# An implementation of the Pathfinder algorithm for sparse networks and its application on text networks (May 2010)

Anže Vavpetič, *Faculty of Computer and Information Science, Ljubljana,*
anze.vavpetic@gmail.com

***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.**

***Index Terms*— graph algorithms, network pruning, Pathfinder networks, visual inspection of networks**

## I. Introduction

NETWORK theory centres on the study of graphs associated with data, which are called networks. These networks represent entities and the relations between them in a formal way. Such networks are being applied in various fields: particle physics, computer science, biology, economics, sociology and so on. Usually researchers use different visualization techniques to gather new information from such networks, however, most networks of any value are very complex - they have a large number of nodes and links between them. In larger weighted networks (at least some hundreds of nodes) visual inspection cannot be used anymore for identifying essential parts of the network. An approach to this problem are network pruning algorithms. They are used to remove less significant links, allowing the more salient links to be found. Of course there is no universal solution to network pruning, since links that may not be important for a given structure may be so for another as stated in [5].

An example of a network pruning algorithm is the Pathfinder algorithm, developed in cognitive science to determine the most important links in a network [8]. 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.

Manuscript received May 23, 2010.

Anže Vavpetič is an undergraduate student at the Faculty of Computer and Information Science, Ljubljana, Slovenia (e-mail: anze.vavpetic@gmail.com).

### A. 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 $w : E \to \mathbb{R}_0^+$ is the weight. We denote $n = |V|$ and $m = |E|$. Assuming that a 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. The triangle inequality 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 between these two nodes; therefore also less than or equal to the length of the geodesic path (i.e. the shortest path). The algorithm eliminates the links which violate the extended triangle inequality.

A link that does not satisfy the triangle inequality will never be a part of the shortest path between two nodes, because there will always be a better alternative. Removing the links violating the inequality preserves the geodesic distances between nodes, thus simplifying the network and clarifying it for the subsequent analysis [9].

The extended triangle inequality gives rise to the first parameter of the algorithm, i.e. the *link-length* – which represents the maximum number of intermediate links that will be considered, usually denoted as $q$. From the fact that the link-length of the shortest path cannot exceed $n - 1$ follows that 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 $\odot$, defined as:

$$a \odot_r b = \sqrt[r]{a^r + b^r}. \quad (1)$$

From the definition of the Minkowski operation comes the second parameter (usually denoted $r$) of the Pathfinder procedure. For different values of $r$ we get:

- $a \odot_1 b = a + b$ (i.e. the road-map distance),
- $a \odot_2 b = \sqrt{a^2 + b^2}$ (i.e. the Euclidean distance),
- $a \odot_\infty b = \max(a, b)$.

An important property of the Minkowski operation $\odot$ is that it is *associative.* This means that for a path $\pi$ with links with weights $w_1, w_2, \ldots, w_k$, we calculate its dist-length as $d(\pi) = w_1 \odot w_2 \odot \ldots \odot w_k$.

### B. Related work

Apart from the improvement made by Guerrero-Bote, et al. [5], the Binary Pathfinder, which will be more thoroughly explained later in the paper, there have been other attempts to improve the original Pathfinder procedure. When $q \geq n-1$, the PFnet can be determined by the Fletcher's algorithm. This was proposed by Quirin et al [6] and it reduces the procedure's complexity from $O(qn^3)$ to $O(n^3)$. Another improvement was described for undirected networks in the case where $q \geq n-1$ and $r = \infty$. In this case the PFnet equals the union of all minimal spanning trees of the input network $\mathcal{N}$. Using an adapted version of Kruskal's minimal spanning tree algorithm the PFnet can be computed in $O(m \log n)$ time as described in Quirin, et al [7].

## II. Implementation

### A. The Binary Pathfinder

The first version of the Pathfinder algorithm considered in this paper is the Binary Pathfinder. The Binary Pathfinder [5] is an improvement to the original algorithm, improving the procedure's time complexity, regardless of the input network, from $O(qn^3)$ to $O(n^3 \log q)$ and the space complexity to $O(n^2)$.

The links violating the triangle inequality will be removed, since they have an associated distance which is greater than some other path between the same two points consisting of up to $q$ links, and with the overall distance of this second path calculated with the Minkowski operation $\odot$ with the parameter $r$. Before we describe the Binary Pathfinder, we list some definitions used by [5] and [8]:

- 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 $w_{uv}$. The weights are collected in a $n \times n$ matrix $W$.
- Let $W^i$ be a $n \times n$ matrix with elements $w_{uv}^i$; $w_{uv}^i$ is the minimum dist-length between nodes $u$ and $v$ going through *exactly* $i$ links.
  $W^{i+1} = W \odot W^i$ 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:

$$d_{uv}^i = \min(w_{uv}^1, w_{uv}^2, \dots, w_{uv}^i) \text{ for } u \neq v \text{ and } d_{vv}^i = 0.$$

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 matrices $D^i$, $i = 1,2,\dots,q$. They have shown the following fact: a minimum-distance matrix for paths not exceeding $i+j$ links, $D^{i+j}$, can be calculated directly from matrices $D^i$ and $D^j$; we can define the relation $D^{i+j} = D^i \odot D^j$ with:

$$d_{uv}^{i+j} = \min\{d_{uv}^i, d_{uv}^j, d_{ut}^i \odot d_{tv}^j : t \in V\} \quad (2)$$

where $d_{uv}^1 = w_{uv}$. The formula can be interpreted as follows: we know that all sub-paths into which a minimum distance path of up to $i$ links can be decomposed will also be minimal - at least among the paths with fewer than $i$ links. This means that any minimum distance length considered in $D^{i+j}$ must be already considered in $D^i$ or $D^j$, or a combination of two minimum distance lengths – one from $D^i$ and one from $D^j$. Let $k$ be the length of any minimum path considered (its dist-length) in $D^{i+j}$. The following three cases apply:

- if $k \leq i$, then the path has to be considered in $D^i$,
- if $k \leq j$, then the path has to be considered in $D^j$,
- otherwise, as $k \leq i+j$, the path can always be decomposed into one of length $i$ which has to be considered in $D^i$ and another of length $k-i$ which has to be considered in $D^j$, because, as mentioned before, any sub-path of a path considered minimal for $D^{i+j}$ has to be optimal among the paths with fewer than $i+j$ links, and therefore of up to $j$ links.

Instead of calculating all the matrices $D^1, D^2, D^3, \dots, D^q$ (i.e. doing steps of length 1) the Binary Pathfinder calculates only the matrices $D^1, D^2, D^4, D^8, \dots, D^q$ [5] by the use of formula (2) and so calculating only $\log q$ instead of $q$ matrices (e.g. to compute $D^{100}$ we need to compute $\lceil \log_2 100 \rceil = 7$ $D^1, D^2, D^4, D^8, D^{16}, D^{32}, D^{64}$ and $D^{128}$ instead of 100 matrices):

**BinaryPathfinder**$(r, q, \mathcal{N} =< V, E, w >)$:
$i := 1$
$nq := 0$
Generate $D^1 := W$
$D^q := \infty$
If $(q \ mod \ 2) = 1$:
  Compute $D^q := D^q \odot_r D^1$
  $nq := 1$
While $2i \leq q$:
  Compute $D^{2i} := D^i \odot_r D^i$
  If $\big((q - nq) \ mod \ (4i)\big) > 0$:
    Compute $D^q := D^q \odot_r D^{2i}$
    $nq := nq + 2i$
  $i := 2i$
Compare the elements of $D^q$ and $W$, wherever $d_{uv} = w_{uv}$ add $(u,v)$ as a link to the PFnet
**End BinaryPathfinder**

According to the algorithm, in order to compute the resulting PFnet, one has to compute $\log q$ (this follows from the number of *while* loop iterations) matrices $D^i$. For each matrix we need to calculate $n^2$ elements, and for each element we need $n$ comparisons; therefore the total time complexity of the algorithm is $O(n^3 \log q)$.

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].

### *B. The Pathfinder procedure 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 (where usually $m \ll n^2$) 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 $q < n-1$).

The idea behind the algorithm is as follows. As we generate the distance matrices in the Binary Pathfinder algorithm, we must calculate $n^2$ weights for every matrix and thus making $n^3$ comparisons for each matrix, regardless of the number of links in the network. We can speed up the algorithm by representing 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 the Minkowski operation once for every node as a source node, thus producing a corresponding row of the resulting matrix. For example, if we run Dijkstra's algorithm for node $i$, we actually produce the $i$-th row of the target matrix $D^q$. For $q < n-1$ the Dijsktra's algorithm cannot 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.

#### *1) Specifics*

As mentioned, the Dijkstra's algorithm is modified so that in calculating dist-lengths, we use Minkowski's operation $\odot_r$ instead of addition and 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 $d > \max\{w(v,u) : u \text{ is a successor of } v\}$.

The second part of the BFS adaptation is a simple method to prune some of the search graph in order to improve the performance of the algorithm.

Before we list the algorithm's pseudo-code we should first describe some of the quantities and procedures used in it:

- $dist[w]$ is the (current) distance from the source node $v$ to node $w$,
- $len[w]$ is the (current) link-length of the path from the source node $v$ to node $w$,
- $\min(pq)$ is the minimum element in the priority queue,
- delete_min$(pq)$ removes the minimum element from the priority queue,
- decrease_key$(pq, w, d)$ decreases the key (note that the ordering key is the distance) of $w$ to $d$. Note that after decrease_key executes, $dist[w] = d$ should be true.

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

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

**BFS**$(r, q, \mathcal{N} =< V, E, w >)$:
Initialize a FIFO queue $Q \coloneqq \emptyset$
For each $v \in V$:
  For each $u \in V$:
    $dist[u] \coloneqq \infty$
  $dMax \coloneqq \max\{w_{vu} : u \text{ is a successor of } v\}$
  putLast$(Q, v, 0, 0)$
  $dist[v] \coloneqq 0$
  While $Q \neq \emptyset$:
    $(u, d, l) \coloneqq \text{firstFrom}(Q)$
    $l \coloneqq l + 1$
    For each neighbor $t$ of $u$:
      $new\_dist \coloneqq d \odot_r w_{ut}$
      If $new\_dist \leq dMax$ and $new\_dist < dist[t]$:
        $dist[t] \coloneqq new\_dist$
        If $l < q$:
          putLast$(Q, t, new_dist, l)$
  For each $u \in V$:
    $D^q[v, u] \coloneqq dist[u]$
**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,
- $O(\log n)$ for deleting the minimum element,
- $O(\log n)$ for 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 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 a Fibonacci heap. But we considered the 'Bottom line' of [10, 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 a network (average degree, distribution of weights, parameter $r$) and it is very difficult to analyze analytically.

## III. Experimental results

Both algorithms were implemented in C++ and integrated into the machine learning and data mining framework Orange [4] and can thus be simply used as a Python module.

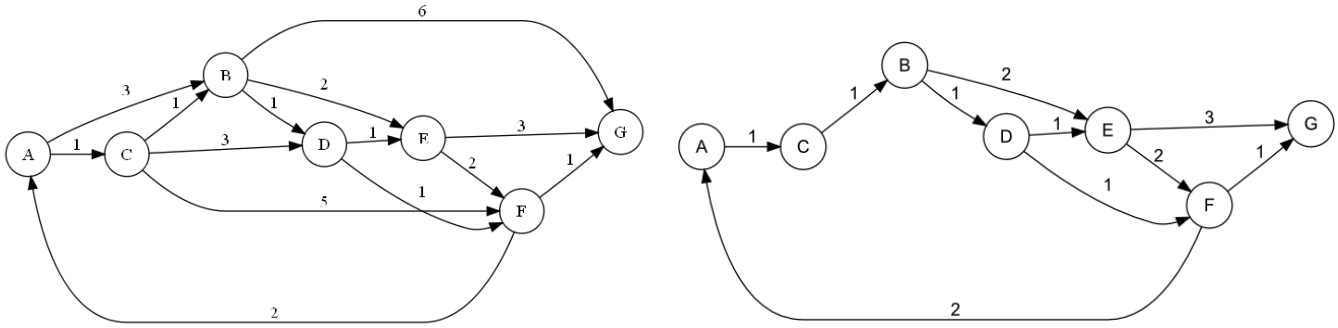
The tests were done on an Intel Core 2 Duo 2GHz with 2GB of RAM running Windows 7 and compiled with Visual Studio 2008.

Input networks were represented in Pajek's format [2]; output networks were written back in the same format as well, and finally also visualized with Pajek.

We mentioned that we applied the Pathfinder to text networks. These networks were created by analyzing various (medical) documents and can be interpreted as follows. Each link between two words represents a co-occurence, meaning 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 (e.g. if two words appear together in 77 of 100 documents the resulting weight on the link connecting the two words would be 0.77). This weight is a similarity measure in the sense that two words (or concepts) linked with a higher weight are more similar (i.e. are more "connected" because they appear together more often) than two words with a lower weight. In order to transform the weights into dissimilarities required by the 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 networks. The statistics gathered can be seen in Tables 1 and 2. On these networks, the Sparse Pathfinder works much faster than the Binary Pathfinder. 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 $q$, i.e. for $q \in \{3, 5, 10\}$, where the adapted BFS algorithm is used. It is evident from the collected data that even when the BFS is used (i.e. when $q < n - 1$), the Sparse Pathfinder performs much better than the Binary Pathfinder. Figure 3 gives a graphical representation of the gathered performance measurements.

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.




$$W = \begin{bmatrix} 0 & 3 & \boxed{1} & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \boxed{1} & \boxed{2} & \infty & 6 \\ \infty & \boxed{1} & 0 & 3 & \infty & 5 & \infty \\ \infty & \infty & \infty & 0 & \boxed{1} & \boxed{1} & \infty \\ \infty & \infty & \infty & \infty & 0 & \boxed{2} & \boxed{3} \\ \boxed{2} & \infty & \infty & \infty & \infty & 0 & \boxed{1} \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

$$D^6 = \begin{bmatrix} 0 & 2 & \boxed{1} & 3 & 4 & 4 & 5 \\ 4 & 0 & 5 & \boxed{1} & \boxed{2} & 2 & 3 \\ 5 & \boxed{1} & 0 & 2 & 3 & 3 & 4 \\ 3 & 5 & 4 & 0 & \boxed{1} & \boxed{1} & 2 \\ 4 & 6 & 5 & 7 & 0 & \boxed{2} & \boxed{3} \\ \boxed{2} & 4 & 3 & 5 & 6 & 0 & \boxed{1} \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

Fig. 1. A small example of a PFnet($\boldsymbol{r = 1, q = 6}$) (right) calculation from the original network (with $\boldsymbol{n = 7, m = 14}$) (left) with the corresponding matrices. The numbers in boxes represent the corresponding weights and dist-lengths that are equal in the two matrices and are thus added into the PFnet.

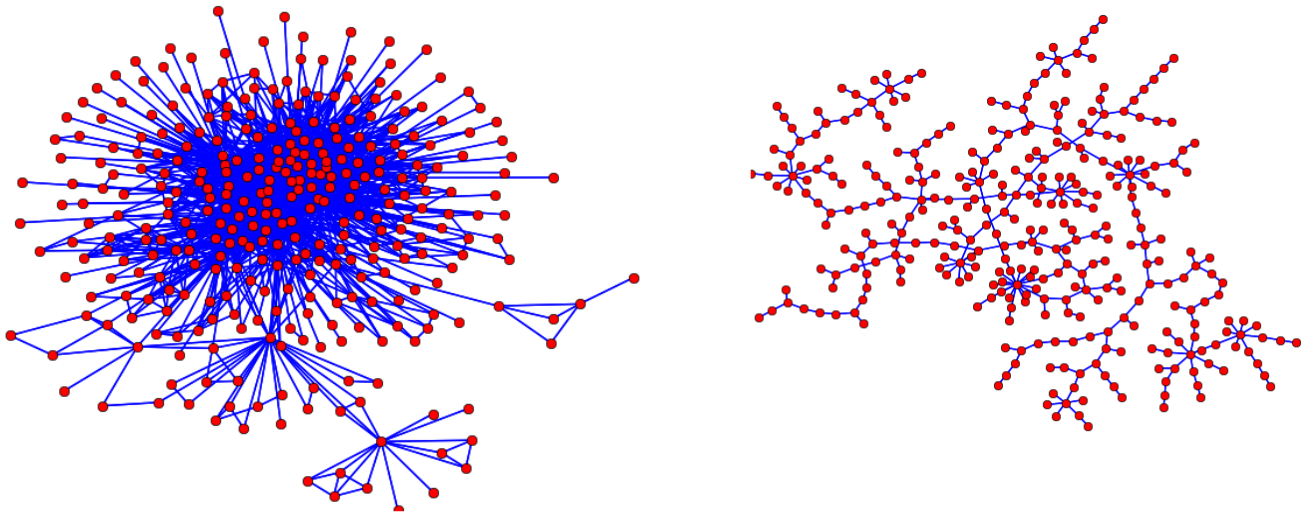


Fig. 2. Effect of the Pathfinder algorithm (right) on the original network (left) with 300 nodes

Figure 1 illustrates how the network changes when the Pathfinder procedure is applied. The original network is given by its weight matrix $W$. In order to compute the resulting PFnet($r = 1, q = 6$) presented on the right side of the figure, we need the minimum distance between each pair of the nodes of up to 6 links (since $q = 6$), thus we need to compute the matrix $D^6$. To get the resulting PFnet, we keep only the edges $(u, v)$, where $w_{uv} = d^6_{uv}$, i.e. the weight of the link between those two nodes is equal to the minimum distance (of up to 6 links) between them.

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. This figure also illustrates that by removing unimportant links, it makes the network easier to inspect and it can also result in a better visualization of the network.

The interpretation of the resulting Pathfinder networks is another matter by itself and seems to be more difficult that in the case of social networks, we thus leave this matter for a another paper.

## IV. 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 [1].

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 the algorithm for sparse networks in particular has potential to be used for the pruning of such networks, as its computational time rises much slower with the number of nodes and links than the Binary Pathfinder's, thus allowing to produce PFnets from larger inputs in reasonable time.

TABLE I
Algorithm performance, $q = n - 1$ / Dijkstra

| Input (with $r = \infty$ and $q = n - 1$) | | | Binary PF | Sparse PF | Output |
|---|---|---|---|---|---|
| Network $\mathcal{N}$ | $n$ | $m$ | $t[s]$ | $t[s]$ | $m$ |
| stem+cell_10docs.net | 64 | 123 | 0.291 | 0.004 | 88 |
| epilepsy+migraine_50docs.net | 517 | 1115 | 75.734 | 1.024 | 622 |
| stem+cell_100docs.net | 1215 | 2828 | 1375.476 | 15.719 | 1763 |
| epilepsy+migraine 100docs.net | 1322 | 3021 | 1730.017 | 18.236 | 1779 |
| migraine+protein_100docs.net | 1322 | 3021 | 2024.116 | 18.011 | 1779 |
| 2ksparse.net | 2622 | 4233 | 13933.394 | 57.027 | 2647 |
| 2kdense.net | 2622 | 29884 | 13733.810 | 425.455 | 2822 |
| 5ksparse.net | 5355 | 9241 | ~ 127000 (about 35 h) | 424.215 | 5292 |
| 5kdense.net | 5355 | 64896 | ~ 127000 (about 35 h) | 4464.224 | 5524 |

TABLE II
ALGORITHM PERFORMANCE, SMALL $q$ / BFS

| 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 | 0.085 | 0.113 | 0.140 | 0.001 | 0.002 | 0.003 |
| epilepsy+migraine_50docs.net | 21.121 | 28.610 | 34.927 | 0.041 | 0.097 | 0.655 |
| stem+cell_100docs.net | 234.363 | 313.640 | 391.906 | 0.261 | 0.670 | 6.343 |
| epilepsy+migraine 100docs.net | 413.890 | 531.624 | 641.495 | 0.297 | 0.772 | 7.940 |
| migraine+protein_100docs.net | 400.810 | 457.918 | 636.951 | 0.296 | 0.782 | 7.948 |

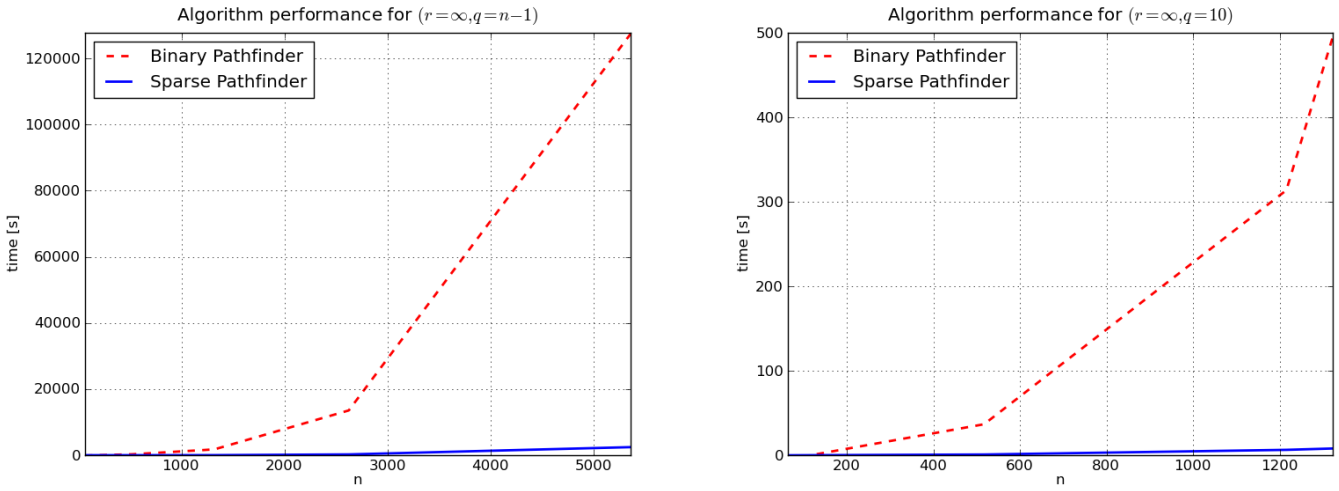



Fig. 3. Graphical representation of the algorithm performance measurements.


## ACKNOWLEDGEMENTS

I wish to thank professor Dr Vladimir Batagelj of the Faculty of Mathemathics and Physics, Department of Mathematics, University of Ljubljana, for the notes about the Sparse Pathfinder. I would also like to thank Mr Vid Podpečan of the Jožef Stefan Institute, Department of Knowledge Technologies, Ljubljana, for the mentorship and general help during the writing of this paper and Mr Matjaž Juršič, also of the Jožef Stefan Institute, Department of Knowledge Technologies, Ljubljana, for his insight into text networks - to both of them also a thanks for providing me with text networks to be used in the experiment.


## 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 nad 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] A. Quirin, O. Cordón, J. Santamaria, B. Vargas-Quesada, F. Moya-Anegón, "A new variant of the Pathfinder algorithm to generate large visual science maps in cubic time". Available: http://www.scimago.es/benjamin/A_new_variant_of_the_Pathfinder_Algorithm.pdf

[7] Arnaud Quirin, Oscar Cordón, Vicente P. Guerrero-Bote, Benjamín Vargas-Quesada and Felix Moya-Anegón, "A Quick MST-Based Algorithm to Obtain Pathfinder Networks (∞, n − 1)", *Journal of the American Society for Information Science and Technology*, Volume 59, Issue 12 (p 1912-1924). Available: http://www3.interscience.wiley.com/cgi-bin/fulltext/120736756/PDFSTART

[8] R. W. Schvaneveldt, Pathfinder associative networks, Norwood, NJ:Ables, 1990.

[9] R. W. Schvaneveldt, D. W. Dearholt, and F. T. Durso, "Graph theoretic foundations of pathfinder networks", Computers and Mathematics with Applications, 1988.

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