# An implementation of the Pathfinder algorithm for sparse networks and its application on text networks

*Anže Vavpetič*
Faculty of Computer and Information Science,
*Vladimir Batagelj*
Faculty of Mathemathics and Physics, Department of Mathematics,
*Vid Podpečan*
Jožef Stefan Institute
Ljubljana

## Abstract

The Pathfinder algorithm is extensively used for pruning weighted networks. It is particularly useful in the analysis of co-citation networks. The present paper reviews two versions of the algorithm. A version with improved time and space complexities, called the Binary Pathfinder, and a version optimized for sparse networks, called Sparse Pathfinder. These two algorithms were implemented and tested on text networks. The obtained results show that the algorithm optimized for sparse networks works notably faster on such data.

## 1 Introduction

In larger (at least some hundreds of nodes) weighted networks the visual inspection can't be used anymore for identifying essential parts of the network. An approach to this problem are the pruning algorithms. They are used to remove less significant links, allowing the more salient links to be found. An example of a network pruning algorithm is the Pathfinder algorithm, developed in cognitive science to determine the most important links in a network [6]. The output defined by the Pathfinder algorithm is known as a Pathfinder network or PFnet. Initially, Pathfinder networks were used exclusively to represent relationships between concepts or keywords, but later works have extended its use to many other fields of application, for example co-citation networks.

### 1.1 Basic idea of the Pathfinder algorithm

Let $\mathcal{N} = (V, E, w)$ be a network. $V$ is the set of nodes, $E$ is the set of links, and $w : E \to \mathbb{R}_0^+$ the weight. We denote $n = |V|$ and $m = |E|$. Assuming that the weight represents a distance, the pruning idea of the Pathfinder algorithm is based on the triangle inequality, which states that the direct distance between two points must be less than or equal to the distance between those two points going through an intermediate point. It can be easily extended to all paths: the direct distance between two nodes must be less than or equal to the dist-length (sum of all weights) of every path of between these two nodes; therefore also less than or equal to the length of geodesic path (i.e. the shortest path). The algorithm eliminates the links which violate the extended triangle inequality.

The Pathfinder algorithm preserves the geodesic distance between every pair of nodes, while simplifiying the network and so clarifying it for the subsequent analysis [7].

The extended triangle inequality gives rise to the first parameter of the algorithm, i.e. the link-length – maximum number of intermediate links that will be considered, usually named $q$. Since the link-length of geodesic can not exceed $n-1$ the maximum possible value of $q$ is $n-1$.

To calculate the distance between two nodes along a path the Pathfinder algorithm uses the Minkowski operation

$$a \odot_r b = \sqrt[r]{a^r + b^r} \tag{1}$$

$r$ is the second parameter of the Pathfinder procedure. For different values of $r$ we get: $a \odot_1 b = a + b$; $a \odot_2 b = \sqrt{a^2 + b^2}$; $a \odot_\infty b = \max(a, b)$. The operation $\odot$ is associative. This means that for a path $\pi$ with links with weights $w_1, w_2, \ldots w_k$, we calculate its length as $d(\pi) = w_1 \odot w_2 \odot \ldots \odot w_k$.

## 2 Implementation

### 2.1 The Binary Pathfinder

The first version of the Pathfinder algorithm considered in this paper is the Binary Pathinder. The original Pathfinder has space complexity $O(qn^2)$ and time complexity $O(qn^3)$. The Binary Pathfinder [5] is an improvement to the original algorithm, reducing its time complexity to $O(n^3 \log q)$ and the space complexity to $O(n^2)$.

Before we describe the Binary Pathfinder, we list some definitions used by [5] and [6]:

- The Pathfinder network PFnet$(r, q) = (V, E, w)$ is a subnetwork of network $\mathcal{N}$.
- The weight of the link from node $u$ to node $v$ is denoted by $w_{uv}$. They are collected in a $n \times n$ matrix $W$.

- $W^{i+1} = W \odot W^i$ is computed as follows:

$$w_{uv}^{i+1} = \min\{w_{ut} \odot w_{tv}^i : t \in V\}$$

- The minimum-distance matrix for paths not exceeding $i$ links is denoted $D^i$ and its elements are computed as follows:

$$d_{uv}^i = \min(w_{uv}^1, w_{uv}^2, \ldots, w_{uv}^i) \text{ for } u \neq v$$

and $d_{vv}^i = 0$.

The crux of the improvements to the original Pathfinder made by [5] lies in the calculation of matrices $D^i$. They have pointed out that for determining the PFnet we only need the matrix $D^q$ for the comparison with the initial weight matrix. It is unnecessary to generate all of the $D^i$s. They have shown that $D^{i+j} = D^i \odot D^j$:

$$d_{uv}^{i+j} = \min\{d_{uv}^i, d_{uv}^j, w_{ut}^i \odot w_{tv}^j : t \in V\}$$

where $d_{uv}^1 = w_{uv}$. Thus we can make larger steps in computing the distance matrices. The Binary Pathfinder algorithm is based on matrices $D^1, D^2, D^4, D^8, \ldots$ [5]:

$i = 1$; $nq = 0$
Generate $D^1 = W$; $D^q = \infty$
If $(q \bmod 2) = 1$ :
    Compute $D^q = D^q \odot D^1$
    $nq = 1$
While $(2 * i) \leq q$ :
    Compute $D^{2i} = D^i \odot D^i$
    If $((q - nq) \bmod (4 * i)) > 0$ :
        Compute $D^q = D^q \odot D^{2i}$
        $nq = nq + (2 * i)$
    $i = 2 * i$
Comparing elements of $D^q$ and $W$, wherever $d_{uv} = w_{uv}$, add $(u, v)$ as a link to the PFnet.

The improvement in time complexity seems only minute, but as larger networks are dealt with, the difference grows immensely, as can be seen in the results section of [5].

## 2.2 The Pathfinder optimized for sparse networks

The second algorithm tested by this study is the Pathfinder procedure optimized for sparse networks. It is based on the idea presented in [1] by Vladimir Batagelj. The motivation for this algorithm is the fact that most real life networks are sparse and that in sparse networks the matrix $D^q$ can be computed faster using an adapted version of Dijkstra's algorithm [3] (for $q = n - 1$) or an adapted breadth-first search (BFS) algorithm (for $q < n-1$). To speed-up the algorithm we represent the network with a graph data structure using adjacency lists, where each node has a list of its neighbor nodes, resulting in a neighbor retrieval query with a time complexity of $O(1)$. To efficiently calculate the matrix $D^q$ for $q = n - 1$ we run the Dijkstra's algorithm based on Minkowski operation once for every node as a source node, thus producing a corresponding row of the resulting matrix. For $q < n-1$ the Dijkstra's algorithm can not be easily adapted. We replace it with an adapted shortest path algorithm based on BFS search. The last step of this algorithm is the same as in other implementations of the Pathfinder algorithm, i.e. comparing the elements of $D^q$ and $W$ and wherever $d_{uv} = w_{uv}$, we add $(u, v)$ as a link in the resulting PFnet.

### 2.2.1 Specifics

As mentioned, the Dijkstra's algorithm is modified so that in calculating dist-lengths, we use Minkowski's operation $\odot$ instead of the addition. With the BFS algorithm we (1) limit the search depth to $q$ and (2) skip all the paths starting from node $v$ that would yield a path length $d > \max\{w(v, u) : u \text{ is a successor of } v\}$.

**Sparse Pathfinder**$(r, q, \mathcal{N} =< V, E, w >)$ :
If $q \geq n - 1$: Dijkstra($r, q, \mathcal{N}$)
Else: BFS($r, q, \mathcal{N}$)
Comparing elements of $D^q$ and $W$, wherever $D^q[u, v] = w_{uv}$, add $(u, v)$ as a link to the PFnet.
**End Sparse Pathfinder**

**Dijkstra**$(r, q, \mathcal{N} =< V, E, w >)$ :
$pq := \{\}$ . . . priority queue sorted by distance
For each $v \in V$ :
    For each $u \in V$ : $dist[u] := \infty$; Mark $u$ as unvisited
    $dist[v] := 0$;
    Mark $v$ as visited and insert it into $pq$.
    While $pq \neq \{\}$ :
        $t := \min(pq)$; delete_min($pq$)
        For each successor $z$ of $t$:
            new_dist := $dist[t] \odot_r w_{tz}$
            If $z$ not visited:
                $dist[z]$ := new_dist
                Mark $z$ as visited and insert it into $pq$.
            Else if new_dist < $dist[z]$:
                decrease_key($pq$, $z$, new_dist)
    For each $u \in V$ : $D^q[v, u] := dist[u]$
**End Dijkstra**

**BFS**$(r, q, \mathcal{N} =< V, E, w >)$ :
$Q := \{\}$ . . . FIFO queue
For each $v \in V$ :
    For each $u \in V$ : $dist[u] := \infty$
    dMax := $\max\{w_{vu} : u \text{ is a successor of } v\}$
    putLast(Q,$v$,0,0)
    $dist[v] := 0$
    While $p \neq \{\}$ :
        $(u, d, l)$ := firstFrom(Q)
        $l := l + 1$
        For each neighbor $t$ of $u$:
            new_dist := $d \odot_r w_{ut}$
            If new_dist $\leq$ dMax and new_dist < dist[t]:
                dist[t] := new_dist
                If $l < q$ : putLast(Q,$t$,new_dist,$l$)

For each $u \in V : D^q[v,u] := dist[u]$

**End BFS**

In order to improve the efficiency of Dijkstra, the priority queue was implemented as a minimum binary heap with the following time complexities: $O(\log n)$ for insertion, deleting the minimum element, decreasing a key, and $O(1)$ for testing whether the priority queue is empty.

For such an implementation the time complexity of Dijkstra's algorithm is $O(m \log n + n \log n)$, which is dominated by $(m \log n)$, giving the total time complexity of the Sparse Pathfinder algorithm of $O(nm \log n)$. For dense networks, $m = O(n^2)$, we get the same time complexity as for the Binary Pathfinder. The space complexity remains the same as for the Binary Pathfinder, $O(n^2)$. In theory the complexity could be further improved to $O(nm)$ by using Fibonacci heap. But we considered the 'Bottom line' of [8, slide 26] which says: Fibonacci heap is best in theory, but not worth implementing.

The BFS algorithm makes a complete search of all possible paths originating in node $v$, of link-length at most $q$, and dist-length at most $dMax$. Its efficiency strongly depends on the properties of network (average degree, distribution of weights, parameter $r$) and it is very difficult to analyze analytically.

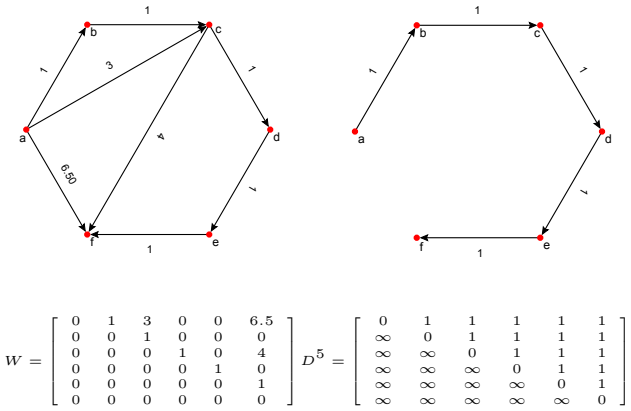


Figure 1: A graph before and after we apply the Pathfinder procedure $(r = \infty, q = 5)$ with the initial weight matrix $W$ and the target matrix $D^5$

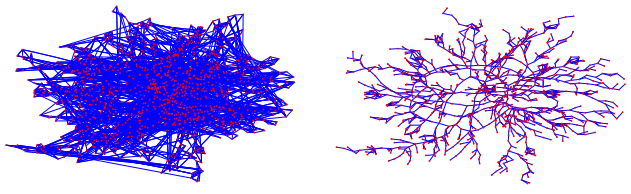

Figure 2: Network of terms generated from 100 documents and its PFnet$(r = \infty, q = n - 1)$

# 3 Experimental results

Both algorithms were implemented in C++ compiled with gcc 4.3.2 and integrated into the machine learning and data mining framework Orange [4] and can thus be easily imported as a Python module. The tests were done on a Intel Core 2 Duo 3GHz machine with 2GB of RAM running Ubuntu Linux. Input files were given in Pajek's format [2]; output networks were written back in the same format as well, and finally also drawn with Pajek.

The given input graphs can be interpreted as follows. Each link represents a co-occurence, which means that both words appear together in at least one document, whereas the link's weight represents the normalized number of co-occurences through all documents. This weight is a similarity measure. In order to transform the weights into dissimilarities required by Pathfinder, we applied the formula $w' = \frac{1}{w}$, where $w$ is the original weight.

First we present the comparison of the performance of both algorithms on the input graphs. The statistics gathered can be seen in Tables 1 and 2. On these networks, the optimized Pathfinder algorithm works much faster than the Binary Pathfinder. The reason for this is that $m$ is much smaller than $n^2$. In Table 1 the parameters $r$ and $q$ were set to $r = \infty$ and $q = n - 1$ as these values are most commonly used in practice; in Table 2 we provide some time measurements for smaller values of $q$, where the adapted BFS algorithm is used. We observe that the Sparse Pathfinder can be efficiently applied on such networks, as its computing time increases much more slowly than the Binary Pathfinder's, thus allowing to produce PFnets from larger networks in a reasonable time frame.

Figure 1 illustrates how the network changes when the Pathfinder procedure is applied. Figure 2 presents the effect of the Pathfinder on the network. On the left side the original network is presented, and on the right side the corresponding PFnet.

# 4 Conclusion

As noted by [5] the Pathfinder networks are of great interest in the study of different types of weighted networks. They are found to be particularly useful in scientometrics in studying advancing frontiers of research, disciplines, profiles of authors, etc.

Since the original algorithm has severe practical limitations, rising from its time and space complexity, we have implemented two improved versions: the Binary Pathfinder, a version presented in [5], and an algorithm optimized for sparse networks.

Both algorithms have been applied to several text networks, generated from a various number of documents in order to check the applicability to such networks. We have found that especially the algorithm for sparse networks has potential to be used for pruning of such networks, as its computational time rises much slower than the Binary Pathfinder's with the number of nodes and links, thus allowing to produce PFnets from

| **Input** (with $r = \infty$ and $q = n - 1$) | | | **Binary PF** | **Sparse PF** | **Output** |
|---|---|---|---|---|---|
| Network | $n$ | $m$ | $t(s)$ | $t(s)$ | $n$ |
| stem+cell_10docs.net | 64 | 127 | 0.187 | 0.006 | 71 |
| epilepsy+migraine_50docs.net | 517 | 1115 | 56.951 | 1.278 | 536 |
| stem+cell_100docs.net | 1215 | 2828 | 1035.302 | 17.606 | 1419 |
| epilepsy+migraine_100docs.net | 1322 | 3021 | 1183.712 | 21.145 | 1480 |
| migraine+protein_100docs.net | 1322 | 3021 | 1200.313 | 21.168 | 1480 |
| 2ksparse.net | 2622 | 5244 | 11069.679 | 75.938 | 2674 |
| 2kdense.net | 2622 | 39330 | 10913.843 | 390.118 | 3800 |
| 5ksparse.net | 5355 | 10710 | ≈97,000 (≈27h) | 593.982 | 5277 |
| 5kdense.net | 5355 | 80325 | ≈97,000 (≈27h) | 3425.378 | 6919 |

Table 1: Algorithm performance, $q = n - 1$ / Dijkstra

| Network | **Binary PF** $t(s)$ | | | **Sparse PF** $t(s)$ | | |
|---|---|---|---|---|---|---|
| $r = \infty$ | $q = 3$ | $q = 5$ | $q = 10$ | $q = 3$ | $q = 5$ | $q = 10$ |
| stem+cell_10docs.net | 0.064 | 0.088 | 0.103 | 0.001 | 0.002 | 0.006 |
| epilepsy+migraine_50docs.net | 18.841 | 27.486 | 33.784 | 0.052 | 0.136 | 1.524 |
| stem+cell_100docs.net | 217.067 | 299.909 | 366.942 | 0.410 | 0.988 | 25.963 |
| epilepsy+migraine_100docs.net | 288.932 | 376.673 | 491.588 | 0.369 | 1.269 | 21.753 |
| migraine+protein_100docs.net | 302.332 | 395.689 | 486.578 | 0.342 | 1.142 | 20.046 |
| 2ksparse.net | 1953.135 | 2512.451 | 3249.126 | 2.264 | 22.020 | 86.208 |

Table 2: Algorithm performance, small $q$ / BFS

larger inputs in reasonable time.

## Acknowledgments

The work presented in this paper was supported by the Slovenian Research Agency grant Knowledge Technologies, and by the grant of the European Commission under the 7th Framework Programme FP7-ICT-2007-C FET-Open, contract no. BISON-211898.

## References

[1] V. Batagelj. Fast pathfinder algorithm for large sparse networks (unpublished). Notes of the talk presented at the 1172-th Sredin seminar, Ljubljana, February 11, 2009.

[2] V. Batagelj and A. Mrvar. Pajek – analysis and visualization of large networks. In M. Jünger and P. Mutzel, editors, *Graph Drawing Software*. Springer, 2003.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 2001.

[4] J. Demšar, B. Zupan, and G. Leban. Orange: From experimental machine learning to interactive data mining. *White Paper*, 2004.

[5] V. P. Guerrero-Bote, F. Zapico-Alonso, M. E. Espinosa-Calvo, R. G. Crisóstomo, and F. de Moya-Anegón. Binary pathfinder: An improvement to the pathfinder algorithm. *Information Processing & Management*, 2006.

[6] R. W. Schvaneveldt. Pathfinder associative networks. *Norwood, NJ: Ablex*, 1990.

[7] R. W. Schvaneveldt, D. W. Dearholt, and F.T. Durso. Graph theoretic foundantions of pathfinder networks. *Computers and Mathematics with Applications*, 1988.

[8] R. Sedgewick and K. Wayne. Lectures 15: Shortest paths, 2009. `http://www.cs.princeton.edu/courses/archive/spr09/cos226/lectures/`.