallelizable algorithm that computes approximate eccentricity, radius, and diameter. It runs in \( O(dm) \) time and uses \( O(n \log n) \) space. An exact version of HADI can be implemented, which uses \( O(n^2) \) space. HADI does not compute any eccentricities exactly, but it provides a probability associated with the correctness of its approximations.

Iterating Bounds. A third method [3] only computes diameter, but runs in \( O(k(n + m)) \) time, where \( k \) is the number of iterations (a parameter). This method alternates between
For all pairs of vertices \( u, v \), the first inequality is simply a restatement of the second, which contradicts the definition of eccentricity. Thus (2) must be true, i.e., the longest geodesic from \( u \) passes through some articulation point \( v \) and then consists only of edges outside of \( \text{BCC}(u) \). Assuming (2) is false leads immediately to the conclusion that all edges in the geodesic are in \( \text{BCC}(u) \), proving (1) true.

**Theorem 2.** For all pairs of vertices \( u, v \) from \( V \), \( \text{sp}(u, v) \leq \text{ecc}(u) \leq \text{sp}(u, v) + \text{ecc}(v) \).

**Proof.** The first inequality is simply a restatement of the definition of eccentricity. To prove the second inequality, \( \text{ecc}(u) \leq \text{sp}(u, v) + \text{ecc}(v) \), one must simply observe that for any node \( w \in V \), a path \( u \sim v \sim w \) has length, at most, \( \text{sp}(u, v) + \text{ecc}(v) \) by the definition of eccentricity.

### 4. OpEx

The OpEx algorithm combines three optimizations to speed up computation of eccentricities in \( G \). The first, Articulation Pruning, takes advantage of biconnectedness to “skip” entire portions of each BFS that is performed. The second, Updating Bounds, uses Theorem 2 to maintain upper- and lower-bounds on \( \text{ecc}(u) \) for all nodes \( u \). Lastly, we borrow Double Sweep from the Iterating Bounds technique to ensure we are choosing good candidates for each search.

#### 4.1 Articulation Pruning

This algorithm computes exact eccentricities for all nodes in \( V \).

The first step is to compute all biconnected components and articulation points, which can be done in \( O(n + m) \) time using a well-known algorithm (note: the algorithm relies on depth-first-search, which should be implemented iteratively if scalability is an issue).

The next step is to do a BFS expansion from each articulation point \( u \). In addition to computing \( \text{ecc}(u) \), we record, for each BCC that \( u \) participates in, the longest path that does not include that BCC. This can be done with no additional asymptotic cost by adding some simple bookkeeping to the BFS algorithm.

Finally, we compute a BFS expansion from each remaining node \( w \). Any time an articulation point \( u \) is added to the OPEN list, record also the BCC \( B \) of the incoming edge to \( u \). Upon expanding \( u \), only expand edges \( u \rightarrow v \) s.t. \( \text{BCC}(u \rightarrow v) = B \). The other edges do not need to be expanded; we already recorded the longest geodesic from \( u \) that does not use edges in \( B \), and can simply use this path (plus \( w \sim u \)) as a candidate for the longest shortest path from \( w \).

Theorem 1 ensures the correctness of this algorithm.

#### 4.2 Updating Bounds

This algorithm can compute exact eccentricities for all nodes in \( V \), or can be tuned to stop after all eccentricities are known to within a given bound. It does not use Articulation Pruning (although we will combine them for the final algorithm).

The method is similar to Johnson’s all-pairs-shortest-paths algorithm, in that it (potentially) computes a BFS from each node. However, we will retain two additional values for
each node \( u \). \( ub(u) \) is an upper bound on \( ecc(u) \) and is initially \( n−1 \). \( lb(u) \) is a lower bound, initially set to 0.

After expanding a vertex \( u \), we know \( ub(u) = lb(u) = ecc(u) \) exactly, and we also have exact values for \( sp(u,v) \) for all \( v \in V \). Using Theorem 2, we can update \( ub(v) = \min(ub(v), sp(u,v)+ecc(u)) \) and \( lb(v) = \max(lb(v), sp(u,v)) \).

Given a threshold \( T \), we can simply stop when all nodes have \( ub(u)−lb(u) ≤ 2T \). This ensures that our estimate, \( ecc(u) ≈ (ub(u) + lb(u))/2 \) will be within \( T \) of the correct eccentricity. If \( T = 0 \) the algorithm is exact. In practice, even when \( T = 0 \) it is not unusual to avoid many BFS iterations as the bounds converge for many nodes in \( G \).

At each iteration, we choose as our BFS root the node that currently has the highest \( gap \), i.e. the node that maximizes \( ub(u)−lb(u) \).

### 4.3 Double Sweep

This technique is borrowed from the Iterating Bounds method for diameter computation. It is used in conjunction with the Updating Bounds method. The method is simple.

1. Perform a BFS from some node \( u \in V \). Record the furthest node \( v \) from \( u \) such that we have not performed a BFS from \( v \).

2. Perform a BFS from \( v \).

The Double Sweep is useful because the first step finds an eccentric node, and the second step finds the longest path from that node. We do a Double Sweep at every step of the Updating Bounds method, updating \( ub \) and \( lb \) on both sweeps. This improves performance significantly.

### 4.4 OPEX Algorithm

First, we compute all BCCs and articulation points as before. We also record, for each articulation point, the total size of each of its branches (i.e. the number of vertices and edges in each direction). Also initialize \( ub \) and \( lb \) as before.

1. If there are no articulation points with \( gap > 2T \), go to 4.

2. Let \( v \) = the “best” articulation point (see below).

3. Do a Double Sweep from \( v \), updating \( ub \) and \( lb \). Go to 1.

4. If there are no vertices with \( gap > 2T \), halt.

5. Do a Double Sweep from the vertex with the largest \( gap \), updating \( ub \) and \( lb \). Go to 4.

The “best” articulation point is the one that minimizes \( balance(v) = \sum_{v \in \text{branches}(v)} |V_v|(|V_v|+|E_v|) \). The branches of \( v \) are the subgraphs (components) that would exist if \( v \) were removed from \( G \).

Note that whenever we do the first sweep of a Double Sweep, we use Articulation Pruning to speed up the search. However, on the second sweep, we do not use Articulation Pruning because it interferes with our ability to update \( ub \) and \( lb \) effectively. Also note that in practice we find a Triple Sweep is usually slightly better than a Double Sweep, so our experimental results reflect this modification.

### 5. Experiments

We implemented OPEX and ran experiments to compare it to the baseline (Johnson’s) algorithm on a variety of datasets. We did not compare to HADI because it gives probabilistic approximations or the Iterating Bounds method because it does not compute eccentricities, only approximate diameter.

Table 1 describes the graphs we used in our experiments. NetSci is a collaboration network from the Network Sciences field. HEP-TH is a collaboration network from High-Energy Physics. Yahoo is a network of users sending instant messages over the Yahoo Messaging service. Twitter is a network of Twitter users mentioning each other.

Table 2 describes the time taken for each method to compute exact eccentricities. Note that OPEX is, in general, at least an order of magnitude faster than the baseline algorithm. For the Yahoo graph it is three orders of magnitude faster. We were unable to run the exact algorithms on the Twitter dataset due to computational resource availability.

Table 3 describes the time taken for each method to compute eccentricities with \( T = 1 \). For the Twitter dataset, we approximated the time for Baseline by sampling 1000 nodes and extrapolating to the full graph size.

### 6. Conclusions

Here we presented OPEX:

- OPEX is effective, capable of computing exact or approximate eccentricity, radius, and diameter for graphs.
- It is scalable, often performing orders of magnitude faster than its worst-case behavior in realistic graphs.

### 7. References


| Graph  | $|V|$ | $|E|$ | Radius | Diameter |
|--------|------|------|--------|----------|
| NetSci | 379  | 1.8K | 9      | 17       |
| HEP-TH | 5.8K | 27K  | 11     | 19       |
| Yahoo  | 27K  | 86K  | 35     | 69       |
| Twitter| 820K | 3M   | ~11    | ~20      |

Table 1. Summary of real-world networks used.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NetSci</th>
<th>HEP-TH</th>
<th>Yahoo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>$3.2 \times 10^{-1}$</td>
<td>128</td>
<td>3004</td>
</tr>
<tr>
<td>Art. Pruning</td>
<td>$1.0 \times 10^{-1}$</td>
<td>65</td>
<td>980</td>
</tr>
<tr>
<td>Updating Bounds</td>
<td>$3.3 \times 10^{-1}$</td>
<td>95</td>
<td>3834</td>
</tr>
<tr>
<td>UB + Double Sweep</td>
<td>$3.1 \times 10^{-1}$</td>
<td>59</td>
<td>13</td>
</tr>
<tr>
<td>OPEx</td>
<td>$8.0 \times 10^{-2}$</td>
<td>37</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Table 2. Timing in Seconds for Exact Eccentricity.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NetSci</th>
<th>HEP-TH</th>
<th>Yahoo</th>
<th>Twitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>$3.2 \times 10^{-1}$</td>
<td>128</td>
<td>3004</td>
<td>3.5M</td>
</tr>
<tr>
<td>Updating Bounds</td>
<td>$2.3 \times 10^{-1}$</td>
<td>16</td>
<td>3588</td>
<td>-</td>
</tr>
<tr>
<td>UB + Double Sweep</td>
<td>$2.1 \times 10^{-1}$</td>
<td>55</td>
<td>1.7</td>
<td>6886</td>
</tr>
<tr>
<td>OPEx</td>
<td>$4.0 \times 10^{-2}$</td>
<td>4</td>
<td>3.0</td>
<td>3342</td>
</tr>
</tbody>
</table>

Table 3. Timing in Seconds for Approximate Eccentricity at $T = 1$. 