Search Result Clustering via Randomized Partitioning of Query-Induced Subgraphs

Aleksandar Bradić

Faculty of Electrical Engineering, Belgrade
abradic@acm.org

November, 25. 2008.
16th Telecommunications forum, TELFOR 2008

Contents

Contents

Search result clustering

1. Problem statement: efficient representation of result set in modern search engines
2. Score-based model is effective in the case of search for the *best* document corresponding to given query
3. However, it's insufficient in situations which require representation of broad set of results (exploratory search)
4. Clustering Search focuses not only on display of *relevance* but the way documents are interrelated and their organizations into **clusters**

Search result clustering

1. Problem statement: efficient representation of result set in modern search engines
2. Score-based model is effective in the case of search for the *best* document corresponding to given query
3. However, it's insufficient in situations which require representation of broad set of results (exploratory search)
4. Clustering Search focuses not only on display of *relevance* but the way documents are interrelated and their organizations into **clusters**

Search result clustering

1. Problem statement: efficient representation of result set in modern search engines
2. Score-based model is effective in the case of search for the *best* document corresponding to given query
3. However, it's insufficient in situations which require representation of broad set of results (exploratory search)
4. Clustering Search focuses not only on display of *relevance* but the way documents are interrelated and their organizations into **clusters**

Search result clustering

1. Problem statement: efficient representation of result set in modern search engines
2. Score-based model is effective in the case of search for the *best* document corresponding to given query
3. However, it's insufficient in situations which require representation of broad set of results (exploratory search)
4. Clustering Search focuses not only on display of *relevance* but the way documents are interrelated and their organizations into **clusters**

Contents

Information-retrieval methods

1. Clustering based on information content has been a well studied topic in *Information Retrieval* (**IR**)
2. Standard IR methods are based on the *clustering hypothesis*, stating that the relevant documents thend to be more similar to each other than to non-relevant documents
3. Such methods operate by calculating appropriate content-based relevance values and imposing *similarity* metric, used for clustering (*K-means clustering* is a example of commonly used distance-based algorithm).
4. These methods have been accepted by Search Engine community and implemented in a number of real-world search engines (*Vivisimo, Carrot Clustering Search Engine, Clusty...*)

Information-retrieval methods

1. Clustering based on information content has been a well studied topic in *Information Retrieval* (**IR**)
2. Standard IR methods are based on the *clustering hypothesis*, stating that the relevant documents thend to be more similar to each other than to non-relevant documents
3. Such methods operate by calculating appropriate content-based relevance values and imposing *similarity* metric, used for clustering (*K-means clustering* is a example of commonly used distance-based algorithm).
4. These methods have been accepted by Search Engine community and implemented in a number of real-world search engines (*Vivisimo, Carrot Clustering Search Engine, Clusty...*)

Information-retrieval methods

1. Clustering based on information content has been a well studied topic in *Information Retrieval* (**IR**)
2. Standard IR methods are based on the *clustering hypothesis*, stating that the relevant documents thend to be more similar to each other than to non-relevant documents
3. Such methods operate by calculating appropriate content-based relevance values and imposing *similarity* metric, used for clustering (*K-means clustering* is a example of commonly used distance-based algorithm).
4. These methods have been accepted by Search Engine community and implemented in a number of real-world search engines (*Vivisimo, Carrot Clustering Search Engine, Clusty...*)

Information-retrieval methods

1. Clustering based on information content has been a well studied topic in *Information Retrieval* (**IR**)
2. Standard IR methods are based on the *clustering hypothesis*, stating that the relevant documents thend to be more similar to each other than to non-relevant documents
3. Such methods operate by calculating appropriate content-based relevance values and imposing *similarity* metric, used for clustering (*K-means clustering* is a example of commonly used distance-based algorithm).
4. These methods have been accepted by Search Engine community and implemented in a number of real-world search engines (*Vivisimo, Carrot Clustering Search Engine, Clusty...*)

Graph-theoretic methods

1. However, IR-based clustering fails to capture the *hyperlink* component of the Web data - reflected in the *link graph*
2. This structure describes the explicit way in which the documents are related
3. A lot of algorithms utilize this structure to extract information about document relevance (*PageRank*) and community structure (*HITS*)
4. Great succes of these algorithms indicated the significance of link structure in Web data analysis and suggested extension of such problem to related problems like *community detection* and *Web data clustering*

Graph-theoretic methods

1. However, IR-based clustering fails to capture the *hyperlink* component of the Web data - reflected in the *link graph*
2. This structure describes the explicit way in which the documents are related
3. A lot of algorithms utilize this structure to extract information about document relevance (*PageRank*) and community structure (*HITS*)
4. Great succes of these algorithms indicated the significance of link structure in Web data analysis and suggested extension of such problem to related problems like *community detection* and *Web data clustering*

Graph-theoretic methods

1. However, IR-based clustering fails to capture the *hyperlink* component of the Web data - reflected in the *link graph*
2. This structure describes the explicit way in which the documents are related
3. A lot of algorithms utilize this structure to extract information about document relevance (*PageRank*) and community structure (*HITS*)
4. Great succes of these algorithms indicated the significance of link structure in Web data analysis and suggested extension of such problem to related problems like *community detection* and *Web data clustering*

Graph-theoretic methods

1. However, IR-based clustering fails to capture the *hyperlink* component of the Web data - reflected in the *link graph*
2. This structure describes the explicit way in which the documents are related
3. A lot of algorithms utilize this structure to extract information about document relevance (*PageRank*) and community structure (*HITS*)
4. Great succes of these algorithms indicated the significance of link structure in Web data analysis and suggested extension of such problem to related problems like *community detection* and *Web data clustering*

Search result clustering using graph-theoretic methods

1. Idea : apply the link-analysis to the search result clustering, by reducing the problem to problem of *clustering* of the underlying hyperlink graph (*graph clustering* problem)
2. Implementing this in practice is **hard** - primary due to the fact that unlike IR-based methods which operate on set of values precomputed for each document, graph-based algorithms operate on dynamical query-dependent representation of entire link graph
3. This makes precomputation impossible and problem both computationaly and space-intensive
4. Currently there are no real-world clustering engines that implement search result clustering using the link-graph approach

Search result clustering using graph-theoretic methods

1. Idea : apply the link-analysis to the search result clustering, by reducing the problem to problem of *clustering* of the underlying hyperlink graph (*graph clustering* problem)
2. Implementing this in practice is **hard** - primary due to the fact that unlike IR-based methods which operate on set of values precomputed for each document, graph-based algorithms operate on dynamical query-dependent representation of entire link graph
3. This makes precomputation impossible and problem both computationaly and space-intensive
4. Currently there are no real-world clustering engines that implement search result clustering using the link-graph approach

Search result clustering using graph-theoretic methods

1. Idea : apply the link-analysis to the search result clustering, by reducing the problem to problem of *clustering* of the underlying hyperlink graph (*graph clustering* problem)
2. Implementing this in practice is **hard** - primary due to the fact that unlike IR-based methods which operate on set of values precomputed for each document, graph-based algorithms operate on dynamical query-dependent representation of entire link graph
3. This makes precomputation impossible and problem both computationaly and space-intensive
4. Currently there are no real-world clustering engines that implement search result clustering using the link-graph approach

Search result clustering using graph-theoretic methods

1. Idea : apply the link-analysis to the search result clustering, by reducing the problem to problem of *clustering* of the underlying hyperlink graph (*graph clustering* problem)
2. Implementing this in practice is **hard** - primary due to the fact that unlike IR-based methods which operate on set of values precomputed for each document, graph-based algorithms operate on dynamical query-dependent representation of entire link graph
3. This makes precomputation impossible and problem both computationaly and space-intensive
4. Currently there are no real-world clustering engines that implement search result clustering using the link-graph approach

Contents

Outline of the paper

1. Proposal of relaxation of the problem of the search result clustering from the problem of clustering the entire graph to the domain of *query-induced subgraph* - representing a subgraph generated by given search query
2. A validity of such proposal is shown by determining that the essential structural properties of entire graph are still preserved in given subgraph
3. Introducing a novel algorithm for *approximate* clustering of such subgraphs - enabling space and computationally efficient clustering with variable error, suitable for practical implementations
4. Practical imlementation of proposed concepts in real-world clustering search engine

Outline of the paper

1. Proposal of relaxation of the problem of the search result clustering from the problem of clustering the entire graph to the domain of *query-induced subgraph* - representing a subgraph generated by given search query
2. A validity of such proposal is shown by determining that the essential structural properties of entire graph are still preserved in given subgraph
3. Introducing a novel algorithm for *approximate* clustering of such subgraphs - enabling space and computationally efficient clustering with variable error, suitable for practical implementations
4. Practical imlementation of proposed concepts in real-world clustering search engine

Outline of the paper

1. Proposal of relaxation of the problem of the search result clustering from the problem of clustering the entire graph to the domain of *query-induced subgraph* - representing a subgraph generated by given search query
2. A validity of such proposal is shown by determining that the essential structural properties of entire graph are still preserved in given subgraph
3. Introducing a novel algorithm for *approximate* clustering of such subgraphs - enabling space and computationally efficient clustering with variable error, suitable for practical implementations
4. Practical imlementation of proposed concepts in real-world clustering search engine

Outline of the paper

1. Proposal of relaxation of the problem of the search result clustering from the problem of clustering the entire graph to the domain of *query-induced subgraph* - representing a subgraph generated by given search query
2. A validity of such proposal is shown by determining that the essential structural properties of entire graph are still preserved in given subgraph
3. Introducing a novel algorithm for *approximate* clustering of such subgraphs - enabling space and computationally efficient clustering with variable error, suitable for practical implementations
4. Practical imlementation of proposed concepts in real-world clustering search engine

Contents

Definition

Query-Induced Subgraph:

Let the hyperlink graph be a graph $G = (V, E)$, where V is a set of vertices representing all the documents in the search engine index, and a set E of edges which represents the hyperlinks between all the documents.
We define **Query-Induced Subgraph** *as a graph $G_q = (V_q, E_q)$, where $V_q \subset V$ is a set of all results matching the given query q and $E_q \subset E$ set of all edges among vertices from the set V_i. ..*

In practice - given subgraph ($G_q \subset G$) represents the hyperlink graph created from G by keeping only the documents matching the query and hyperlinks amongst the result set of documents

Definition

1. We define *node degree* as sum of total number of inlinks and outlinks for each node, and treat it as a measure of information content contained in link data.
2. In [5], authors perform the first analysis of the general structure of the Web, and determine that node degree distribution follows a simple power-law of the form $k^{-\theta}$, with $\theta = 2.1$ for in-degree and $\theta = 2.7$, for out-degree. In [6], a single subset of Web Graph is analyzed - the Web of a single country (*Web of Spain*) and similar distribution is observed, with $\theta = 2.11$ for in-degree and $\theta = 2.84$ for out-degree.

Definition

1. We define *node degree* as sum of total number of inlinks and outlinks for each node, and treat it as a measure of information content contained in link data.
2. In [5], authors perform the first analysis of the general structure of the Web, and determine that node degree distribution follows a simple power-law of the form $k^{-\theta}$, with $\theta = 2.1$ for in-degree and $\theta = 2.7$, for out-degree. In [6], a single subset of Web Graph is analyzed - the Web of a single country (*Web of Spain*) and similar distribution is observed, with $\theta = 2.11$ for in-degree and $\theta = 2.84$ for out-degree.

Definition

1. Given result validating the scale-free structure of the Web Graph and indicates that the link distribution is invariant to the change of scale
2. We analyze whether the scale-free property holds as well, in the case of *query-induced subgraphs*
3. Our goal is to show that the *node degree* in the given *query-induced subgraph*, preserves the same distribution as in the entire graph (anticipated by the general assumption about *scale-free* structure of Web and social networks).
4. Such result could validate the reduction of a general graph clustering problem to the domain of *query-induced subgraph*

Definition

1. Given result validating the scale-free structure of the Web Graph and indicates that the link distribution is invariant to the change of scale
2. We analyze whether the scale-free property holds as well, in the case of *query-induced subgraphs*
3. Our goal is to show that the *node degree* in the given *query-induced subgraph*, preserves the same distribution as in the entire graph (anticipated by the general assumption about *scale-free* structure of Web and social networks).
4. Such result could validate the reduction of a general graph clustering problem to the domain of *query-induced subgraph*

Definition

1. Given result validating the scale-free structure of the Web Graph and indicates that the link distribution is invariant to the change of scale
2. We analyze whether the scale-free property holds as well, in the case of *query-induced subgraphs*
3. Our goal is to show that the *node degree* in the given *query-induced subgraph*, preserves the same distribution as in the entire graph (anticipated by the general assumption about *scale-free* structure of Web and social networks).
4. Such result could validate the reduction of a general graph clustering problem to the domain of *query-induced subgraph*

Definition

1. Given result validating the scale-free structure of the Web Graph and indicates that the link distribution is invariant to the change of scale
2. We analyze whether the scale-free property holds as well, in the case of *query-induced subgraphs*
3. Our goal is to show that the *node degree* in the given *query-induced subgraph*, preserves the same distribution as in the entire graph (anticipated by the general assumption about *scale-free* structure of Web and social networks).
4. Such result could validate the reduction of a general graph clustering problem to the domain of *query-induced subgraph*

Contents

Properties

1. In order to validate the given assumption about degree distributions, we analyze the dataset obtained as a part of *randomNode* clustering engine.
2. Given dataset consists of data about 1.1 million nodes (representing the subset of *.yu* Web), generated by calculating inlink degrees for resulting sets of 1000 top-frequency queries in *randomNode* clustering engine.

Properties

1. In order to validate the given assumption about degree distributions, we analyze the dataset obtained as a part of *randomNode* clustering engine.
2. Given dataset consists of data about 1.1 million nodes (representing the subset of *.yu* Web), generated by calculating inlink degrees for resulting sets of 1000 top-frequency queries in *randomNode* clustering engine.

Properties

1. We analyze the distribution of inlink degrees for both full graph and induced subgraphs obtained for each of given queries and test the hypothesis that both graphs have distribution, commonly found in Internet and social networks [7] - a power law distribution with β and x_{min} parameters, and density function of the form :

$$p(x; \beta, x_{min}) = \frac{x^{-\beta}}{\zeta(\beta, x_{min})} \tag{1}$$

where $\zeta(\beta, x_{min})$, represents the generalized zeta function $\zeta(\beta, x_{min}) = \sum_{n=0}^{\infty}(n + x_{min})^{-\beta}$.

Properties

1. We use the method of Maximum Likelihood (ML) for estimation of distribution parameters, as described in [8]. The approximate expression for MLE estimator of β parameter is given by :

$$\hat{\beta} \equiv 1 + n\left[\sum_{i=1}^{\infty} \ln \frac{x_i}{x_{min} - \frac{1}{2}}\right]^{-1} \tag{2}$$

where x_{min}, represents the lower bound on the power law behavior.

Contents

Analysis

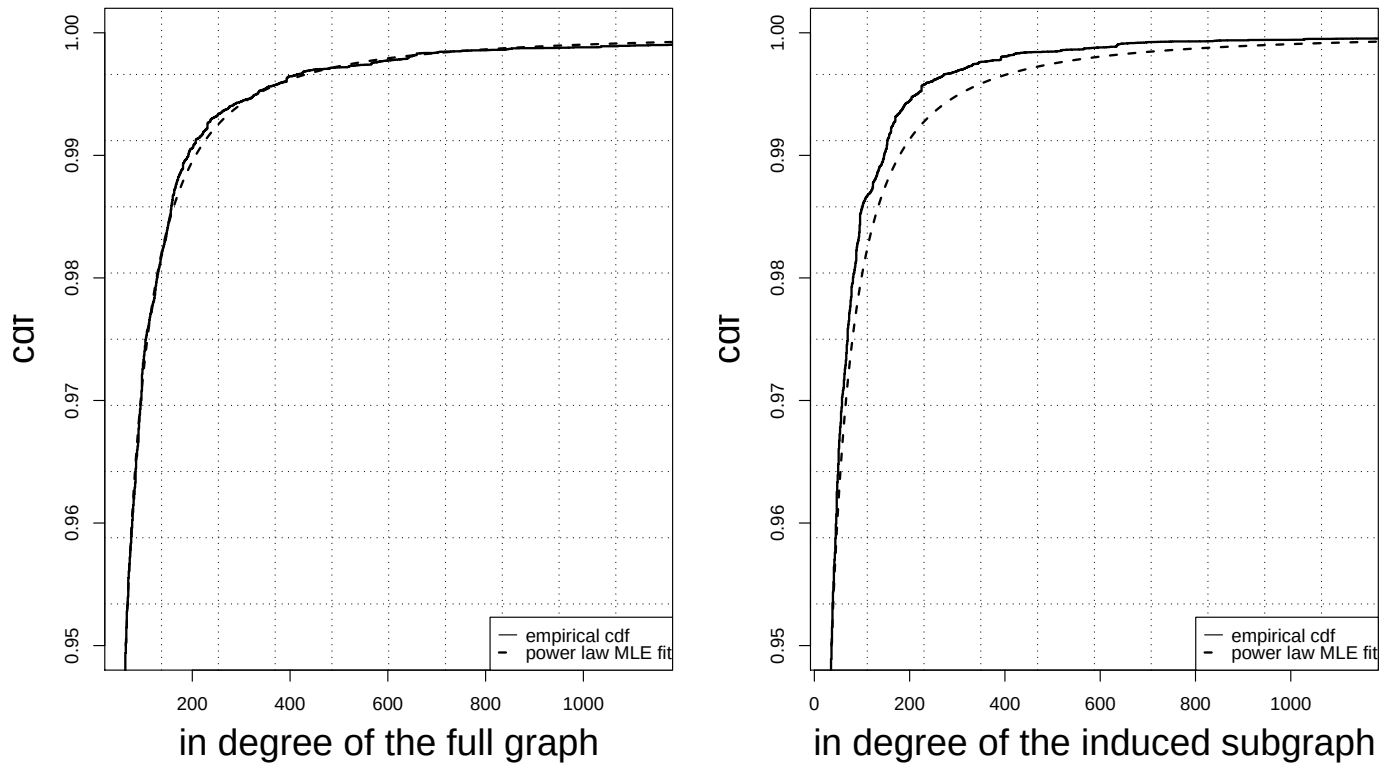

Figure I : cummulative distribution function (cdf) of node degrees in entire graph (full line) and in query-induced subgraphs (dotted line), obtained from the given dataset, vs the cdf of fitted power law distribution

Analysis

Tabela: Link Distribution Power Law Fit

	median	mean	$\hat{\beta}$	std.error
full graph	3.00	17.96	2.500576	0.001184400
induced subgraph	1.00	9.98	2.533536	0.001531097

Estimated values for the β using given procedures are shown in *Table I*, with goodness of estimation given in terms of standard error. Given error values are in acceptable regions, confirming the hypothesis that the inlink distribution observed in given dataset can indeed be characterized by power-law distribution of the form given in formula *(1)*.

Properties

1. Finally, from Table I we observe estimated values of $\beta = 2.500576$ for full graph and $\beta = 2.533536$ for induced subgraph, which validates the proposed concept of scale-invariance of graph structure.
2. This further indicates that the essential graph properties (like high-degree "authoritative" nodes and random walk convergence properties), existing in the entire graph, are still preserved in the query-induced subgraph. Hence, we can reduce the dimension of *search result clustering* problem, by restating it as a problem of clustering the *query-induced subgraph* G_q, corresponding to the given query q.
3. Such problem relaxation enables us to perform computation in much efficient manner, while still preserving essential information contained in the link structure.

Properties

1. Finally, from Table I we observe estimated values of $\beta = 2.500576$ for full graph and $\beta = 2.533536$ for induced subgraph, which validates the proposed concept of scale-invariance of graph structure.
2. This further indicates that the essential graph properties (like high-degree "authoritative" nodes and random walk convergence properties), existing in the entire graph, are still preserved in the query-induced subgraph. Hence, we can reduce the dimension of *search result clustering* problem, by restating it as a problem of clustering the *query-induced subgraph* G_q, corresponding to the given query q.
3. Such problem relaxation enables us to perform computation in much efficient manner, while still preserving essential information contained in the link structure.

Properties

1. Finally, from Table I we observe estimated values of $\beta = 2.500576$ for full graph and $\beta = 2.533536$ for induced subgraph, which validates the proposed concept of scale-invariance of graph structure.
2. This further indicates that the essential graph properties (like high-degree "authoritative" nodes and random walk convergence properties), existing in the entire graph, are still preserved in the query-induced subgraph. Hence, we can reduce the dimension of *search result clustering* problem, by restating it as a problem of clustering the *query-induced subgraph* G_q, corresponding to the given query q.
3. Such problem relaxation enables us to perform computation in much efficient manner, while still preserving essential information contained in the link structure.

Contents

Algorithm Description

1. We propose an algorithm for graph clustering using random walks on directed power-law graphs.
2. The algorithm operates by performing a number of independent random walks on the link graph and attempts to exploit the specific structure of common power-law graphs in order to bound the average walk length.
3. For each walk, we record a number of times each node was visited, and obtain partial sets, each containing the nodes visited during the walk and appropriate visit counts.
4. Finally, we use that info in order to perform the *merge* stage of the algorithm, in which we use *pivot* nodes (nodes with maximum visit counts), in order to merge the given partial sets into a number of final sets, representing the *cluster set* for a given graph.

Algorithm Description

1. We propose an algorithm for graph clustering using random walks on directed power-law graphs.
2. The algorithm operates by performing a number of independent random walks on the link graph and attempts to exploit the specific structure of common power-law graphs in order to bound the average walk length.
3. For each walk, we record a number of times each node was visited, and obtain partial sets, each containing the nodes visited during the walk and appropriate visit counts.
4. Finally, we use that info in order to perform the *merge* stage of the algorithm, in which we use *pivot* nodes (nodes with maximum visit counts), in order to merge the given partial sets into a number of final sets, representing the *cluster set* for a given graph.

Algorithm Description

1. We propose an algorithm for graph clustering using random walks on directed power-law graphs.
2. The algorithm operates by performing a number of independent random walks on the link graph and attempts to exploit the specific structure of common power-law graphs in order to bound the average walk length.
3. For each walk, we record a number of times each node was visited, and obtain partial sets, each containing the nodes visited during the walk and appropriate visit counts.
4. Finally, we use that info in order to perform the *merge* stage of the algorithm, in which we use *pivot* nodes (nodes with maximum visit counts), in order to merge the given partial sets into a number of final sets, representing the *cluster set* for a given graph.

Algorithm Description

1. We propose an algorithm for graph clustering using random walks on directed power-law graphs.
2. The algorithm operates by performing a number of independent random walks on the link graph and attempts to exploit the specific structure of common power-law graphs in order to bound the average walk length.
3. For each walk, we record a number of times each node was visited, and obtain partial sets, each containing the nodes visited during the walk and appropriate visit counts.
4. Finally, we use that info in order to perform the *merge* stage of the algorithm, in which we use *pivot* nodes (nodes with maximum visit counts), in order to merge the given partial sets into a number of final sets, representing the *cluster set* for a given graph.

Algorithm Description

Let the $G(V, E)$ be the connected, directed graph with $|V| = N$ and $|E| = m$. By *random walk on graph*, we assume Markov chain M_g, where V represents the set of *states* of the chain and $P = [p_{ij}]$ is a stochastic matrix, with p_{ij} representing transitional probability for any two states $i, j \in V$, given by :

$$P_{ij} = \begin{cases} \frac{1}{d(i)}, & \text{if } \exists(i,j)|(i \to j) \in E \\ 0, & \text{if otherwise} \end{cases} \tag{3}$$

and $d(i)$ represents the outdegree of a vertex i.

Algorithm Description

1. We define *stationary distribution* of a Markov chain M_g corresponding to a given walk on graph G, as a probability distribution $\bar{\pi}$, such that $\bar{\pi} = \bar{\pi} * P$, were each entry $\bar{\pi}_i$ is proportional to the amount of time walk will spend in a given node.
2. Such distribution is often used as a measure of *importance* of given node i. In the undirected case, the random walk on the graph converges to the stationary distribution [1], as well as in the case of directed strongly connected graph [12].
3. Allthough this does not hold for the general case of arbitrary walks on power law graphs we perform, it does hold for the case of *strongly connected components* of such graph, which are shown to exist in the general case of power law graphs [11].

Algorithm Description

1. We define *stationary distribution* of a Markov chain M_g corresponding to a given walk on graph G, as a probability distribution $\bar{\pi}$, such that $\bar{\pi} = \bar{\pi} * P$, were each entry $\bar{\pi}_i$ is proportional to the amount of time walk will spend in a given node.
2. Such distribution is often used as a measure of *importance* of given node i. In the undirected case, the random walk on the graph converges to the stationary distribution [1], as well as in the case of directed strongly connected graph [12].
3. Allthough this does not hold for the general case of arbitrary walks on power law graphs we perform, it does hold for the case of *strongly connected components* of such graph, which are shown to exist in the general case of power law graphs [11].

Algorithm Description

1. We define *stationary distribution* of a Markov chain M_g corresponding to a given walk on graph G, as a probability distribution $\bar{\pi}$, such that $\bar{\pi} = \bar{\pi} * P$, were each entry $\bar{\pi}_i$ is proportional to the amount of time walk will spend in a given node.
2. Such distribution is often used as a measure of *importance* of given node i. In the undirected case, the random walk on the graph converges to the stationary distribution [1], as well as in the case of directed strongly connected graph [12].
3. Allthough this does not hold for the general case of arbitrary walks on power law graphs we perform, it does hold for the case of *strongly connected components* of such graph, which are shown to exist in the general case of power law graphs [11].

Algorithm Description

1. We define the *stopping state* of random walk on directed graph as a state corresponding to the *terminating* node, that is node u such that $\nexists v \in V | P_{uv} > 0$.
2. We define the *stopping time* of the walk as a number of steps of M_g it takes for a chain to reach the *stopping state*.
3. For the purpose of a given algorithm, we define *stopping condition* for given walk either as a condition of process entering the *stopping state*, or as a threshold value for the length of the walk.
4. Additionally - we define *maximum walk length* L (usualy of $O(N)$ order), which should prevent infinite loops, yet be large enough for the walk to capture the sufficient approximation of a distribution of node visit counts for given walk.

Algorithm Description

1. We define the *stopping state* of random walk on directed graph as a state corresponding to the *terminating* node, that is node u such that $\nexists v \in V | P_{uv} > 0$.
2. We define the *stopping time* of the walk as a number of steps of M_g it takes for a chain to reach the *stopping state*.
3. For the purpose of a given algorithm, we define *stopping condition* for given walk either as a condition of process entering the *stopping state*, or as a threshold value for the length of the walk.
4. Additionally - we define *maximum walk length* L (usualy of $O(N)$ order), which should prevent infinite loops, yet be large enough for the walk to capture the sufficient approximation of a distribution of node visit counts for given walk.

Algorithm Description

1. We define the *stopping state* of random walk on directed graph as a state corresponding to the *terminating* node, that is node u such that $\nexists v \in V | P_{uv} > 0$.
2. We define the *stopping time* of the walk as a number of steps of M_g it takes for a chain to reach the *stopping state*.
3. For the purpose of a given algorithm, we define *stopping condition* for given walk either as a condition of process entering the *stopping state*, or as a threshold value for the length of the walk.
4. Additionally - we define *maximum walk length* L (usualy of $O(N)$ order), which should prevent infinite loops, yet be large enough for the walk to capture the sufficient approximation of a distribution of node visit counts for given walk.

Algorithm Description

1. We define the *stopping state* of random walk on directed graph as a state corresponding to the *terminating* node, that is node u such that $\nexists v \in V | P_{uv} > 0$.
2. We define the *stopping time* of the walk as a number of steps of M_g it takes for a chain to reach the *stopping state*.
3. For the purpose of a given algorithm, we define *stopping condition* for given walk either as a condition of process entering the *stopping state*, or as a threshold value for the length of the walk.
4. Additionally - we define *maximum walk length* L (usualy of $O(N)$ order), which should prevent infinite loops, yet be large enough for the walk to capture the sufficient approximation of a distribution of node visit counts for given walk.

Algorithm 1 Random Walk Clustering / walk phase

WALK phase:
$i \Leftarrow 0$
while $i \leq K$ **do**
  $S \Leftarrow rand(1, N)$
  while $s \neq 0$ **do**
    if $\nexists s | s \in w_i$ **then**
      $w_i \Leftarrow (s, 1)$
    end if
    $s \Leftarrow rand(adj); v \in adj | \exists (s \Rightarrow v) \in E$
    if $adj = \{\}$ **then**
      $s \Leftarrow 0$
    end if
  end while
end while

Algorithm 2 Random Walk Clustering / merge phase

MERGE phase:
for each $w_i \in W, i \in (1, K)$:
for each node $n \in w_i$:
if $\exists s \in w_m | deg(s) > deg(n)$ **then**
we remove **cut** *node n from* w_i
end if
if $\exists s \in w_m | deg(s) = deg(n)$ **then**
we perform **merge** (of) w_i and w_m
end if
return $C = (w_1 ... w_m)$, $M \leq K$ - *the final set of clusters in given graph*

Algorithm Description

1. We perform the *WALK* phase of the algorithm by selecting $K = k * N$ random nodes, where $k \in (0, 1)$, represents the *approximation constant* of the algorithm, and performing K walks on graph G.
2. Walks are performed untill they reach the stopping condition , either by entering the *stopping state* or by hitting the *maximum walk length*.
3. Finally, in the *MERGE* size, we sort walks by lenght, and internally by visit count, and iterate the result set by performing *CUT* and *MERGE* operations, interchangeably.

Algorithm Description

1. We perform the *WALK* phase of the algorithm by selecting $K = k * N$ random nodes, where $k \in (0, 1)$, represents the *approximation constant* of the algorithm, and performing K walks on graph G.
2. Walks are performed untill they reach the stopping condition , either by entering the *stopping state* or by hitting the *maximum walk length*.
3. Finally, in the *MERGE* size, we sort walks by lenght, and internally by visit count, and iterate the result set by performing *CUT* and *MERGE* operations, interchangeably.

Algorithm Description

1. We perform the *WALK* phase of the algorithm by selecting $K = k * N$ random nodes, where $k \in (0, 1)$, represents the *approximation constant* of the algorithm, and performing K walks on graph G.
2. Walks are performed untill they reach the stopping condition , either by entering the *stopping state* or by hitting the *maximum walk length*.
3. Finally, in the *MERGE* size, we sort walks by lenght, and internally by visit count, and iterate the result set by performing *CUT* and *MERGE* operations, interchangeably.

Algorithm Description

1. If, for a given node, there is a walk having visit count *significantly* greater than in the current walk, we remove it (*CUT*) from given walk, whereas, if there is a walk having *similar* visit count for a given walk, we perform *MERGE* of two walks based on given (*pivot*) node.
2. In such manner, we hope to identify the *key* (*pivot*) nodes for every walk, and perform a join of two walks, in case they share the key nodes.
3. In such manner, we hope to identify the *key* (*pivot*) nodes for every walk, and perform a join of two walks, in case they share the key nodes.

Algorithm Description

1. If, for a given node, there is a walk having visit count *significantly* greater than in the current walk, we remove it (*CUT*) from given walk, whereas, if there is a walk having *similar* visit count for a given walk, we perform *MERGE* of two walks based on given (*pivot*) node.
2. In such manner, we hope to identify the *key* (*pivot*) nodes for every walk, and perform a join of two walks, in case they share the key nodes.
3. In such manner, we hope to identify the *key* (*pivot*) nodes for every walk, and perform a join of two walks, in case they share the key nodes.

Algorithm Description

1. If, for a given node, there is a walk having visit count *significantly* greater than in the current walk, we remove it (*CUT*) from given walk, whereas, if there is a walk having *similar* visit count for a given walk, we perform *MERGE* of two walks based on given (*pivot*) node.
2. In such manner, we hope to identify the *key* (*pivot*) nodes for every walk, and perform a join of two walks, in case they share the key nodes.
3. In such manner, we hope to identify the *key* (*pivot*) nodes for every walk, and perform a join of two walks, in case they share the key nodes.

Contents

Analysis

1. In order to analyze given algorithm, we use results proved in [11], stating that for a class of power law graphs with exponents in range $\beta \in (2, 3)$ (which correspond to the general case of Internet, social and citation networks, such as the dataset analyzed in this paper), average distance between any two is almost surely of order $O(loglogn)$.
2. In such a graph, it is guaranteed that there are more than zero terminating nodes, and the expected average distance between arbitrary node and given terminating node is of order $O(loglogn)$.
3. We can determine that the expected average run length of the $WALK$ phase is or the $O(Nloglogn)$ order.

Analysis

1. In order to analyze given algorithm, we use results proved in [11], stating that for a class of power law graphs with exponents in range $\beta \in (2, 3)$ (which correspond to the general case of Internet, social and citation networks, such as the dataset analyzed in this paper), average distance between any two is almost surely of order $O(loglogn)$.
2. In such a graph, it is guaranteed that there are more than zero terminating nodes, and the expected average distance between arbitrary node and given terminating node is of order $O(loglogn)$.
3. We can determine that the expected average run length of the *WALK* phase is or the $O(Nloglogn)$ order.

Analysis

1. In order to analyze given algorithm, we use results proved in [11], stating that for a class of power law graphs with exponents in range $\beta \in (2, 3)$ (which correspond to the general case of Internet, social and citation networks, such as the dataset analyzed in this paper), average distance between any two is almost surely of order $O(loglogn)$.
2. In such a graph, it is guaranteed that there are more than zero terminating nodes, and the expected average distance between arbitrary node and given terminating node is of order $O(loglogn)$.
3. We can determine that the expected average run length of the $WALK$ phase is or the $O(Nloglogn)$ order.

Analysis

1. Additionally, such graphs contain the *strongly connected component* of the size $n^{c/loglogn}$ [11], therefore, we define the $O(N)$ *maximum walk length* in order to cover walks not hitting the terminating node.
2. This finally results in $O(N^2)$ worst case time for a given algorithm and $O(NloglogN)$ expected average case time for the WALK phase and for a complete algorithm (the merge phase can be implemented efficiently in $O(nloglogn)$ time).

Analysis

1. Additionally, such graphs contain the *strongly connected component* of the size $n^{c/loglogn}$ [11], therefore, we define the $O(N)$ *maximum walk length* in order to cover walks not hitting the terminating node.
2. This finally results in $O(N^2)$ worst case time for a given algorithm and $O(NloglogN)$ expected average case time for the WALK phase and for a complete algorithm (the merge phase can be implemented efficiently in $O(nloglogn)$ time).

Analysis

1. In practice, Efficient graph clustering algorithms vary in computational complexity from $O(n^3)$, in the case of *recursive partitioning*, to $O(nlogn)$ in the case of *multilevel clustering* algorithm described in [10].
2. However, all of given algorithms operate in $O(n^2)$ space complexity, as they require availability of entire graph representation, making it hard for implementation on the scale of n found in practical problems.

Analysis

1. In practice, Efficient graph clustering algorithms vary in computational complexity from $O(n^3)$, in the case of *recursive partitioning*, to $O(nlogn)$ in the case of *multilevel clustering* algorithm described in [10].
2. However, all of given algorithms operate in $O(n^2)$ space complexity, as they require availability of entire graph representation, making it hard for implementation on the scale of n found in practical problems.

Analysis

1. However, although the worst case time of given algorithm is $O(n^2)$, both his average running time, and the fact that by reducing the problem to the induced subgraph, we operate on N which represents the number of nodes matching the given query and is significantly smaller than the total number of nodes in search engine index.
2. Additionally, given random walk implementation is much more space efficient , as it only requires storage of adjacency list for every node $O(NlogN)$ in order to perform random walks and get partial sets, as opposed to the matrix-based eigenvalue methods, which require $O(N^2)$ space for storage of the entire adjacency matrix.

Analysis

1. However, although the worst case time of given algorithm is $O(n^2)$, both his average running time, and the fact that by reducing the problem to the induced subgraph, we operate on N which represents the number of nodes matching the given query and is significantly smaller than the total number of nodes in search engine index.
2. Additionally, given random walk implementation is much more space efficient , as it only requires storage of adjacency list for every node $O(NlogN)$ in order to perform random walks and get partial sets, as opposed to the matrix-based eigenvalue methods, which require $O(N^2)$ space for storage of the entire adjacency matrix.

Results

1. As a part of the research , and as a base for obtaining practical results, we have created a clustering search engine called *RandomNode*, accessible at *http://www.randomnode.com*, which performs query-time clustering of search results by implementing the *Random Walk Clustering* algorithm, proposed in section IV, implemented on top of the *Lucene* search library.
2. It operates on 1.1-million node dataset, represents a significant portion of *.yu* web, generated by performing a crawl starting at the homepage of the Belgrade University (*http://www.bg.ac.yu*).

Results

1. As a part of the research , and as a base for obtaining practical results, we have created a clustering search engine called *RandomNode*, accessible at *http://www.randomnode.com*, which performs query-time clustering of search results by implementing the *Random Walk Clustering* algorithm, proposed in section IV, implemented on top of the *Lucene* search library.
2. It operates on 1.1-million node dataset, represents a significant portion of *.yu* web, generated by performing a crawl starting at the homepage of the Belgrade University (*http://www.bg.ac.yu*).

Results

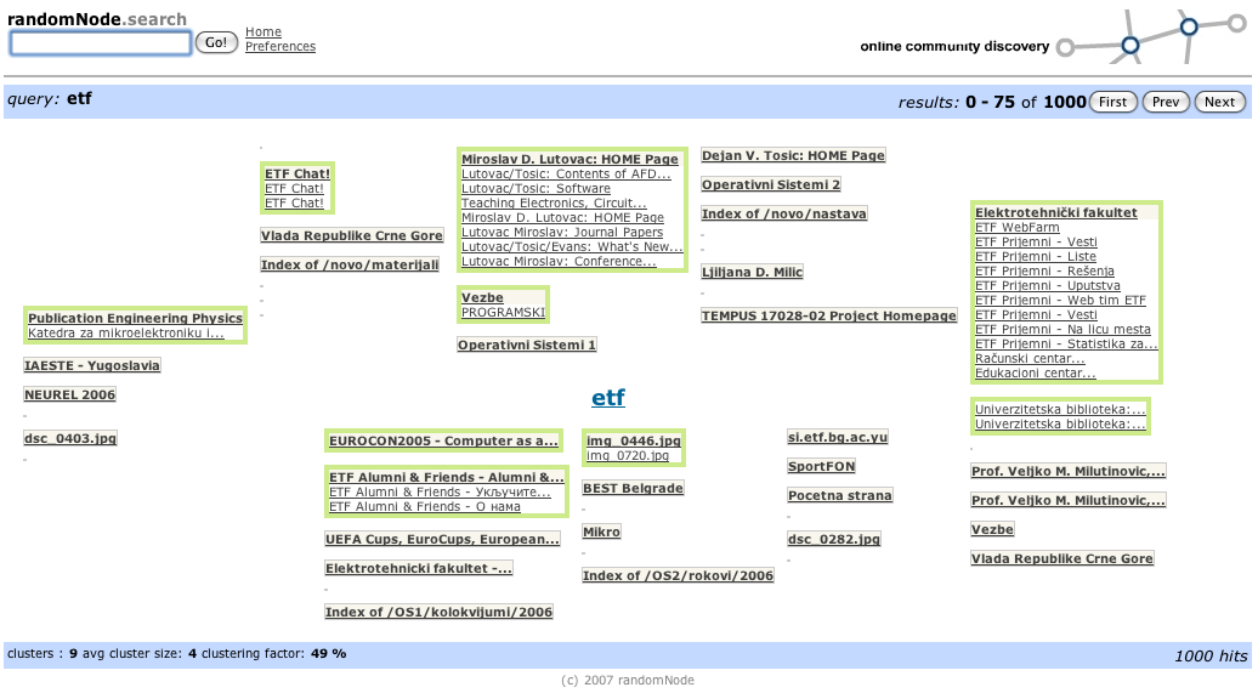

Figure II : randomNode search engine

Results

We use the *randomNode* clustering engine in order to analyze the impact of approximation factor K on the performace of the proposed algorithm. We use the *coverage*(C), of a graph clustering $C = (C_1 ... C_k)$, as as a measure of clustering quality, defined as:

$$coverage(C) = \frac{m(c)}{m} = \frac{m(C)}{m(C) + \bar{m}(C)} \tag{4}$$

where $m(C)$ represents the number of *inter-cluster edges*, while $\bar{m}(C)$ represents a number of *intra-cluster edges*. Optimal clustering should minimize the $\bar{m}(C)$, as it represents the size of the *cut* in the graph performed by given clustering.

Results

1. We perform analysis using *randomNode* engine, by performing clustering on 1000 top-scoring keywords in given dataset, varying the approximation coefficient in the $(0.1, 1.0)$ range with 0.1 step and calculating the *coverage* metric.
2. The results are shown in Figure II, with scatterplot showing exact coverage values for each of each sample instance, and the average coverage given by line segment. We observe that the coverage increases logarithmically with the approximation coefficient, which indicates that the algorithm can be provide acceptable approximations, even for the small values of K.
3. Finally, we use the randomNode engine to extract a set of queries, shown in Figure II, representing top-scoring clusters, both in terms of results and a cluster coverage, for a given subset of *.yu* Web.

Results

1. We perform analysis using *randomNode* engine, by performing clustering on 1000 top-scoring keywords in given dataset, varying the approximation coefficient in the $(0.1, 1.0)$ range with 0.1 step and calculating the *coverage* metric.
2. The results are shown in Figure II, with scatterplot showing exact coverage values for each of each sample instance, and the average coverage given by line segment. We observe that the coverage increases logarithmically with the approximation coefficient, which indicates that the algorithm can be provide acceptable approximations, even for the small values of K.
3. Finally, we use the randomNode engine to extract a set of queries, shown in Figure II, representing top-scoring clusters, both in terms of results and a cluster coverage, for a given subset of *.yu* Web.

Results

1. We perform analysis using *randomNode* engine, by performing clustering on 1000 top-scoring keywords in given dataset, varying the approximation coefficient in the $(0.1, 1.0)$ range with 0.1 step and calculating the *coverage* metric.
2. The results are shown in Figure II, with scatterplot showing exact coverage values for each of each sample instance, and the average coverage given by line segment. We observe that the coverage increases logarithmically with the approximation coefficient, which indicates that the algorithm can be provide acceptable approximations, even for the small values of K.
3. Finally, we use the randomNode engine to extract a set of queries, shown in Figure II, representing top-scoring clusters, both in terms of results and a cluster coverage, for a given subset of *.yu* Web.

Results

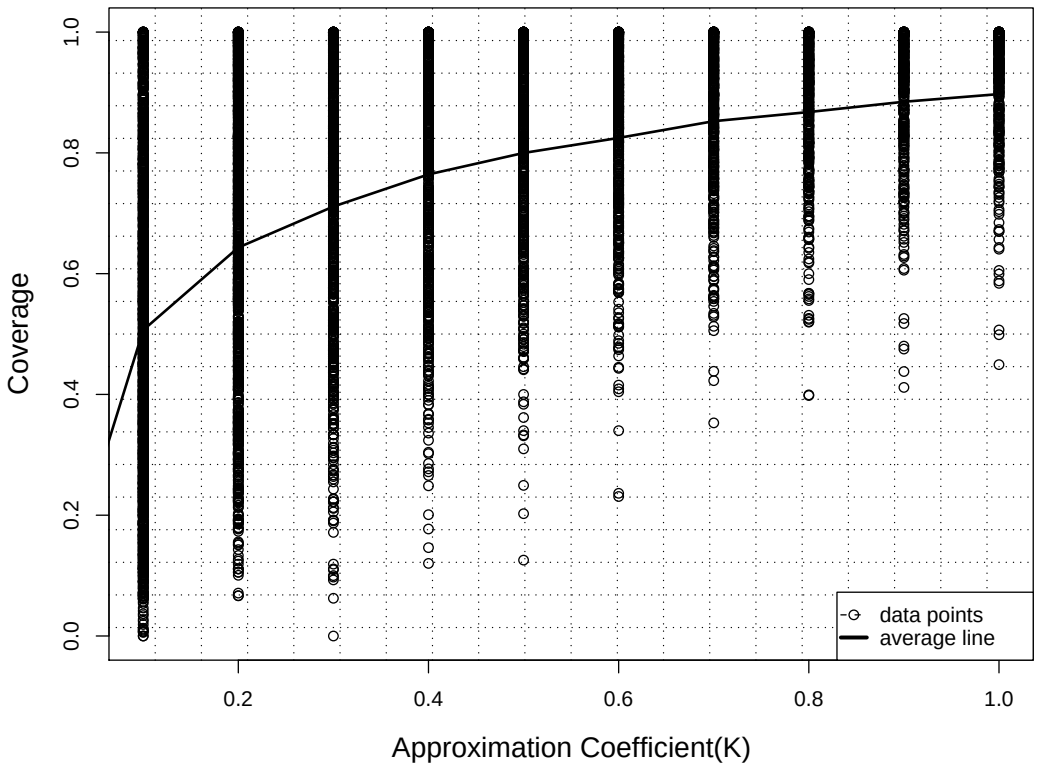

Figure II : algorithm performance as function of approximation coefficient

Results

Finally, we use the randomNode engine to extract a set of queries, shown in Figure II, representing top-scoring clusters, both in terms of results and a cluster coverage, for a given subset of *.yu* Web.

Tabela: Top clusters in randomNode dataset

query	coverage	n.links	incluster	n.clusters	max size
politika	0.999	37473	37417	29	820
pravda	0.967	34688	33556	43	682
rubrike	0.995	33200	33053	13	817
shop	0.967	29440	28482	88	549
nekretnine	0.989	28451	28157	30	535
leasing	0.988	28185	27847	35	272
dekanat	0.947	28783	27264	63	326
banking	0.965	26840	25916	120	211
expo	0.963	26456	24629	69	273
filologija	0.976	23160	22609	39	625

Thanks for listening

Questions?