PROJECT PERIODIC REPORT

Grant Agreement number: 288956

Project acronym: NADINE

Project title: New tools and Algorithms for Directed Network analysis

Funding Scheme: Small or medium-scale focused research project (STREP)

Periodic report: **1st X 2nd**

Period covered: **from 1.5.2012** **to 31.10.2013**

Name, title and organisation of the scientific representative of the project's coordinator[1]:

Dr. Dima Shepelyansky

Directeur de recherche au CNRS

Lab de Phys. Theorique, Universite Paul Sabatier, 31062 Toulouse, France

Tel: +331 5 61556068, Fax: +33 5 61556065, Secr.: +33 5 61557572

E-mail: dima@irsamc.ups-tlse.fr; URL: www.quantware.ups-tlse.fr/dima

Project website address: **www.quantware.ups-tlse.fr/FETNADINE/**

[1] Usually the contact person of the coordinator as specified in Art. 8.1. of the grant agreement

NADINE DELIVERABLE D5.1.

It is based on milestones M7(in progress), M9(in progress), M14(in progress) with deliverable publications:

[8] P1.8 Y.-H.Eom, K.M.Frahm, A.Benczur and D.L. Shepelyansky, **"Time evolution of Wikipedia network ranking"**, submitted Eur. Phys. J. B (2013) (arXiv:1304.6601 [physics.soc-ph], 2013)

[21] P3.3 M.Erdelyi, A.A.Benczur, B.Daroczy, A.Garzo, T.Kiss and D.Siklosi, **"The classification power of Web features"**, Internet Mathematics, to appear (2013)

[22] P3.4 J.Gobolos-Szabo, and A.A.Benczur, **"Temporal Wikipedia search by editr and linkage"**, SIGIR 2013 Workshop on Time-aware Information Access, 28 July - 1 August 2013, Dublin, Ireland

[32] P4.8 P.Boldi, A.Marino, M.Santini and S.Vigna. **"BUbiNG: massive crawling for the masses"** submitted for publication, Oct 2013

EPJ manuscript No.
(will be inserted by the editor)

Time evolution of Wikipedia network ranking

Young-Ho Eom[1], Klaus M. Frahm[1], András Benczúr[2], and Dima L. Shepelyansky[1]

[1] Laboratoire de Physique Théorique du CNRS, IRSAMC, Université de Toulouse, UPS, 31062 Toulouse, France

[2] Informatics Laboratory, Institute for Computer Science and Control,
Hungarian Academy of Sciences (MTA SZTAKI), Pf. 63, H-1518 Budapest, Hungary

Received: April 24, 2013

Abstract. We study the time evolution of ranking and spectral properties of the Google matrix of English Wikipedia hyperlink network during years 2003 - 2011. The statistical properties of ranking of Wikipedia articles via PageRank and CheiRank probabilities, as well as the matrix spectrum, are shown to be stabilized for 2007 - 2011. A special emphasis is done on ranking of Wikipedia personalities and universities. We show that PageRank selection is dominated by politicians while 2DRank, which combines PageRank and CheiRank, gives more accent on personalities of arts. The Wikipedia PageRank of universities recovers 80 percents of top universities of Shanghai ranking during the considered time period.

PACS. 89.75.Fb Structures and organization in complex systems – 89.75.Hc Networks and genealogical trees – 89.20.Hh World Wide Web, Internet

1 Introduction

At present Wikipedia [1] became the world largest Encyclopedia with open public access to its contain. A recent review [2] represents a detailed description of publications and scientific research of this modern Library of Babel, which stores an enormous amount of information, approaching the one described by Jorge Luis Borges [3]. The hyperlinks of citations between Wikipedia articles represent a directed network which reminds the structure of the World Wide Web (WWW). Hence, the mathematical tools developed for WWW search engines, based on the Markov chains [4], Perron-Frobenius operators [5] and the PageRank algorithm of the corresponding Google matrix [6,7], give solid mathematical grounds for analysis of information flow on the Wikipedia network. In this work we perform the Google matrix analysis of Wikipedia network of English articles extending the results presented in [8,9],[10,11]. The main new element of this work is the study of time evolution of Wikipedia network during the years 2003 to 2011. We analyze how the ranking of Wikipedia articles and the spectrum of the Google matrix G of Wikipedia are changed during this period.

The directed network of Wikipedia articles is constructed in a usual way: a directed link is formed from an article j to an article i when j quotes i and an element A_{ij} of the adjacency matrix is taken to be unity when there is such a link and zero in absence of link. Then the matrix S_{ij} of Markov transitions is constructed by normalizing elements of each column to unity ($\sum_j S_{ij} = 1$) and replacing columns with only zero elements (*dangling nodes*) by $1/N$, with N being the matrix size. Then the Google matrix of the network takes the form [6,7]:

$$G_{ij} = \alpha S_{ij} + (1-\alpha)/N \ . \qquad (1)$$

The damping parameter α in the WWW context describes the probability $(1-\alpha)$ to jump to any node for a random surfer. For WWW the Google search engine uses $\alpha \approx 0.85$ [7]. The matrix G belongs to the class of Perron-Frobenius operators [5,7], its largest eigenvalue is $\lambda = 1$ and other eigenvalues have $|\lambda| \leq \alpha$. The right eigenvector at $\lambda = 1$, which is called the PageRank, has real nonnegative elements $P(i)$ and gives a probability $P(i)$ to find a random surfer at site i. It is possible to rank all nodes in a decreasing order of PageRank probability $P(K(i))$ so that the PageRank index $K(i)$ counts all N nodes i according their ranking, placing the most popular articles or nodes at the top values $K = 1, 2, 3....$

Due to the gap $1-\alpha \approx 0.15$ between the largest eigenvalue $\lambda = 1$ and other eigenvalues the PageRank algorithm permits an efficient and simple determination of the PageRank by the power iteration method [7]. It is also possible to use the powerful Arnoldi method [12,13],[14] to compute efficiently the eigenspectrum λ_i of the Google matrix:

$$\sum_{k=1}^{N} G_{jk}\psi_i(k) = \lambda_i \psi_i(j) \ . \qquad (2)$$

The Arnoldi method allows to find a several thousands of eigenvalues λ_i with maximal $|\lambda|$ for a matrix size N as large as a few tens of millions [10,11], [14,15]. Usually,

at $\alpha = 1$ the largest eigenvalue $\lambda = 1$ is highly degenerate [15] due to many invariant subspaces which define many independent Perron-Frobenius operators providing (at least) one eigenvalue $\lambda = 1$.

In addition to a given directed network A_{ij} it is useful to analyze an inverse network with inverted direction of links with elements of adjacency matrix $A_{ij} \to A_{ji}$. The Google matrix G^* of the inverse network is then constructed via corresponding matrix S^* according to the relations (1) using the same value of α as for the G matrix. This time inversion approach was used in [16,17] but the statistical properties and correlations between direct and inversed ranking were not analyzed there. In [18], on an example of the Linux Kernel network, it was shown thus this approach allows to obtain an additional interesting characterization of information flow on directed networks. Indeed, the right eigenvector of G^* at eigenvalue $\lambda = 1$ gives a probability $P^*(i)$, called CheiRank vector [8]. It determines a complementary rank index $K^*(i)$ of network nodes in a decreasing order of probability $P^*(K^*(i))$ [8, 9],[10,18]. It is known that the PageRank probability is proportional to the number of ingoing links characterizing how popular or known is a given node. In a similar way the CheiRank probability is proportional to the number of outgoing links highlighting the node communicativity (see e.g. [7,19], [20,21],[8,9]). The statistical properties of distribution of indexes $K(i), K^*(i)$ on the PageRank-CheiRank plane are described in [9].

In this work we apply the above mathematical methods to the analysis of time evolution of Wikipedia network ranking using English Wikipedia snapshots dated by December 31 of years 2003, 2005, 2007, 2009, 2011. In addition we use the snapshot of August 2009 (200908) analyzed in [8]. The parameters of networks with the number of articles (nodes) N, number of links N_ℓ and other information are given in Tables 1,2 with the description of notations given in Appendix.

The paper is composed as following: the statistical properties of PageRank and CheiRank are analyzed in Section 2, ranking of Wikipedia personalities and universities are considered in Sections 3, 4 respectively, the properties of spectrum of Google matrix are considered in Section 5, the discussion of the results is presented in Section 6, Appendix Section 7 gives network parameters.

2 CheiRank versus PageRank

The dependencies of PageRank and CheiRank probabilities $P(K)$ and $P^*(K^*)$ on their indexes K, K^* at different years are shown in Fig. 1. The top positions of K are occupied by countries starting from *United States* while at the top positions of K^* we find various listings (e.g. geographical names, prime ministers etc.; in 2011 we have appearance of listings of listings). Indeed, the countries accumulate links from all types of human activities and nature, that make them most popular Wikipedia articles, while listings have the largest number of outgoing links making them the most communicative articles.

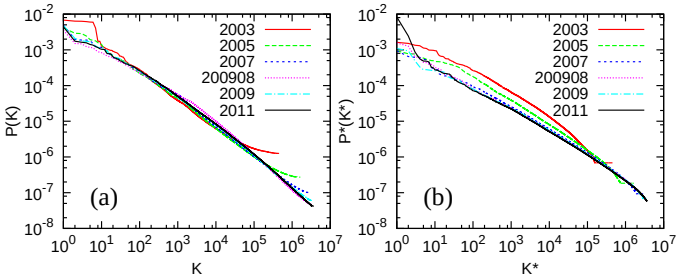

Fig. 1. PageRank probability $P(K)$ (left panel) and CheiRank probability $P^*(K^*)$ (right panel) are shown as a function of the corresponding rank indexes K and K^* for English Wikipedia articles at years 2003, 2005, 2007, 200908, 2009, 2011; here the damping factor is $\alpha = 0.85$.

The data of Fig. 1 show that the global behavior of $P(K)$ remains stable from 2007 to 2011. The probability $P^*(K^*)$ is stable in the time interval 2007 - 2009 while at 2011 we see the appearance of peak at $1 \leq K^* < 10$ that is related to introduction of listings of listings which were absent at earlier years. At the same time the behavior of $P^*(K^*)$ in the range $10 \leq K^* \leq 10^6$ remains stable for 2007 - 2011.

Each article i has its PageRank and CheiRank indexes $K(i)$, $K^*(i)$ so that all articles are distributed on two-dimensional plane of PageRank-CheiRank indexes. Following [8,9] we present the density of articles in the 2D plane (K, K^*) in Fig. 2. The density is computed for 100×100 logarithmically equidistant cells which cover the whole plane (K, K^*) for each year. The density distribution is globally stable for years 2007-2011 even if there are articles which change their location in 2D plane. We see an appearance of a mountain like ridge of probability along a line $\ln K^* \approx \ln K + 4.6$ that indicate the presence of correlation between $P(K(i))$ and $P^*(K^*(i))$. Following [8,9, 18] we characterize the interdependence of PageRank and CheiRank vectors by the correlator

$$\kappa = N \sum_{i=1}^{N} P(K(i)) P^*(K^*(i)) - 1 \ . \qquad (3)$$

We find the following values of the correlator at various time slots: $\kappa = 2.837(2003), 3.894(2005), 4.121(2007), 4.084(200908), 6.629(2009), 5.391(2011)$. During that period the size of the network increased almost by 10 times while κ increased less than 2 times. This confirms the stability of the correlator κ during the time evolution of the Wikipedia network.

In the next two Sections we analyze the time variation of ranking of personalities and universities.

3 Ranking of personalities

To analyze the time evolution of ranking of Wikipedia personalities (persons or humans) we chose the top 100 persons appearing in the ranking list of Wikipedia 200908

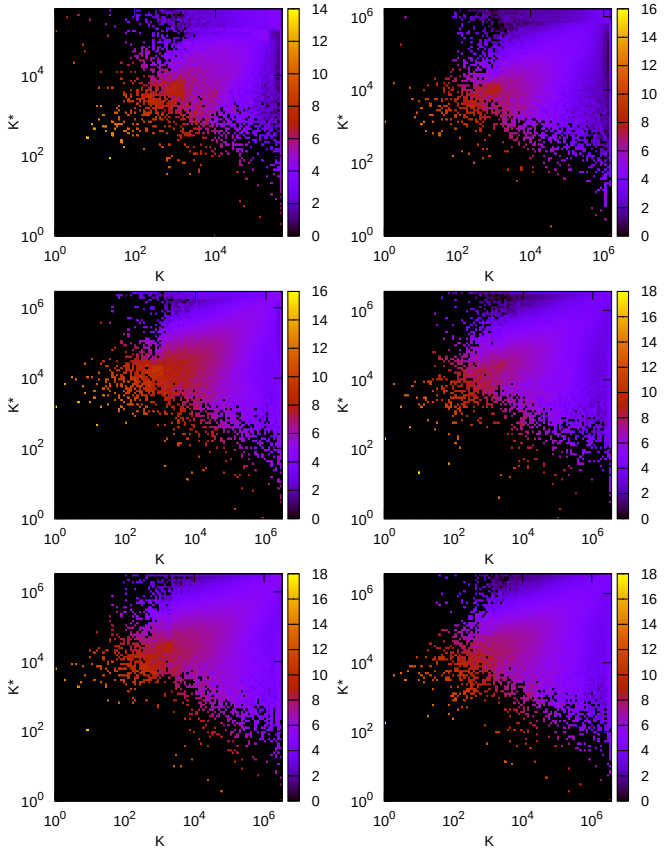

Fig. 2. Density of Wikipedia articles in the CheiRank versus PageRank plane at different years. Color is proportional to logarithm of density changing from minimal nonzero density (dark) to maximal one (white), zero density is shown by black (distribution is computed for 100×100 cells equidistant in logarithmic scale; bar shows color variation of natural logarithm of density); left column panels are for years 2003, 2007, 200908 and right column panels are for 2005, 2009, 2011 (from top to bottom).

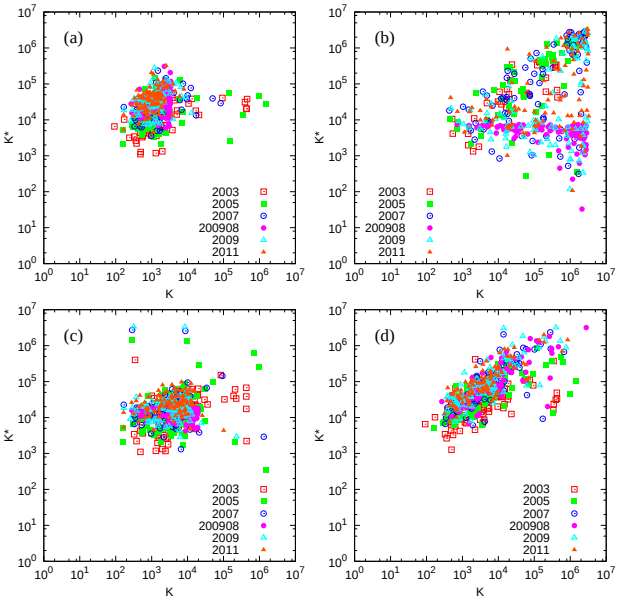

Fig. 3. Change of locations of top-rank persons of Wikipedia in K-K* plane. Each list of top ranks is determined by data of top 100 personalities of time slot 200908 in corresponding rank. Data sets are shown for (a) PageRank, (b) CheiRank, (c) 2DRank, (d) rank from Hart [22].

given in [8] in order of PageRank, CheiRank and 2DRank. We remind that 2DRank K_2 is obtained by counting nodes in order of their appearance on ribs of squares in (K, K^*) plane with their size growing from $K = 1$ to $K = N$ [8].

The distributions of personalities in PageRank-CheiRank plane is shown at various time slots in Fig. 3. There are visible fluctuations of distribution of nodes for years 2003, 2005 when the Wikipedia size has rapid growth. For other years the distribution of top 100 nodes of PageRank and 2DRank is stable even if individual nodes change their ranking. For top 100 of CheiRank the fluctuations remain strong during all years. Indeed, the number of outgoing links is more easy to be modified by authors writing a given article, while a modification of ingoing links depends on authors of other articles.

In Fig. 3 we also show the distribution of top 100 personalities from Hart's book [22] (the list of names is also available at the web page [8]). This distribution also remains stable in years 2007-2011. It is interesting to note that while top PageRank and 2DRank nodes form a kind of droplet in (K, K^*) plane, the distribution of Hart's personalities approximately follows the ridge along the line $\ln K^* \approx \ln K + 4.6$.

The time evolution of top 10 personalities of slot 200908 is shown in Fig. 4 for PageRank and 2DRank. For PageRank the main part of personalities keeps their rank position in time, e.g. G.W.Bush remains at first-second position. B.Obama significantly improves his ranking as a result of president elections. There are strong variations for Elizabeth II which we relate to modification of article name during the considered time interval. We also see a steady improvement of ranking of C.Linnaeus that we attribute to a growth of various botanic descriptions and listings at Wikipedia articles which quote his name. For 2DRank we observe stronger variations of K_2 index with time. Such a politician as R.Nixon has increasing K_2 index with time since the period of his presidency goes in the past. At the same time singers and artists remain at approximately constant level of K_2.

In [8] it was pointed out that the top personalities of PageRank are dominated by politicians while for 2DRank the dominant component of human activity is represented by artists. We analyze the time evolution of the distribution of top 30 personalities over 6 categories of human activity (*politics, arts, science, religion, sport and etc (or others)*). The category *etc* contains only C.Columbus. The results are presented in Fig. 5. They clearly show that the PageRank personalities are dominated by politicians whose percentage increases with time, while the percent of arts decreases. For 2DRank we see that the arts are dominant even if their percentage decreases with time. We also see the appearance of sport which is absent in

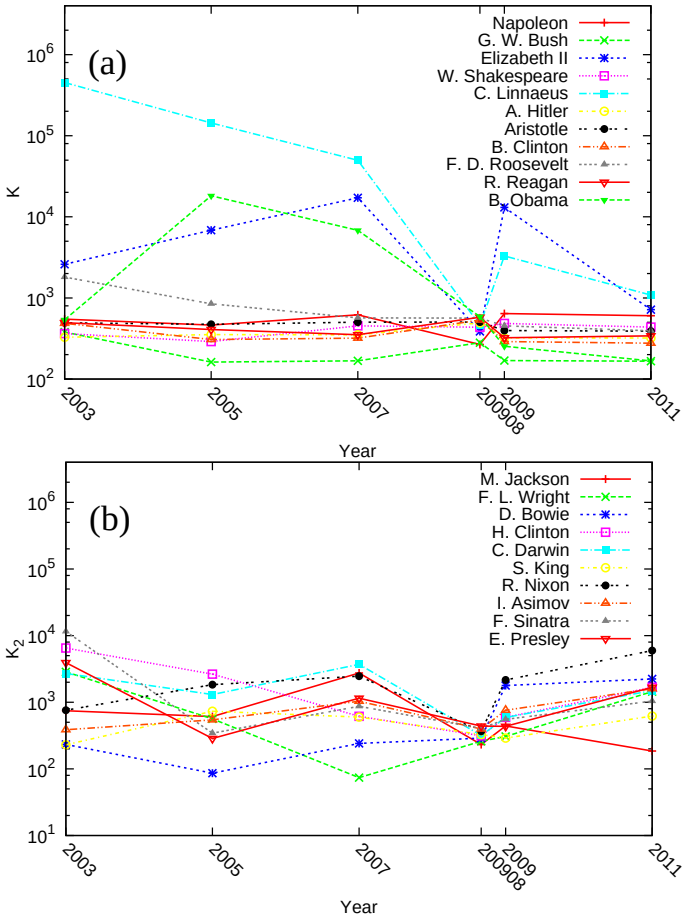

Fig. 4. Time evolution of top 10 personalities of year 200908 in indexes of PageRank K (a) and 2DRank K_2 (b); B.Obama is added in panel (a).

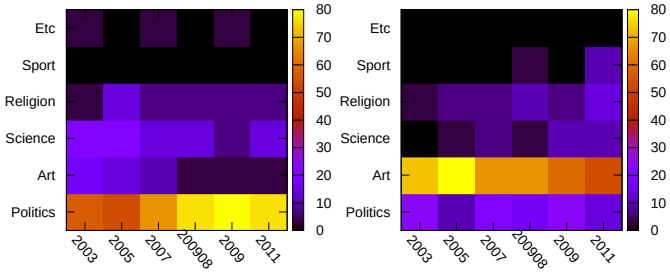

Fig. 5. Left panel: distribution of top 30 PageRank personalities over 6 activity categories at various years of Wikipedia. Right panel: distribution of top 30 2DRank personalities over the same activity categories at same years. Categories are politics, art, science, religion, sport, etc (other). Color shows the number of personalities for each activity expressed in percents.

PageRank. The mechanism of the qualitative ranking differences between two ranks is related to the fact that 2DRank takes into account via CheiRank a contribution of outgoing links. Due to that singers, actors, sportsmen increase their ranking since they are listed in various music albums, movies sport competition results. Due to that the component of arts gets higher positions in 2DRank in contrast to politics dominance in PageRank. Thus the two-dimensional ranking on PageRank-CheiRank plane allows to select qualities of nodes according to their popularity and communicativity.

4 Ranking of universities

The local ranking of top 100 universities is shown in Fig. 6 for years 2003, 2005, 2007 and in Fig. 7 for 2009, 200908, 2011. The local ranking is obtained by selecting top 100 universities appearing in PageRank listing so that they get their university ranking K from 1 to 100. The same procedure is done for CheiRank listing of universities obtaining their local CheiRank index K^* from 1 to 100. Those universities which enter inside 100×100 square on the local index plane (K, K^*) are shown in Figs. 6, 7.

The data show that the top PageRank universities are rather stable in time, e.g. U Harvard is always on the first top position. At the same time the positions in K^* are strongly changing in time. To understand the origin of this variations in CheiRank we consider the case of U Cambridge. Its Wikipedia article in 2003 is rather short but it contains the list of all 31 Colleges with direct links to their corresponding articles. This leads to a high position of U Cambridge with university $K^* = 4$ in 2003 (Fig. 8). However, with time the direct links remain only to about 10 Colleges while the whole number of Colleges are presented by a list of names without links. This leads to a significant increase of index up to $K^* \approx 40$ at Dec 2009. However, at Dec 2011 U Cambridge again improves significantly its CheiRank obtaining $K^* = 2$. The main reason of that is the appearance of section of "Notable alumni and academics" which provides direct links to articles about outstanding scientists studied and/or worked at U Cambridge that leads to second position at $K^* = 2$ among all universities. We note that in 2011 the top CheiRank University is George Mason University with university $K^* = 1$. The main reason of this high ranking is the presence of detailed listings of alumni in politics, media, sport with direct links to articles about corresponding personalities (including former director of CIA). These two examples show that the links, kept with a large number of university alumni, significantly increase CheiRank position of university. We note that artistic and politically oriented universities usually preserve more links with their alumni.

The time evolution of global ranking of top 10 universities of year 200908 for PageRank and 2DRank is shown in Fig. 8. The results show the stability of PageRank order with a clear tendency of top universities (e.g. Harvard) to go with time to higher and higher top positions of K. Thus for U Harvard the global value of K changes from $K \approx 300$ in 2003 to $K \approx 100$ in 2011, while the whole size N of the Wikipedia network increases almost by a factor 10 during this time interval. Since Wikipedia ranks all human knowledge, the stable improvement of PageRank indexes of universities reflects the global growing impor-

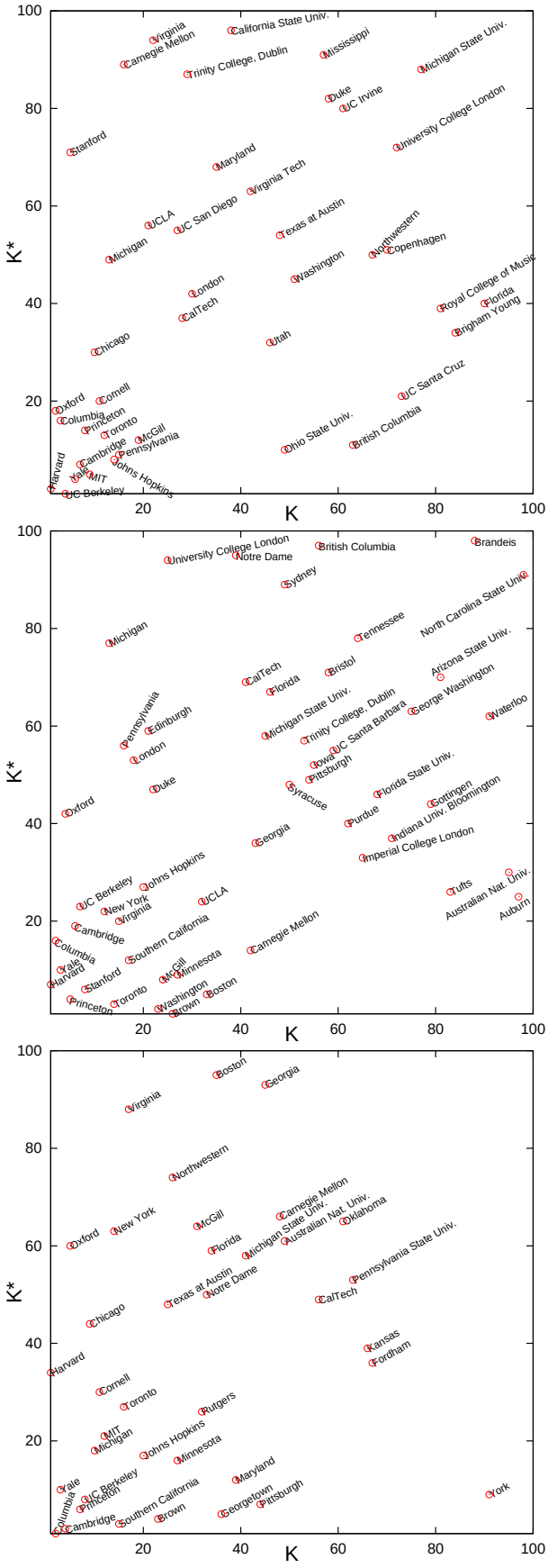

Fig. 6. University of Wikipedia articles in the local CheiRank versus PageRank plane at different years; panels are for years 2003, 2005, 2007 (from top to bottom).

Fig. 7. Same as in Fig. 6 for years 2009, 200908, 2011 (from top to bottom).

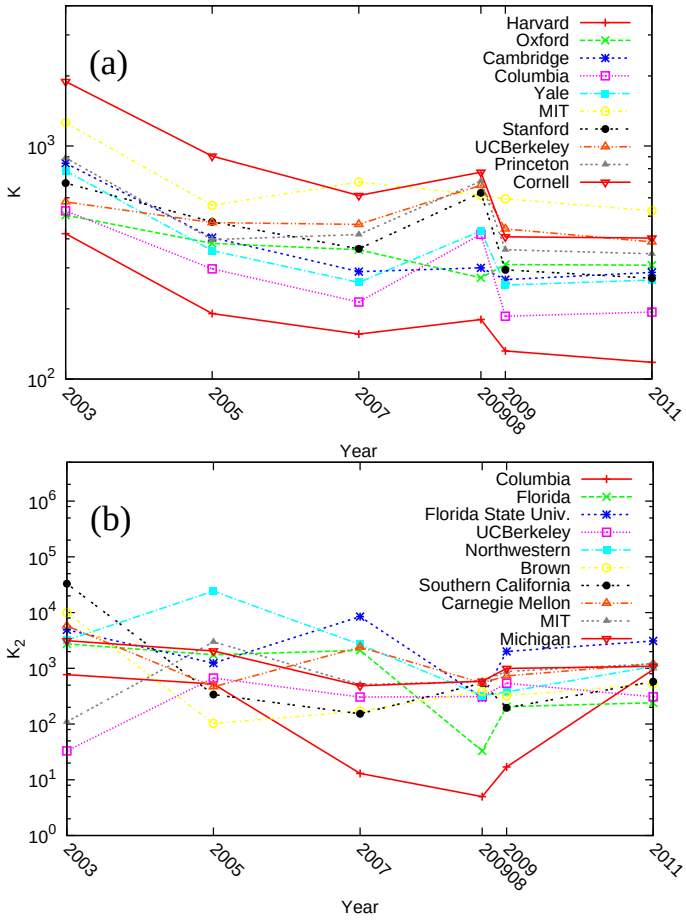

Fig. 8. Time evolution of global ranking of top 10 Universities of year 200908 in indexes of PageRank K (a) and 2DRank K_2 (b).

tance of universities in the world of human activity and knowledge.

The time evolution of the same universities in 2DRank remains stable in time showing certain interchange of their ranking order. We think that an example of U Cambridge considered above explains the main reasons of these fluctuations. In view of 10 times increase of the whole network size during the period 2003 - 2011 the average stability of 2DRank of universities also confirms the significant importance of their place in human activity.

Finally we compare the Wikipedia ranking of universities in their local PageRank index K with those of Shanghai university ranking [23]. In the top 10 of Shanghai university rank the Wikipedia PageRank recovers 9 (2003), 9 (2005), 8 (2007), 7 (2009), 7 (2011). This shows that the Wikipedia ranking of universities gives the results being very close to the real situation. A small decrease of overlap with time can be attributed to earlier launched activity of leading universities on Wikipedia.

5 Google matrix spectrum

Finally we discuss the time evolution of the spectrum of Wikipedia Google matrix taken at $\alpha = 1$. We perform the numerical diagonalization based on the Arnoldi method [12,13] using the additional improvements described in [14,15] with the Arnold dimension $n_A = 6000$. The Google matrix is reduced to the form

$$S = \begin{pmatrix} S_{ss} & S_{sc} \\ 0 & S_{cc} \end{pmatrix} \qquad (4)$$

where S_{ss} describes disjoint subspaces V_j of dimension d_j invariant by applications of S; S_{cc} depicts the remaining part of nodes forming the wholly connected *core space*. We note that S_{ss} is by itself composed of many small diagonal blocks for each invariant subspace and hence those eigenvalues can be efficiently obtained by direct ("exact") numerical diagonalization. The total subspace size N_s, the number of independent subspaces N_d, the maximal subspace dimension $d_{\max}$ and the number N_1 of S eigenvalues with $\lambda = 1$ are given in Table 2 (See also Appendix). The spectrum and eigenstates of the core space S_{cc} are determined by the Arnoldi method with Arnoldi dimension n_A giving the eigenvalues λ_i of S_{cc} with largest modulus. Here we restrict ourselves to the statistical analysis of the spectrum λ_i. The analysis of eigenstates ψ_i ($G\psi_i = \lambda_i\psi_i$), which has been done in [11] for the slot 200908, is left for future studies.

The spectrum for all Wikipedia time slots is shown in Fig. 9 for G and in Fig. 10 for G^*. We see that the spectrum remains stable for the period 2007 - 2001 even if there is a small difference of slot 200908 due to a slightly different cleaning link procedure (see Appendix). For the spectrum of G^* in 2007 - 2001 we observe a well pronounced 3-6 arrow star structure. This structure is very similar to those found in random unistochastic matrices of side 3-4 [24] (see Fig.4 therein). This fact has been pointed in [11] for the slot 200908. Now we see that this is a generic phenomenon which remains stable in time. This indicates that there are dominant groups of 3-4 nodes which have structure similar to random unistochastic matrices with strong ties between 3-4 nodes and various random permutations with random hidden complex phases. The spectral arrow star structure is significantly more pronounce for the case of G^* matrix. We attribute this to more significant fluctuations of outgoing links that probably makes sectors of G^* to be more similar to elements of unistochastic matrices. A further detailed analysis will be useful to understand these arrow star structure and its links with various communities inside Wikipedia.

As it is shown in [11] the eigenstates of G and G^* select certain well defined communities of the Wikipedia network. Such an eigenvector detection of the communities provides a new method of communities detection in addition to more standard methods developed in network science and described in [25]. However, the analysis of eigenvectors represents a separate detailed research and in this work we restrict ourselves to PageRank and CheiRank vectors.

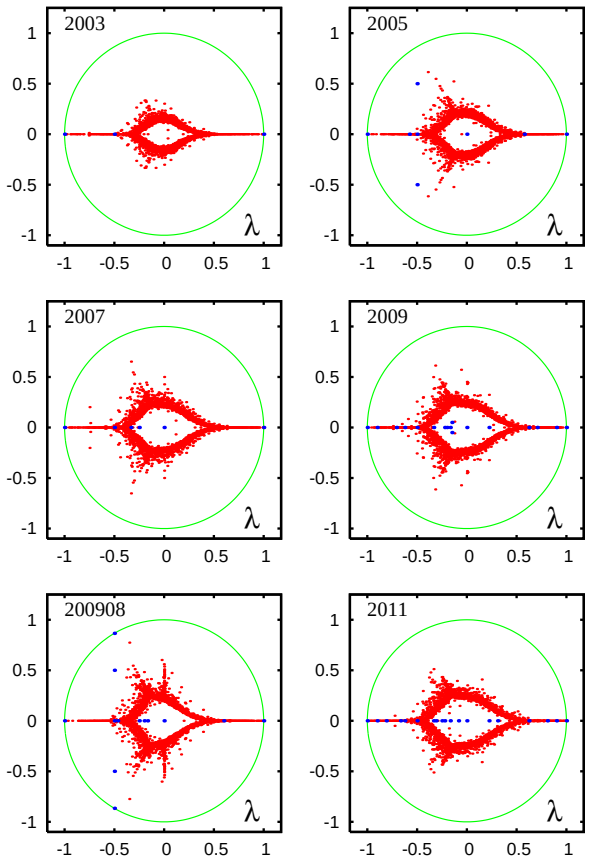

Fig. 9. Spectrum of eigenvalues λ of the Google matrix G of Wikipedia at different years. Red dots are core space eigenvalues, blue dots are subspace eigenvalues and the full green curve shows the unit circle. The core space eigenvalues were calculated by the projected Arnoldi method with Arnoldi dimensions $n_A = 6000$.

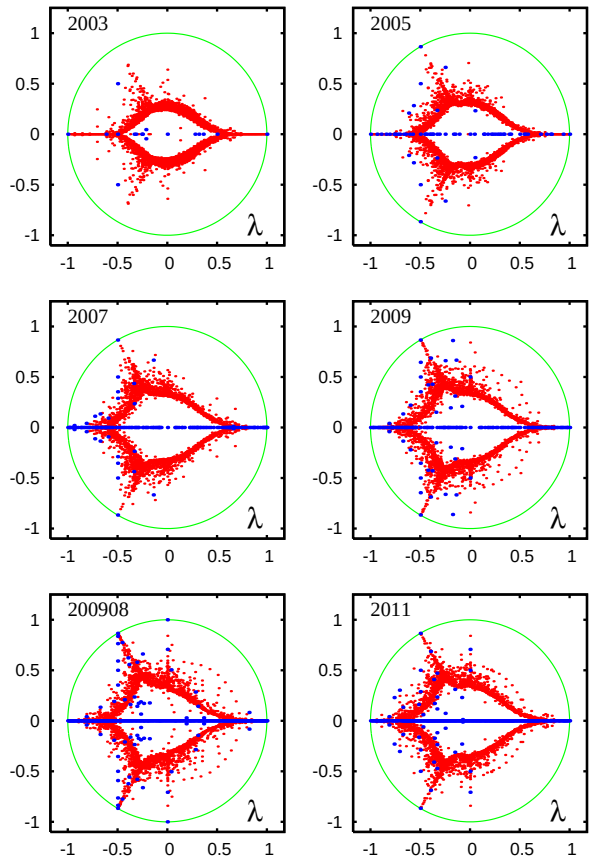

Fig. 10. Same as in Fig. 9 but for the spectrum of matrix G^*.

Finally we note that the fraction of isolated subspaces is very small for G matrix. It is increased approximately by a factor of order 10 for G^* but still it remains very small compared to the networks of UK universities analyzed in [15]. This fact reflects a strong connectivity of network of Wikipedia articles.

6 Discussion

In this work we analyzed the time evolution of ranking of network of English Wikipedia articles. Our study demonstrates the stability of such statistical properties as PageRank and CheiRank probabilities, the article density distribution in PageRank-CheiRank plane during the period 2007 - 2011. The analysis of human activities in different categories shows that PageRank gives main accent to politics while the combined 2DRank gives more importance to arts. We find that with time the number of politicians in the top positions increases. Our analysis of ranking of universities shows that on average the global ranking of top universities goes to higher and higher positions. This clearly marks the growing importance of universities for the whole range of human activities and knowledge. We find that Wikipedia PageRank recovers 70 - 80 % of top 10 universities from Shanghai ranking [23]. This confirms the reliability of Wikipedia ranking.

We also find that the spectral structure of the Wikipedia Google matrix remains stable during the time period 2007 -2011 and show that its arrow star structure reflects certain features of small size unistochastic matrices.

Acknowledgments: Our research presented here is supported in part by the EC FET Open project "New tools and algorithms for directed network analysis" (NADINE *No* 288956). This work was granted access to the HPC resources of CALMIP (Toulouse) under the allocations 2012-P0110, 2013-P0110. We also acknowledge the France-Armenia collaboration grant CNRS/SCS *No* 24943 (IE-017) on "Classical and quantum chaos".

7 Appendix

The tables with all network parameters used in this work are given in the text of the paper. The notations used in the tables are: N is network size, N_ℓ is the number of links, n_A is the Arnoldi dimension used for the Arnoldi method for the core space eigenvalues, N_d is the number of invariant subspaces, $d_{\max}$ gives a maximal subspace dimension, $N_{\rm circ.}$ notes number of eigenvalues on the unit

	N	N_ℓ	n_A
2003	455436	2033173	6000
2005	1635882	11569195	6000
2007	2902764	34776800	6000
2009	3484341	52846242	6000
200908	3282257	71012307	6000
2011	3721339	66454329	6000

Table 1. Parameters of all Wikipedia networks at different years considered in the paper.

	N_s	N_d	$d_{\max}$	$N_{\rm circ.}$	N_1
2003	15	7	3	11	7
2003*	940	162	60	265	163
2005	152	97	4	121	97
2005*	5966	1455	1997	2205	1458
2007	261	150	6	209	150
2007*	10234	3557	605	5858	3569
2009	285	121	8	205	121
2009*	11423	4205	134	7646	4221
200908	515	255	11	381	255
200908*	21198	5355	717	8968	5365
2011	323	131	8	222	131
2011*	14500	4637	1323	8591	4673

Table 2. G and G^* eigespectrum parameters for all Wikipedia networks, year marks spectrum of G, year with star marks spectrum of G^*.

circle with $|\lambda_i| = 1$, N_1 notes number of unit eigenvalues with $\lambda_i = 1$. We remark that $N_s \geq N_{\rm circ.} \geq N_1 \geq N_d$ and $N_s \geq d_{\max}$. The data for G are marked by the corresponding year of the time slot, the data for G^* are marked by the year with a star. Links cleaning procedure eliminates all redirects (nodes with one outgoing link), this procedure is slightly different from the one used for the slot 200908 in [8]. All data sets and high resolution figures are available at the web page [26].

References

1. Wikipedia, *Wikipedia*, `en.wikipedia.org/wiki/Wikipedia`
2. F.A. Nielsen, *Wikipedia research and tools: review and comments*, (2012), available at SSRN: `dx.doi.org/10.2139/ssrn.2129874`
3. J.L. Borges, *The Library of Babel* in *Ficciones* (Grove Press. N.Y. 1962).
4. A.A. Markov, *Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga*, Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete, 2-ya seriya, **15** (1906) 135 (in Russian) [English trans.: *Extension of the limit theorems of probability theory to a sum of variables connected in a chain* reprinted in Appendix B of: R.A. Howard *Dynamic Probabilistic Systems*, volume 1: *Markov models*, Dover Publ. (2007)]
5. M. Brin and G. Stuck, *Introduction to dynamical systems*, Cambridge Univ. Press, Cambridge, UK (2002)
6. S. Brin and L. Page, Computer Networks and ISDN Systems **30**, 107 (1998)
7. A. M. Langville and C. D. Meyer, *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, Princeton (2006)
8. A.O.Zhirov, O.V.Zhirov and D.L.Shepelyansky, Eur. Phys. J. B **77**, 523 (2010); `www.quantware.ups-tlse.fr/QWLIB/2drankwikipedia/`
9. L.Ermann, A.D.Chepelianskii and D.L.Shepelyansky, J. Phys. A: Math. Theor. **45**, 275101 (2012); `www.quantware.ups-tlse.fr/QWLIB/dvvadi/`
10. K.M.Frahm and D.L.Shepelyansky, Eur. Phys. J. B **85**, 355 (2012); `www.quantware.ups-tlse.fr/QWLIB/twittermatrix/`
11. L.Ermann, K.M. Frahm and D.L.Shepelyansky, *Spectral properties of Google matrix of Wikipedia and other networks*, arXiv:1212.1068 [cs.IR] (2012) (Eur. Phys. J. B in press).
12. G.W. Stewart, *Matrix Algorithms Eigensystems*, (SIAM, 2001), Vol. II
13. G.H. Golub and C. Greif, *BIT Num. Math.* **46**, 759 (2006)
14. K.M. Frahm and D.L. Shepelyansky, *Eur. Phys. J. B* **76**, 57 (2010)
15. K.M.Frahm, B.Georgeot and D.L.Shepelyansky, J. Phys, A: Math. Theor. **44**, 465101 (2011)
16. D. Fogaras, *Lect. Notes Comp. Sci.* **2877**, 65 (2003)
17. V. Hrisitidis, H. Hwang and Y. Papakonstantino, *ACM Trans. Database Syst.* **33**, 1 (2008)
18. A. D. Chepelianskii, *Towards physical laws for software architecture*, arXiv:1003.5455[cs.SE] (2010); `www.quantware.ups-tlse.fr/QWLIB/linuxnetwork/`
19. D. Donato, L. Laura, S. Leonardi and S. Millozzi, Eur. Phys. J. B **38**, 239 (2004)
20. G. Pandurangan, P. Raghavan and E. Upfal, Internet Math. **3**, 1 (2005)
21. N. Litvak, W.R.W. Scheinhardt, and Y. Volkovich, Lecture Notes in Computer Science, **4936**, 72 (2008).
22. M.H. Hart, *The 100: ranking of the most influential persons in history*, Citadel Press, N.Y. (1992).
23. `www.shanghairanking.com/`
24. K. Zyczkowski, M. Kus, W. Slomczynski and H.-J. Sommers, J. Phys. A: Math. Gen. **36**, 3425 (2003)
25. S. Fortunato, Phys. Rep. *486*, 75 (2010)
26. `www.quantware.ups-tlse.fr/QWLIB/wikirankevolution/`

The classification power of Web features*

Miklós Erdélyi[1,2], András A. Benczúr[1,3], Bálint Daróczy[1,3], András Garzó[1,3], Tamás Kiss[1,3] and Dávid Siklósi[1,3]

[1]Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SZTAKI)
[2]University of Pannonia, Department of Computer Science and Systems Technology, Veszprém
[3]Eötvös University Budapest

Abstract

In this paper we give a comprehensive overview of features devised for Web spam detection and investigate how much various classes, some requiring very high computational effort, add to the classification accuracy.

- We collect and handle a large number of features based on recent advances in Web spam filtering, including temporal ones, in particular we analyze the strength and sensitivity of linkage change.
- We propose new temporal link similarity based features and show how to compute them efficiently on large graphs.
- We show that machine learning techniques including ensemble selection, LogitBoost and Random Forest significantly improve accuracy.
- We conclude that, with appropriate learning techniques, a simple and computationally inexpensive feature subset outperforms all previous results published so far on our data set and can only slightly be further improved by computationally expensive features.
- We test our method on three major publicly available data sets, the Web Spam Challenge 2008 data set WEBSPAM-UK2007, the ECML/PKDD Discovery Challenge data set DC2010 and the Waterloo Spam Rankings for ClueWeb09.

*This work was supported in part by the EC FET Open project "New tools and algorithms for directed network analysis" (NADINE No 288956), by the EU FP7 Project LAWA—Longitudinal Analytics of Web Archive Data, OTKA NK 105645 and by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013). The research was carried out as part of the EITKIC_12-1-2012-0001 project, which is supported by the Hungarian Government, managed by the National Development Agency, financed by the Research and Technology Innovation Fund and was performed in cooperation with the EIT ICT Labs Budapest Associate Partner Group. (`www.ictlabs.elte.hu`) This paper is a comprehensive comparison of the best performing classification techniques based on [9, 37, 36, 38] and new experiments.

Our classifier ensemble sets the strongest classification benchmark as compared to participants of the Web Spam and ECML/PKDD Discovery Challenges as well as the TREC Web track.

To foster research in the area, we make several feature sets and source codes public[1], including the temporal features of eight `.uk` crawl snapshots that include WEBSPAM-UK2007 as well as the Web Spam Challenge features for the labeled part of ClueWeb09.

1 Introduction

Web classification finds several use, both for content filtering and for building focused corpora from a large scale Web crawl. As one notable use, Internet archives actively participate in large scale experiments [8], some of them building analytics services over their collections [6]. Most of the existing results on Web classification originate from the area of Web spam filtering that have turned out to generalize to a wide class of tasks including genre, Open Directory category, as well as quality classification. Closely related areas include filtering and tagging in social networks [50].

Web spam filtering, the area of devising methods to identify useless Web content with the sole purpose of manipulating search engine results, has drawn much attention in the past years [63, 49, 46]. The first mention of Web spam, termed *spamdexing* as a combination of words *spam* and (search engine) *indexing*, appears probably in a 1996 news article [27] as part of the early Web era discussions on the spreading porn content [24]. In the area of the so-called Adversarial Information Retrieval workshop series ran since 2005 [40] and evaluation campaigns including the Web Spam Challenges [18], the ECML/PKDD Discovery Challenge 2010 [50] and the Spam task of TREC 2010 Web Track [29] were organized. A recent comprehensive survey on Web spam filtering research is found in [19].

In this paper we present, to our best knowledge, the most comprehensive experimentation based on content, link as well as temporal features, both new and recently published. Our spam filtering baseline classification procedures are collected by analyzing the results [28, 1, 44] of the Web Spam Challenges and the ECML/PKDD Discovery Challenge 2010 [45, 2, 58]. Our comparison is based on AUC values [42] that we believe to be more stable as it does not depend on the split point; indeed, while Web Spam Challenge 2007 used F-measure and AUC, Web Spam Challenge 2008 used AUC only as evaluation measure.

Web spam appears in sophisticated forms that manipulate content as well as linkage [47] with examples such as

- Copied content, "honey pots" that draw attention but link to unrelated, spam targets;
- Garbage content, stuffed with popular or monetizable query terms and phrases such as university degrees, online casinos, bad credit status or

[1] `https://datamining.sztaki.hu/en/download/web-spam-resources`

adult content;

- Link farms, a large number of strongly interlinked pages across several domains.

The Web spammer toolkit consists of a clearly identifiable set of manipulation techniques that has not changed much recently. The Web Spam Taxonomy of Gyöngyi et al. [47] distinguishes content (term) and link spamming along with techniques of hiding, cloaking and removing traces by e.g. obfuscated redirection. Most of the features designed fight either link or content spamming.

We realize that recent results have ignored the importance of the machine learning techniques and concentrated only on the definition of new features. Also the only earlier attempt to unify a large set of features [20] is already four years old and even there little comparison is given on the relative power of the feature sets. For classification techniques, a wide selection including decision trees, random forest, SVM, class-feature-centroid, boosting, bagging and oversampling in addition to feature selection (Fisher, Wilcoxon, Information Gain) were used [45, 2, 58] but never compared and combined. In this paper we address the following questions.

- Do we get the maximum value out of the features we have? Are we sufficiently sophisticated at applying machine learning?
- Is it worth calculating computationally expensive features, in particular some related to page-level linkage?
- What is an optimal feature set for a fast spam filter that can quickly react at crawl time after fetching a small sample of a Web site?

We compare our result with the very strong baselines of the Web Spam Challenge 2008 and ECML/PKDD 2010 Discovery Challenge data sets. Our main results are as follows.

- We apply state-of-the-art classification techniques by the lessons learned from KDD Cup 2009 [57]. Key in our performance is ensemble classification applied both over different feature subsets as well as over different classifiers over the same features. We also apply classifiers yet unexplored against Web spam, including Random Forest [14] and LogitBoost [43].
- We compile a small yet very efficient feature set that can be computed by sample pages from the site while completely ignoring linkage information. By this feature set a filter may quickly react to a recently discovered site and intercept in time before the crawler would start to follow a large number of pages from a link farm. This feature set itself reaches AUC 0.893 over WEBSPAM-UK2007.
- Last but not least we gain strong improvements over the Web Spam Challenge best performance [18]. Our best result in terms of AUC reaches 0.9 and improves on the best Discovery Challenge 2010 results.

Several recent papers propose temporal features [61, 55, 31, 52] to improve classification accuracy. We extend link-based similarity algorithms by proposing

metrics to capture the linkage change of Web pages over time. We describe a method to calculate these metrics efficiently on the Web graph and then measure their performance when used as features in Web spam classification. We propose an extension of two link-based similarity measures: XJaccard and PSimRank [41].

We investigate the combination of temporal and non-temporal, both link- and content-based features using ensemble selection. We evaluate the performance of ensembles built on the latter feature sets and compare our results to that of state-of-the-art techniques reported on our dataset. Our conclusion is that temporal and link-based features in general do not significantly increase Web spam filtering accuracy. However, information about linkage change might improve the performance of a language independent classifier: the best results for the French and German classification tasks of the ECML/PKDD Discovery Challenge [45] were achieved by using host level link features only, outperforming those who used all features [2].

In this paper we address not just the quality but also the computational efficiency. Earlier lightweight classifiers include Webb et al. [64] describing a procedure based solely on the HTTP session information. Unfortunately they only measure precision, recall and F-measure that are hard to compare with later results on Web spam that use AUC. In fact the F and similar measures greatly depend on the classification threshold and hence make comparison less stable and for this reason they are not used starting with the Web Spam Challenge 2008. Furthermore in [64] the IP address is a key feature that is trivially incorporated in the DC2010 data set by placing all hosts from the same IP address into the same training or testing set. The intuition is that if an IP address contains spam hosts, all hosts from that IP address are likely to be spam and should be immediately manually checked and excluded from further consideration.

The rest of this paper is organized as follows. In Section 2 we describe the data sets used in this paper. We give an overview of temporal features for spam detection and propose new temporal link similarity based ones in Section 3. In Section 4 we describe our classification framework. The results of the experiments to classify WEBSPAM-UK2007, ClueWeb09 and DC2010 can be found in Section 5. The computational resource needs of various feature sets are summarized in Section 6.

2 Data Sets

In this paper we use three data sets, WEBSPAM-UK2007 of the Web Spam Challenge 2008 [18], the Waterloo Spam Rankings for ClueWeb09, and DC2010 created for the ECML/PKDD Discovery Challenge 2010 on Web Quality. We only give a brief summary of the first data set described well in [18, 22] and the second in [38], however, we describe the third one in more detail in Section 2.3. Also we compare the amount of spam in the data sets.

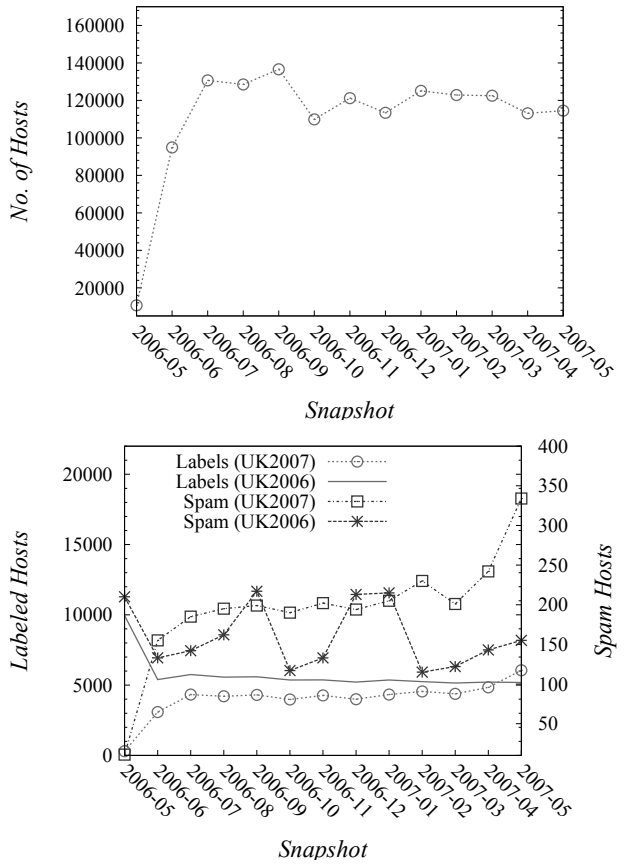

Figure 1: The number of total and labeled hosts in the 13 UK snapshots. We indicate the number of positive and negative labels separate for the WEBSPAM-UK2006 and WEBSPAM-UK2007 label sets.

2.1 Web Spam Challenge 2008: WEBSPAM-UK2007

The Web Spam Challenge was first organized in 2007 over the WEBSPAM-UK2006 data set. The last Challenge over the WEBSPAM-UK2007 set was held in conjunction with AIRWeb 2008 [18]. The Web Spam Challenge 2008 best result [44] achieved an AUC of 0.85 by also using ensemble undersampling [23]. They trained a bagged classifier on the standard content-based and link-based features published by the organizers of the Web Spam Challenge 2008 and on custom host-graph based features, using the ERUS strategy for class-inbalance learning. For earlier challenges, best performances were achieved by a semi-supervised version of SVM [1] and text compression [28]. Best results either used bag of words vectors or the so-called "public" feature sets of [20].

We extended the WEBSPAM-UK2007 data set with 13 `.uk` snapshots pro-

Label Set	Instances	%Positive
Training	4000	5.95%
Testing	2053	4.68%

Table 1: Summary of label sets for Web Spam Challenge 2008.

vided by the Laboratory for Web Algorithmics of the Università degli studi di Milano. We use the training and testing labels of the Web Spam Challenge 2008, as summarized in Table 1. In order to prepare a temporal collection, we extracted maximum 400 pages per site from the original crawls. The last 12 of the above `.uk` snapshots were analyzed by Bordino et al. [12] who observe a relative low URL but high host overlap[2]. The first snapshot (2006-05) that is identical to WEBSPAM-UK2006 was chosen to be left out from their experiment since it was provided by a different crawl strategy. We observed that in the last eight snapshots the number of hosts have stabilized in the sample and these snapshots have roughly the same amount of labeled hosts as seen in Fig. 1. From now on we restrict attention to the aforementioned subset of the snapshots and the WEBSPAM-UK2007 labels only.

2.2 The Waterloo Spam Rankings for ClueWeb09

The English part of ClueWeb09 consist of approximately 20M domains and 500M pages. For Web spam labels we used the Waterloo Spam Rankings [29]. While the Waterloo Spam Rankings contain negative training instances as well, we extended the negative labels with the set of the Open Directory Project (ODP) hosts. We used 50% split for training and testing.

We labeled hosts in both the `.pt` crawl and ClueWeb09 by top-level ODP categories using links extracted from topic subtrees in the directory. Out of all labeled hosts, 642643 received a unique label. Because certain sites (e.g., `bbc.co.uk`) may belong to even all 14 top-level English categories, we discarded the labels of 18734 hosts with multiple labels to simplify the multi-label task. As Bordino et al. [13] indicate, multitopical hosts are often associated to poor quality sites and spam as another reason why their labels may mislead the classification process. The resulting distribution of labels is shown in Table 2.

2.3 Discovery Challenge 2010: DC2010

The Discovery Challenge was organized over DC2010, a new data set that we describe in more detail next. DC2010 is a large collection of annotated Web hosts labeled by the Hungarian Academy of Sciences (English documents), Internet Memory Foundation (French) and L3S Hannover (German). The base data is a set of 23M pages in 190K hosts in the `.eu` domain crawled by the Internet Memory Foundation in early 2010.

[2]The dataset can be downloaded from: `http://law.di.unimi.it/datasets.php`

Category	No. of Hosts	% of Labeled Hosts
spam	439	0.07%
Arts	97355	15.1%
Business	193678	30.1%
Computers	66159	10.3%
Recreation	65594	10.2%
Science	43317	6.7%
Society	122084	19%
Sports	54456	8.5%

Table 2: Number of positive ClueWeb09 host labels for spam and the ODP categories.

	UK2006	UK2007	ClueWeb09	DC2010			
				en	de	fr	all
Hosts	10 660	114 529	500,000	61 703	29 758	7 888	190 000
Spam	19.8%	5.3%	unknown	8.5% of valid labels; 5% of all in large domains.			

Table 3: Fraction of Spam in WEBSPAM-UK2006, UK2007, ClueWeb09 and DC2010. Note that three languages English, German and French were selected for labeling DC2010, although Polish and Dutch language hosts constitute a larger fraction than the French. Since to our best knowledge, no systematic random sample was labeled for ClueWeb09, the number 439 of labeled spam hosts is not representative for the collection.

The labels extend the scope of previous data sets on Web Spam in that, in addition to sites labeled spam, we included manual classification for genre into five categories Editorial, Commercial, Educational, Discussion and Personal as well as trust, factuality and bias as three aspects of quality. Spam label is exclusive since no other assessment was made for spam. However other labels are non-exclusive and hence define nine binary classification problems. We consider no multi-class tasks in this paper. Assessor instructions are for example summarized in [62], a paper concentrating on quality labels.

In Table 3, we summarize the amount of spam in the DC2010 data set in comparison with the Web Spam Challenge data sets. This amount is well-defined for the latter data sets by the way they were prepared for the Web Spam Challenge participants. However for DC2010, this figure may be defined in several ways. First of all, when creating the DC2010 labels, eventually we considered domains with or without a `www.` prefix the same such as `www.domain.eu` vs. `domain.eu`. However in our initial sampling procedure we considered them as two different hosts and merged them after verifying that the labels of the two versions were identical. Also, several domains consist of a single redirection page to another domain and we counted these domains, too. Finally, a large fraction of spam is easy to spot and can be manually removed. As an example of many

Count	IP address	Comment
3544	80.67.22.146	spam farm `*-palace.eu`
3198	78.159.114.140	spam farm `*auts.eu`
1374	62.58.108.214	`blogactiv.eu`
1109	91.204.162.15	spam farm `x-mp3.eu`
1070	91.213.160.26	spam farm `a-COUNTRY.eu`
936	81.89.48.82	`autobazar.eu`
430	78.46.101.76	spam farm `77k.eu` and 20+ domains
402	89.185.253.73	spam farm `mp3-stazeni-zdarma.eu`

Table 4: Selection of IP addresses with many subdomains in the DC2010 data set.

Label	Group	Yes	Maybe	No
Spam	*Spam*	423		4 982
News/Editorial	*Genre*	191		4 791
Commercial		2 064		2 918
Educational		1 791		3 191
Discussion		259		4 724
Personal-Leisure		1 118		3 864
Non-Neutrality	*Quality*	19	216	3 778
Bias		62		3 880
Dis-Trustiness		26	201	3 786

Table 5: Distribution of assessor labels in the DC2010 data set.

hosts on same IP, we include a labeled sample from DC2010, that itself contains over 10,000 spam domains in Table 4. These hosts were identified by manually looking at the IP addresses that serve the largest number of domain names. Thus our sample is biased and obtaining an estimate of the spam fraction is nontrivial, as indicated in Table 3.

The distribution of labels for the nine categories with more than 1% positive samples (spam, news, commercial, educational, discussion, personal, neutral, biased, trusted) is given in Table 5. For Neutrality and Trust the strong negative categories have low frequency and hence we fused them with the intermediate negative (maybe) category for the training and testing labels.

The Discovery Challenge 2010 best result [58] achieved an AUC of 0.83 for spam classification while the overall winner [45] was able to classify a number of quality components at an average AUC of 0.80. As for the technologies, bag of words representation variants proved to be very strong for the English collection while only language independent features were used for German and French. The applicability of dictionaries and cross-lingual technologies remains open.

New to the construction of the DC2010 training and test set is the handling of hosts from the same domain and IP address. Since no IP address and domain was allowed to be split between training and testing, we might have to

reconsider the applicability of propagation [48, 66] and graph stacking [54]. The Web Spam Challenge data sets were labeled by uniform random sampling and graph stacking appeared to be efficient in several results [22] including our prior work [30]. The applicability of graph stacking remains however unclear for the DC2010 data set. Certain teams used some of these methods but reported no improvement [2].

3 Temporal Features for Spam Detection

Spammers often create bursts in linkage and content: they may add thousands or even millions of machine generated links to pages that they want to promote [61] that they again very quickly regenerate for another target or remove if blacklisted by search engines. Therefore changes in both content and linkage may characterize spam pages.

Recently the evolution of the Web has attracted interest in defining features, signals for ranking [34] and spam filtering [61, 55, 31, 52, 37]. The earliest results investigate the changes of Web content with the primary interest of keeping a search engine index up-to-date [25, 26]. The decay of Web pages and links and its consequences on ranking are discussed in [4, 35]. One main goal of Boldi et al. [11] who collected the `.uk` crawl snapshots also used in our experiments was the efficient handling of time-aware graphs. Closest to our temporal features is the investigation of host overlap, deletion and content dynamics in the same data set by Bordino et al. [12].

Perhaps the first result on the applicability of temporal features for Web spam filtering is due to Shen et al. [61] who compare pairs of crawl snapshots and define features based on the link growth and death rate. However by extending their ideas to consider multi-step neighborhood, we are able to define a very strong feature set that can be computed by the Monte Carlo estimation of Fogaras and Rácz [41]. Another result defines features based on the change of the content [31] who obtain page history from the Wayback Machine.

For calculating the temporal link-based features we use the host level graph. As observed in [12], pages are much more unstable over time compared to hosts. Note that page-level fluctuations may simply result from the sequence the crawler visited the pages and not necessarily reflect real changes. The inherent noise of the crawling procedure and problems with URL canonization [5] rule out the applicability of features based on the change of page-level linkage.

3.1 Linkage Change

In this section we describe link-based temporal features that capture the extent and nature of linkage change. These features can be extracted from either the page or the host level graph where the latter has a directed link from host a to host b if there is a link from a page of a to a page of b.

The starting point of our new features is the observation of [61] that the in-link growth and death rate and change of clustering coefficient characterize the

evolution patterns of spam pages. We extend these features for the multi-step neighborhood in the same way as PageRank extends the in-degree. The *ℓ-step neighborhood* of page v is the set of pages reachable from v over a path of length at most ℓ. The ℓ-step neighborhood of a host can be defined similarly over the host graph.

We argue that the changes in the multi-step neighborhood of a page should be more indicative of the spam or honest nature of the page than its single-step neighborhood because spam pages are mostly referred to by spam pages [21], and spam pages can be characterized by larger change of linkage when compared to honest pages [61].

In the following we review the features related to linkage growth and death from [61] in Section 3.1.1, then we introduce new features based on the similarity of the multi-step neighborhood of a page or host. We show how the XJaccard and PSimRank similarity measure can be used for capturing linkage change in Section 3.1.3 and Section 3.1.4, respectively.

3.1.1 Change Rate of In-links and Out-links

We compute the following features introduced by Shen et al. [61] on the host level for a node a for graph instances from time t_0 and t_1. We let $G(t)$ denote the graph instance at time t and $I^{(t)}(a)$, $\Gamma^{(t)}(a)$ denote the set of in and out-links of node a at time t, respectively.

- In-link death (IDR) and growth rate (IGR):

$$\mathrm{IDR}(a) = \frac{|I^{(t_0)}(a)| - |I^{(t_0)}(a) \cap I^{(t_1)}(a)|}{|I^{(t_0)}(a)|}$$

$$\mathrm{IGR}(a) = \frac{|I^{(t_1)}(a)| - |I^{(t_0)}(a) \cap I^{(t_1)}(a)|}{|I^{(t_0)}(a)|}$$

- Out-link death and growth rates (ODR, OGR): the above features calculated for out-links;
- Mean and variance of IDR, IGR, ODR and OGR across in-neighbors of a host (IDRMean, IDRVar, etc.);
- Change rate of the clustering coefficient (CRCC), i.e. the fraction of linked hosts within those pointed by pairs of edges from the same host:

$$CC(a,t) = \frac{|\{(b,c) \in G(t) | b,c \in \Gamma^{(t)}(a)|}{|\Gamma^{(t)}(a)|}$$

$$CRCC(a) = \frac{CC(a,t_1) - CC(a,t_0)}{CC(a,t_0)}$$

- Derivative features such as the ratio and product of the in and out-link rates, means and variances. We list the in-link derivatives; out-link ones are defined similarly:

IGR·IDR, IGR/IDR, IGRMean/IGR, IGRVar/IGR, IDRMean/IDR, IDRVar/IDR, IGRMean · IDRMean, IGRMean/IDRMean, IGRVar · IDRVar, IGRVar/IDRVar.

3.1.2 Self-Similarity Along Time

In the next sections we introduce new linkage change features based on multi-step graph similarity measures that in some sense generalize the single-step neighborhood change features of the previous section. We characterize the change of the multi-step neighborhood of a node by defining the similarity of a single node *across* snapshots instead of two nodes within a single graph instance. The basic idea is that, for each node, we measure its similarity to itself in two identically labeled graphs representing two consecutive points of time. This enables us to measure the linkage change occurring in the observed time interval using ordinary graph similarity metrics.

First we describe our new contribution, the extension of two graph similarity measures, XJaccard and PSimRank [41] to capture temporal change; moreover, we argue why SimRank [51] is inappropriate for constructing temporal features.

SimRank of a pair of nodes u and v is defined recursively as the average similarity of the neighbors of u and v:

$$\begin{aligned} \mathrm{Sim}_{\ell+1}(u,v) &= 0, \text{ if } I(u) \text{ or } I(v) \text{ is empty;} \\ \mathrm{Sim}_{\ell+1}(u,v) &= 1, \text{ if } u = v; \\ \mathrm{Sim}_{\ell+1}(u,v) &= \frac{c}{|I(u)||I(v)|} \sum_{\substack{v' \in I(v) \\ u' \in I(u)}} \mathrm{Sim}_{\ell}(u',v'), \end{aligned} \tag{1}$$

where $I(x)$ denotes the set of vertices linking to x and $c \in (0,1)$ is a decay factor. In order to apply SimRank for similarity of a node v between two snapshots t_0 and t_1, we apply (2) so that v' and u' are taken from different snapshots.

Next we describe a known deficiency of SimRank in its original definition that rules out its applicability for temporal analysis. First we give the example for the single graph SimRank. Consider a bipartite graph with k nodes pointing all to another two u and v. In this graph there are no directed paths of length more than one and hence the Sim values can be computed in a single iteration. Counter-intuitively, we get $\mathrm{Sim}(u,v) = c/k$, i.e. the larger the cocitation of u and v, the smaller their SimRank value. The reason is that the more the number of in-neighbors, the more likely is that a pair of random neighbors will be different.

While the example of the misbehavior for SimRank is somewhat artificial in the single-snapshot case, next we show that this phenomenon almost always happens if we consider the similarity of a single node v across two snapshots. If there is no change at all in the neighborhood of node v between the two snapshots, we expect the Sim value to be maximal. However the situation is identical to the bipartite graph case and Sim will be inversely proportional to the number of out-links.

3.1.3 Extended Jaccard Similarity Along Time

Our first definition of similarity is based on the extension of the Jaccard coefficient in a similar way XJaccard is defined in [41]. The Jaccard similarity of a page or host v across two snapshots t_0 and t_1 is defined by the overlap of its neighborhood in the two snapshots, $\Gamma^{(t_0)}(v)$ and $\Gamma^{(t_1)}(v)$ as

$$\mathsf{Jac}^{(t_0,t_1)}(v) = \frac{|\Gamma^{(t_0)}(v) \cap \Gamma^{(t_1)}(v)|}{|\Gamma^{(t_0)}(v) \cup \Gamma^{(t_1)}(v)|}$$

The *extended Jaccard coefficient, XJaccard* for length ℓ of a page or host is defined via the notion of the neighborhood $\Gamma_k^{(t)}(v)$ at distance exactly k as

$$\mathsf{XJac}_{\ell}^{(t_0,t_1)}(v) = \sum_{k=1}^{\ell} \frac{|\Gamma_k^{(t_0)}(v) \cap \Gamma_k^{(t_1)}(v)|}{|\Gamma_k^{(t_0)}(v) \cup \Gamma_k^{(t_1)}(v)|} \cdot c^k(1-c),$$

where c is a decay factor.

The XJac values can be approximated by the min-hash fingerprinting technique for Jaccard coefficients [15], as described in Algorithm 3 of [41]. The fingerprint generation algorithm has to be repeated for each graph snapshot, with the same set of independent random permutations.

We generate temporal features based on the XJac values for four length values $\ell = 1 \ldots 4$. We also repeat the computation on the transposed graph, i.e. replacing out-links $\Gamma^{(t)}(v)$ by in-links $I^{(t)}(v)$. As suggested in [41], we set the decay factor $c = 0.1$ as this is the value where, in their experiments, XJaccard yields best average quality for similarity prediction.

Similar to [61], we also calculate the mean and variance $\mathsf{XJac}^{(t_0,t_1)}{}_{\ell}(w)$ of the neighbors w for each node v. The following derived features are also calculated:

- similarity at path length $\ell = 2, 3, 4$ divided by similarity at path length $\ell - 1$, and the logarithm of these;
- logarithm of the minimum, maximum, and average of the similarity at path length $\ell = 2, 3, 4$ divided by the similarity at path length $\ell - 1$.

3.1.4 PSimRank Along Time

Next we define similarity over time based on PSimRank, a SimRank variant defined in [41] that can be applied similar to XJaccard in the previous section. As we saw in Section 3.1.2, SimRank is inappropriate for measuring linkage change in time. In the terminology of the previous subsection, the reason is that path fingerprints will be unlikely to meet in a large neighborhood and SimRank values will be low even if there is completely no change in time.

We solve the deficiency of SimRank by allowing the random walks to meet with higher probability when they are close to each other: a pair of random walks at vertices u', v' will advance to the same vertex (i.e., meet in one step) with probability of the Jaccard coefficient $\frac{|I(u') \cap I(v')|}{|I(u') \cup I(v')|}$ of their in-neighborhood $I(u')$ and $I(v')$.

The random walk procedure corresponding to PSimRank along with a fingerprint generation algorithm is defined in [41].

For the temporal version, we choose independent random permutations σ_ℓ on the hosts for each step ℓ. In step ℓ if the random walk from vertex u is at u', it will step to the in-neighbor with smallest index given by the permutation σ_ℓ in each graph snapshot.

Temporal features are derived from the PSimRank similarity measure very much the same way as for XJaccard, for four length values $\ell = 1 \dots 4$. We also repeat the computation on the transposed graph, i.e. replacing out-links $\Gamma^{(t)}(v)$ by in-links $I^{(t)}(v)$. As suggested in [41], we set the decay factor $c = 0.15$ as this is the value where, in their experiments, PSimRank yields best average quality for similarity prediction. Additionally, we calculate the mean and variance PSimRank(w) of the neighbors w for each node v and derived features as for XJaccard.

3.2 Content and its Change

The content of Web pages can be deployed in content classification either via statistical features such as entropy [59] or via term weight vectors [67, 31]. Some of the more complex features that we do not consider in this work include language modeling [3].

In this section we focus on capturing term-level changes over time. For each target site and crawl snapshot, we collect all the available HTML pages and represent the site as the bag-of-words union of all of their content. We tokenize content using the ICU library[3], remove stop words[4] and stem using Porter's method.

We treat the resulting term list as the virtual document for a given site at a point of time. As our vocabulary we use the most frequent 10,000 terms found in at least 10% and at most 50% of the virtual documents.

To measure the importance of each term i in a virtual document d at time snapshot T, we use the BM25 weighting [60]:

$$t_{i,d}^{(T)} = \mathrm{IDF}_i^{(T)} \cdot \frac{(k_1+1)\mathrm{tf}_{i,d}^{(T)}}{K + \mathrm{tf}_{i,d}^{(T)}}$$

where $\mathrm{tf}_{i,d}^{(T)}$ is the number of occurrences of term i in document d and $\mathrm{IDF}_i^{(T)}$ is the inverse document frequency (Robertson-Spärck Jones weight) for the term at time T. The length normalized constant K is specified as

$$k_1((1-b) + b \times \mathrm{dl}^{(T)}/\mathrm{avdl}^{(T)})$$

such that $dl^{(T)}$ and $avdl^{(T)}$ denote the virtual document length and the average length over all virtual documents at time T, respectively. Finally

$$\mathrm{IDF}^{(T)} = log\frac{N - n^{(T)} + 0.5}{n^{(T)} + 0.5}$$

[3] http://icu-project.org/

[4] http://www.lextek.com/manuals/onix/stopwords1.html

where N denotes the total number of virtual documents and $n^{(T)}$ is the number of virtual documents containing term i. Note that we keep N independent of T and hence if document d does not exist at T, we consider all $\mathrm{tf}_{i,d}^{(T)} = 0$.

By using the term vectors as above, we calculate the temporal content features described in [31] in the following five groups.

- **Ave:** Average BM25 score of term i over the $T_{\max}$ snapshots:

$$\mathrm{Ave}_{i,d} = \frac{1}{T_{\max}} \cdot \sum_{T=1}^{T_{\max}} t_{i,d}^{(T)}$$

- **AveDiff:** Mean difference between temporally successive term weight scores:

$$\mathrm{AveDiff}_{i,d} = \frac{1}{T_{\max} - 1} \cdot \sum_{T=1}^{T_{\max}-1} |t_{i,d}^{(T+1)} - t_{i,d}^{(T)}|$$

- **Dev:** Variance of term weight vectors at all time points:

$$\mathrm{Dev}_{i,d} = \frac{1}{T_{\max} - 1} \cdot \sum_{T=1}^{T_{\max}} (t_{i,d}^{(T)} - \mathrm{Ave}_{i,d})^2$$

- **DevDiff:** Variance of term weight vector differences of temporally successive virtual documents:

$$\mathrm{DevDiff}_{i,d} = \frac{1}{T_{\max} - 2} \cdot \sum_{T=1}^{T_{\max}-1} (|t_{i,d}^{(T+1)} - t_{i,d}^{(T)}| - \mathrm{AveDiff}_{i,d})^2$$

- **Decay:** Weighted sum of temporally successive term weight vectors with exponentially decaying weight. The base of the exponential function, the *decay rate* is denoted by λ. **Decay** is defined as follows:

$$\mathrm{Decay}_{i,d} = \sum_{T=1}^{T_{\max}} \lambda e^{\lambda(T_{\max}-T)} t_{i,d}^{(T)}$$

4 Classification Framework

For the purposes of our experiments we computed all the public Web Spam Challenge content and link features of [20]. We built a classifier ensemble by splitting features into related sets and for each we use a collection of classifiers that fit the data type and scale. These classifiers were then combined by ensemble selection. We used the classifier implementations of the machine learning toolkit Weka [65].

Ensemble selection is an overproduce and choose method allowing to use large collections of diverse classifiers [17]. Its advantages over previously published methods [16] include optimization to any performance metric and refinements to prevent overfitting, the latter being unarguably important when more

classifiers are available for selection. The motivation for using ensemble selection is that recently this particular ensemble method gained more attention thanks to the winners of KDD Cup 2009 [57]. In our experiments [38] ensemble selection performed significantly better than other classifier combination methods used for Web spam detection in the literature, such as log-odds based averaging [56] and bagging.

In the context of combining classifiers for Web classification, to our best knowledge, ensemble selection has not been applied yet. Previously, only simple methods that combine the predictions of SVM or decision tree classifiers through logistic regression or random forest have been used [28]. We believe that the ability to combine a large number of classifiers while preventing overfitting makes ensemble selection an ideal candidate for Web classification, since it allows us to use a large number of features and learn different aspects of the training data at the same time. Instead of tuning various parameters of different classifiers, we can concentrate on finding powerful features and selecting the main classifier models which we believe to be able to capture the differences between the classes to be distinguished.

We used the ensemble selection implementation of Weka [65] for performing the experiments. Weka's implementation supports the proven strategies to avoid overfitting such as model bagging, sort initialization and selection with replacement. We allow Weka to use all available models in the library for greedy sort initialization and use 5-fold embedded cross-validation during ensemble training and building. We set AUC as the target metric to optimize for and run 100 iterations of the hillclimbing algorithm.

We mention that we have to be careful with treating missing feature values. Since the temporal features are based on at least two snapshots, for a site that appears only in the last one, all temporal features have missing value. For classifiers that are unable to treat missing values we define default values depending on the type of the feature.

4.1 Learning Methods

We use the following models in our ensemble: bagged and boosted decision trees, logistic regression, naive Bayes and variants of random forests. For most classes of features we use all classifiers and let selection choose the best ones. The exception is static and temporal term vector based features where, due to the very large number of features, we may only use Random Forest and SVM. We train our models as follows.

Bagged LogitBoost: we do 10 iterations of bagging and vary the number of iterations from 2 to 64 in multiples of two for LogitBoost.

Decision Trees: we generate J48 decision trees by varying the splitting criterion, pruning options and use either Laplacian smoothing or no smoothing at all.

Bagged Cost-sensitive Decision Trees: we generate J48 decision trees with default parameters but vary the cost sensitivity for false positives in steps

of 10 from 10 to 300. We do the same number of iterations of bagging as for LogitBoost models.

Logistic Regression: we use a regularized model varying the ridge parameter between 10^{-8} to 10^{4} by factors of 10. We normalize features to have mean 0 and standard deviation 1.

Random Forests: we use FastRandomForest [39] instead of the native Weka implementation for faster computation. The forests have 250 trees and, as suggested in [14], the number of features considered at each split is $s/2$, s, $2s$, $4s$ and $8s$, where s is the square root of the total number of features available.

Naive Bayes: we allow Weka to model continuous features either as a single normal or with kernel estimation, or we let it discretize them with supervised discretization.

5 Results and Discussion

In this section we describe the various ensembles we built and measure their performance[5]. We compare feature sets by using the same learning methods described in Section 4 while varying the subset of features available for each of the classifier instances when training and combining these classifiers using ensemble selection. We also concentrate on the value of temporal information for Web spam detection. As our goal is to explore the computational cost vs. classification performance tradeoff, we will describe the resource needs for various features in detail in Section 6.

For training and testing we use the official Web Spam Challenge 2008 training and test sets [20]. As it can be seen in Table 1 these show considerable class imbalance which makes the classification problem harder. For DC2010 we also use the official training set as described in Table 5. For ClueWeb09 we used a 50% random split.

To make it easy to compare our results to previous results, we cite the Web Spam Challenge 2008 and Discovery Challenge 2010 winner's performance in the summary tables next. For ClueWeb09 the only previous evaluation is in terms of TREC retrieval performance [29] that we cannot directly compare here.

5.1 Content-only Ensemble

We build three different ensembles over the content-only features in order to assess performance by completely eliminating linkage information. The feature sets available for these ensembles are the following:

- (A) Public content [59, 22] features without any link based information. Features for the page with maximum PageRank in the host are not used to save the PageRank computation. Corpus precision, the fraction of words in a page that is corpuswise frequent and corpus recall, the fraction

[5]The exact classifier model specification files used for Weka and the data files used for the experiments are available upon request from the authors.

of corpuswise frequent terms in the page are not used either since they require global information from the corpus.

- (Aa) The tiniest feature set of 24 features from (A): query precision and query recall defined similar to corpus precision and recall but based on popular terms from a proprietary query log[6] instead of the entire corpus. A very strong feature set based on the intuition that spammers use terms that make up popular queries.
- (B) The full public content feature set [22], including features for the maximum PageRank page of the host.
- Feature set (B) plus a bag of words representation derived from the BM25 [60] term weighting scheme.

Table 6 presents the performance comparison of ensembles built using either of the above feature sets. The DC2010 and ClueWeb09 detailed results are in Table 8 and Table 9, respectively. Performance is given in AUC for all data sets.

Feature Set	Number of Features	UK2007	DC2010	ClueWeb09
Content (A)	74	0.859	0.757	0.829
Content (Aa)	24	0.841	0.726	0.635
Content (B)	96	0.879	0.799	0.827
BM25 + (B)	10096	**0.893**	**0.891**	**0.870**
Challenge best	-	0.852	0.830	-

Table 6: AUC value of spam ensembles built from content based features.

Surprisingly, with the small (Aa) feature set of only 24 features a performance only 1% worse than that of the Web Spam Challenge 2008 winner can be achieved who employed more sophisticated methods to get their result. By using all the available content based features without linkage information, we get roughly the same performance as the best which have been reported on our data set so far. However this achievement can be rather attributed to the better machine learning techniques used than the feature set itself since the features used for this particular measurement were already publicly accessible at the time of the Web Spam Challenge 2008.

As it can be seen in Table 6 relative performance of content based features over different corpora varies a lot. In case of DC2010 and ClueWeb09 the small (Aa) feature set achieves much worse result than the largest feature set having best performance for all data sets. The fact that the content (A, Aa, B) and link (Table 7) performances are always better for UK2007 might be explained

[6] A summary is available as part of our data release at `https://dms.sztaki.hu/sites/dms.sztaki.hu/files/download/2013/enpt-queries.txt.gz`.

by the fact that the UK2007 training and testing sets were produced by random sampling without considering domain boundaries. Hence in a large domain with many subdomains, part of the hosts belong to the training and part to the testing set with very similar distribution. This advantage disappears for the BM25 features.

5.2 Full Ensemble

Feature Set	Number of Features	UK2007	DC2010	ClueWeb09
Public link-based [7]	177	0.759	0.587	0.806
All combined	10 273	**0.902**	0.885	0.876

Table 7: Performance of ensembles built on link based and all features.

Results of the ensemble incorporating all the previous classifiers is seen in Table 7. The DC2010 detailed results are in Table 8. Overall, we observe that BM25 is a very strong feature set that may even be used itself for a lightweight classifier. On the other hand, link features add little to quality and the gains apparently diminish for DC2010, likely due to the fact that the same domain and IP address is not split between training and testing.

The best Web Spam Challenge 2008 participant [44] reaches an AUC of 0.85 while for DC2010, the best spam classification AUC of [58] is 0.83. We outperform these results by a large margin.

For DC2010 we also show detailed performance for nine attributes in Table 8, averaged in three groups: spam, genre and quality (as in Table 5). Findings are similar: with BM25 domination, part or all of the content features slightly increase the performance. Results for the quality attributes and in particular for trust are very low. Classification for these aspects remains a challenging task for the future.

For ClueWeb09 detailed performance for selected ODP categories can be seen in Table 9. Identically to DC2010 results BM25 features provide the best classification performance. However, combinations with other feature sets yield gains only for spam classification. For the ODP classification tasks linkage information does not help in general: the content based feature set has roughly the same performance with or without page-level linkage information, and combining with the link based feature set does not improve performance notably in most labeling tasks.

5.3 Temporal Link Ensembles

First, we compare the temporal link features proposed in Section 3.1 with those published earlier [61]. Then, we build ensembles that combine the temporal with

Feature Set	spam	genre **average**	quality **average**	**average**
Public link-based [7]	0.655	0.614	0.519	0.587
Content (A)	0.757	0.713	0.540	0.660
Content (Aa)	0.726	0.662	0.558	0.634
Content (B)	0.799	0.735	0.512	0.668
BM25	**0.876**	**0.805**	**0.584**	**0.739**
Public link-based + (B)	0.812	0.731	0.518	0.669
BM25 + (A)	0.872	**0.816**	0.580	**0.754**
BM25 + (B)	**0.891**	0.810	**0.612**	0.744
All combined	0.885	0.813	0.553	0.734

Table 8: Performance over the DC2010 labels in terms of AUC.

Feature Set	spam	Arts	Business	Computers	Recreation	Science	Society	Sports	**ODP average**
Link [7]	.806	.569	.593	.591	.532	.624	.540	.504	.595
Content (A)	.829	.676	.726	.632	.669	.720	.639	.673	.695
Content (Aa)	.635	.508	.524	.554	.487	.558	.502	.522	.536
Content (B)	.827	.673	.727	.634	.670	.720	.629	.674	.694
BM25	.845	**.913**	**.890**	**.931**	**.907**	**.883**	**.915**	**.959**	**.914**
Link + (B)	.848	.675	.731	.646	.669	.727	.631	.669	.699
BM25 + (A)	.871	.895	.881	.896	.879	.851	.904	.935	.892
BM25 + (B)	.869	.895	.881	.898	.892	.850	.906	.934	.894
All combined	**.876**	.896	.883	.898	.892	.852	.905	.936	.895

Table 9: Performance over the ClueWeb09 labels in terms of AUC.

the public link-based features described by [7]. The results are summarized in Table 10. Note that all experiments in this section and Section 5.4 were carried out on the WEBSPAM-UK2007 data set.

As these measurements show, our proposed graph similarity based features successfully extend the growth and death rate based ones by achieving higher accuracy, improving AUC by 1.3%. However, by adding temporal to static link-based features we get only marginally better ensemble performance.

To rank the link-based feature sets by their contribution in the ensemble, we build classifier models on the three separate feature subsets (public link-based, growth/death rate based and graph similarity based features, respectively) and let ensemble selection combine them. This restricted combination results in a slightly worse AUC of 0.762. By calculating the total weight contribution,

Section	Feature Set	No. of Features	AUC
3.1.1	Growth/death rates	29	0.617
3.1.3-4	XJaccard + PSimRank	63	0.625
	Public link-based [7]	176	0.765
3.1.1	Public + growth/death rates	205	0.758
3.1.3-4	Public + XJaccard + PSimRank	239	**0.769**
	All link-based	268	0.765
	WSC 2008 Winner	-	0.852

Table 10: Performance of ensembles built on link-based features.

we get the following ranked list (weight contribution showed in parenthesis): public link-based (60.8%), graph similarity based (21.5%), growth/death rate based (17.7%). This ranking also supports the findings presented in Table 10 that graph similarity based temporal link-based features should be combined with public link-based features if temporal link-based features are used.

To separate the effect of ensemble selection on the performance of temporal link-based feature sets we repeat the experiments with bagged cost-sensitive decision trees only, a model reported to be effective for web spam classification [59]. The results for these experiments are shown in Table 11.

Section	Feature Set	No. of Features	AUC
3.1.1	Growth/death rates	29	0.605
3.1.3	XJaccard	42	0.626
3.1.4	PSimRank	21	0.593
3.1.3-4	XJaccard + PSimRank	63	0.610
	Public link-based [7]	176	**0.731**
3.1.1	Public + growth/death rates	205	0.696
3.1.3-4	Public + XJaccard + PSimRank	239	*0.710*
	All link-based	268	0.707
	WSC 2008 Winner	-	0.852

Table 11: Performance of bagged cost-sensitive decision trees trained on link-based features.

As it can be seen in Table 11, when using bagged cost-sensitive decision trees, our proposed temporal link-based similarity features achieve 3.5% better performance than the growth/death rate based features published earlier.

When comparing results in Table 11 and in Table 10 we can see that ensemble selection i) significantly improves accuracy (as expected) and ii) diminishes the performance advantage achievable by the proposed temporal link-based features over the previously published ones.

As evident from Table 11, the proposed PSimRank based temporal features perform roughly the same as the growth and death rate based ones while the XJaccard based temporal features perform slightly better.

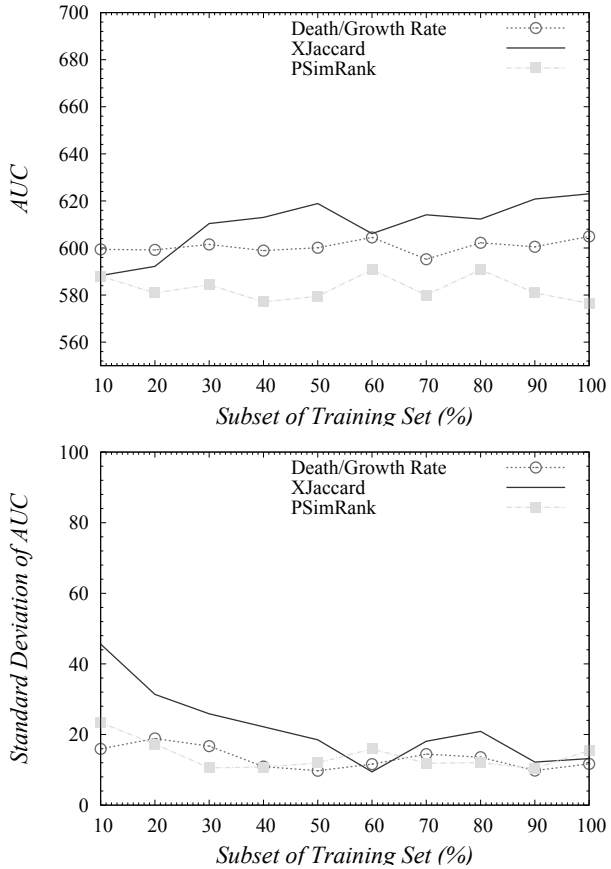

Figure 2: Sensitivity of temporal link-based features. **Top:** AUC values averaged across 10 measurements. **Bottom:** standard deviations of AUC for different training set sizes.

Next we perform sensitivity analysis of the temporal link-based features by using bagged cost-sensitive decision trees. We build 10 different random training samples for each of the possible fractions 10%, 20%, ..., 100% of all available labels. In Fig. 2 we can see that the growth/death rate based features as well as the PSimRank based features are not sensitive to training set size while the

XJaccard based ones are. That is, even though XJaccard is better in terms of performance than the other two feature sets considered it is more sensitive to the amount of training data used as well.

5.4 Temporal Content Ensembles

We build ensembles based on the temporal content features described in Section 3.2 and their combination themselves, with the static BM25 features, and with the content-based features of [59]. The performance comparison of temporal content-based ensembles is presented in Table 12.

Feature Set	**No. of Features**	**AUC**
Static BM25	10,000	0.736
Ave	10,000	0.749
AveDiff	10,000	0.737
Dev	10,000	0.767
DevDiff	10,000	0.752
Decay	10,000	0.709
Temporal combined	50,000	0.782
Temporal combined + BM25	60,000	**0.789**
Public content-based [59] + temporal	50,096	0.901
All combined	60,096	**0.902**

Table 12: Performance of ensembles built on temporal content-based features.

By combining all the content and link-based features, both temporal and static ones, we train an ensemble which incorporates all the previous classifiers. This combination resulted in an AUC of 0.908 meaning no significant improvement can be achieved with link-based features over the content-based ensemble.

6 Computational Resources

For the experiments we used a 45-node Hadoop cluster of dual core machines with 4GB RAM each as well as multi-core machines with over 40GB RAM. Over this architecture we were able to compute all features, some of which would require excessive resources either when used by a smaller archive or if the collection is larger or if fast classification is required for newly discovered sites during crawl time. Some of the most resource bound features involve the multi-step neighborhood in the page level graph that already requires approximation techniques for WEBSPAM-UK2007 [22].

We describe the computational requirements of the features by distinguishing update and batch processing. For batch processing an entire collection is analyzed at once, a procedure that is probably performed only for reasons of research. Update is probably the typical operation for a search engine. For an

Feature Set	Step	Hours	Configuration
Content (A) + BM25	Parsing	36	45 dual core Pentium-D 3.0GHz machines, 4GB RAM, Hadoop 0.21
	Feature generation	36	
	Selection of labeled pages	3	
Link	PageRank	10	5 eight-core Xeon 1.6GHz machines, 40+GB RAM
	Neighborhood	4	
	Local features	1	

Table 13: Processing times and cluster configurations for feature sets over ClueWeb09.

Internet Archive, update is also advantageous as long as it allows fast reaction to sample, classify and block spam from a yet unknown site.

6.1 Batch Processing

The first expensive step involves parsing to create terms and links. The time requirement scales linearly with the number of pages. Since apparently a few hundred page sample of each host suffices for feature generation, the running time is also linear in the number of hosts. For a very large collection such as ClueWeb09, distributed processing may be necessary. Over 45 dual core Pentium-D 3.0GHz machines running Hadoop 0.21, we parsed the uncompressed 9.5TB English part of ClueWeb09 in 36 hours. Additional tasks such as term counting, BM25 or content feature generation fits within the same time frame. If features are generated only a small labeled part of the data, it took us 3 hours to select the appropriate documents and additional processing time was negligible. Processing times are summarized in Table 13.

Host level aggregation allows us to proceed with a much smaller size data. However for aggregation we need to store a large number of partial feature values for all hosts unless we sort the entire collection by host, again by external memory or Map-Reduce sort.

After aggregation, host level features are inexpensive to compute. The following features however remain expensive:

- Page level PageRank. Note that this is required for all content features involving the maximum PageRank page of the host.
- Page level features involving multi-step neighborhood such as neighborhood size at distance k as well as graph similarity.

In order to be able to process graphs of ClueWeb09 scale (4.7 billion nodes and 17 billion edges), we implemented message passing C++ codes. Over a total 30 cores of six Xeon 1.6GHz machines, each with at least 40GB RAM, one PageRank and one Bit Propagation iteration both took approximately one hour while all other, local features completed within one hour.

Training the classifier for a few 100,000 sites can be completed within a day on a single CPU on a commodity machine with 4-16GB RAM; here costs

Configuration	Number of Hosts	Feature Sets	Example	Expected Accuracy	Computation
Small 1-2 machines	10,000	Content (A) BM25	subset of UK2007	0.80-0.87	Non-distributed
Medium 3-10 machines	100,000	Content (A) BM25, link	DC2010	0.87-0.90	MapReduce and Disk-based e.g. GraphChi
Large 10+ machines	1,000,000	Content (B) BM25, link	ClueWeb09	0.9+	MapReduce and Pregel

Table 14: Sample configurations for Web spam filtering in practice.

strongly depend on the classifier implementation. Our entire classifier ensemble for the labeled WEBSPAM-UK2007 hosts took a few hours to train.

6.2 Incremental Processing

As preprocessing and host level aggregation is linear in the number of hosts, this reduces to a small job for an update. This is especially true if we are able to split the update by sets of hosts; in this case we may even trivially parallelize the procedure.

The only nontrivial content based information is related to document frequencies: both the inverse document frequency term of BM25 [60] and the corpus precision and recall dictionaries may in theory be fully updated when new data is added. We may however approximate by the existing values under the assumption that a small update batch will not affect these values greatly. From time to time however all features beyond (Aa) need a global recomputation step.

The link structure is however nontrivial to update. While incremental algorithms exist to create the graph and to update PageRank type features [32, 33, 53], these algorithms are rather complex and their resource requirements are definitely beyond the scale of a small incremental data.

Incremental processing may have the assumption that no new labels are given, since labeling a few thousand hosts takes time comparable to batch process hundreds of thousands of them. Given the trained classifier, a new site can be classified in seconds right after its feature set is computed.

7 Conclusions

With the illustration over the 100,000 host WEBSPAM-UK2007, the half billion page ClueWeb09, and the 190,000 host DC2010 data sets, we have investigated the tradeoff between feature generation and spam classification accuracy. We observe that more features achieve better performance, however, when combining them with the public link based feature set we get only marginal performance gain. By using the WEBSPAM-UK2007 data along with seven previous monthly snapshots of the `.uk` domain, we have presented a survey of temporal

features for Web spam classification. We investigated the performance of link, content and temporal[7] Web spam features with ensemble selection. As practical message, we may conclude that, as seen in Table 14, single machines may compute content and BM25 features for a few 10,000 hosts only. Link features need additional resources and either compressed, disk based or, in the largest configuration, Pregel-like distributed infrastructures.

We proposed graph similarity based temporal features which aim to capture the nature of linkage change of the neighborhoods of hosts. We have shown how to compute these features efficiently on large graphs using a Monte Carlo method. Our features achieve better performance than previously published methods, however, when combining them with the public link-based feature set we get only marginal performance gain.

By our experiments it has turned out that the appropriate choice of the machine learning techniques is probably more important than devising new complex features. We have managed to compile a minimal feature set that can be computed incrementally very quickly to allow to intercept spam at crawl time based on a sample of a new Web site. Sample configurations for Web spam filtering are summarized in Table 14.

Our results open the possibility for spam filtering practice in Internet archives who are mainly concerned about their resource waste and would require fast reacting filters. BM25 based models are suitable even for filtering at crawl time.

Some technologies remain open to be explored. For example, unlike expected, the ECML/PKDD Discovery Challenge 2010 participants did not deploy cross-lingual technologies for handling languages other than English. Some ideas worth exploring include the use of dictionaries to transfer a bag of words based model and the normalization of content features across languages to strengthen the language independence of the content features. The natural language processing based features were not used either, that may help in particular with the challenging quality attributes.

Acknowledgment

To Sebastiano Vigna, Paolo Boldi and Massimo Santini for providing us with the UbiCrawler crawls [10, 11]. In addition to them, also to Ilaria Bordino, Carlos Castillo and Debora Donato for discussions on the WEBSPAM-UK data sets [12].

To the large team of organizers and assessors for the complex labeling process of the DC2010 data set.

[7]The temporal feature data used in our research is available at: `https://datamining.sztaki.hu/en/download/web-spam-resources`

References

[1] J. Abernethy, O. Chapelle, and C. Castillo. WITCH: A New Approach to Web Spam Detection. In *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2008.

[2] L. D. Artem Sokolov, Tanguy Urvoy and O. Ricard. Madspam consortium at the ecml/pkdd discovery challenge 2010. In *Proceedings of the ECML/PKDD 2010 Discovery Challenge*, 2010.

[3] J. Attenberg and T. Suel. Cleaning search results using term distance features. In *Proceedings of the 4th international workshop on Adversarial information retrieval on the web*, pages 21–24. ACM New York, NY, USA, 2008.

[4] Z. Bar-Yossef, A. Z. Broder, R. Kumar, and A. Tomkins. Sic transit gloria telae: Towards an understanding of the web's decay. In *Proceedings of the 13th World Wide Web Conference (WWW)*, pages 328–337. ACM Press, 2004.

[5] Z. Bar-Yossef, I. Keidar, and U. Schonfeld. Do not crawl in the dust: different urls with similar text. *ACM Transactions on the Web (TWEB)*, 3(1):1–31, 2009.

[6] S. Barton. Mignify, a big data refinery built on hbase. In *HBASE CON*, 2012.

[7] L. Becchetti, C. Castillo, D. Donato, S. Leonardi, and R. Baeza-Yates. Link-based characterization and detection of web spam. In *Proceedings of the 2nd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2006.

[8] A. A. Benczúr, M. Erdélyi, J. Masanés, and D. Siklósi. Web spam challenge proposal for filtering in archives. In *AIRWeb '09: Proceedings of the 5th international workshop on Adversarial information retrieval on the web*. ACM Press, 2009.

[9] A. A. Benczúr, D. Siklósi, J. Szabó, I. Bíró, Z. Fekete, M. Kurucz, A. Pereszlényi, S. Rácz, and A. Szabó. Web spam: a survey with vision for the archivist. In *Proc. International Web Archiving Workshop*, 2008.

[10] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):721–726, 2004.

[11] P. Boldi, M. Santini, and S. Vigna. A Large Time Aware Web Graph. *SIGIR Forum*, 42, 2008.

[12] I. Bordino, P. Boldi, D. Donato, M. Santini, and S. Vigna. Temporal evolution of the uk web. In *Workshop on Analysis of Dynamic Networks (ICDM-ADN'08)*, 2008.

[13] I. Bordino, D. Donato, and R. Baeza-Yates. Coniunge et impera: Multiple-graph mining for query-log analysis. In *Machine Learning and Knowledge Discovery in Databases*, pages 168–183. Springer, 2010.

[14] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[15] A. Z. Broder. On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences (SEQUENCES'97)*, pages 21–29, 1997.

[16] R. Caruana, A. Munson, and A. Niculescu-Mizil. Getting the most out of ensemble selection. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 828–833, Washington, DC, USA, 2006. IEEE Computer Society.

[17] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 18, New York, NY, USA, 2004. ACM.

[18] C. Castillo, K. Chellapilla, and L. Denoyer. Web spam challenge 2008. In *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2008.

[19] C. Castillo and B. Davison. *Adversarial web search*, volume 4. Now Publishers Inc, 2011.

[20] C. Castillo, D. Donato, L. Becchetti, P. Boldi, S. Leonardi, M. Santini, and S. Vigna. A reference collection for web spam. *SIGIR Forum*, 40(2):11–24, December 2006.

[21] C. Castillo, D. Donato, A. Gionis, V. Murdock, and F. Silvestri. Know your neighbors: Web spam detection using the web topology. Technical report, DELIS – Dynamically Evolving, Large-Scale Information Systems, 2006.

[22] C. Castillo, D. Donato, A. Gionis, V. Murdock, and F. Silvestri. Know your neighbors: web spam detection using the web topology. *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 423–430, 2007.

[23] N. Chawla, N. Japkowicz, and A. Kotcz. Editorial: special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter*, 6(1):1–6, 2004.

[24] C. Chekuri, M. H. Goldwasser, P. Raghavan, and E. Upfal. Web search using automatic classification. In *Proceedings of the 6th International World Wide Web Conference (WWW)*, San Jose, USA, 1997.

[25] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *The VLDB Journal*, pages 200–209, 2000.

[26] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of the International Conference on Management of Data*, pages 117–128, 2000.

[27] E. Convey. Porn sneaks way back on web. *The Boston Herald*, May 1996.

[28] G. Cormack. Content-based Web Spam Detection. In *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2007.

[29] G. Cormack, M. Smucker, and C. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Information retrieval*, 14(5):441–465, 2011.

[30] K. Csalogány, A. Benczúr, D. Siklósi, and L. Lukács. Semi-Supervised Learning: A Comparative Study for Web Spam and Telephone User Churn. In *Graph Labeling Workshop in conjunction with ECML/PKDD 2007*, 2007.

[31] N. Dai, B. D. Davison, and X. Qi. Looking into the past to better classify web spam. In *AIRWeb '09: Proceedings of the 5th international workshop on Adversarial information retrieval on the web*. ACM Press, 2009.

[32] P. Desikan, N. Pathak, J. Srivastava, and V. Kumar. Incremental page rank computation on evolving graphs. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1094–1095, New York, NY, USA, 2005. ACM.

[33] P. K. Desikan, N. Pathak, J. Srivastava, and V. Kumar. Divide and conquer approach for efficient pagerank computation. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 233–240, New York, NY, USA, 2006. ACM.

[34] A. Dong, Y. Chang, Z. Zheng, G. Mishne, J. Bai, K. Buchner, R. Zhang, C. Liao, and F. Diaz. Towards recency ranking in web search. In *Proc. WSDM*, 2010.

[35] N. Eiron, K. S. McCurley, and J. A. Tomlin. Ranking the web frontier. In *Proceedings of the 13th International World Wide Web Conference (WWW)*, pages 309–318, New York, NY, USA, 2004. ACM Press.

[36] M. Erdélyi and A. A. Benczúr. Temporal analysis for web spam detection: An overview. In *1st International Temporal Web Analytics Workshop (TWAW) in conjunction with the 20th International World Wide Web Conference in Hyderabad, India.* CEUR Workshop Proceedings, 2011.

[37] M. Erdélyi, A. A. Benczúr, J. Masanés, and D. Siklósi. Web spam filtering in internet archives. In *AIRWeb '09: Proceedings of the 5th international workshop on Adversarial information retrieval on the web*. ACM Press, 2009.

[38] M. Erdélyi, A. Garzó, and A. A. Benczúr. Web spam classification: a few features worth more. In *Joint WICOW/AIRWeb Workshop on Web Quality (WebQuality 2011) In conjunction with the 20th International World Wide Web Conference in Hyderabad, India.* ACM Press, 2011.

[39] FastRandomForest. Re-implementation of the random forest classifier for the weka environment. `http://code.google.com/p/fast-random-forest/`.

[40] D. Fetterly and Z. Gyöngyi. Fifth international workshop on adversarial information retrieval on the web (AIRWeb 2009). 2009.

[41] D. Fogaras and B. Rácz. Scaling link-based similarity search. In *Proceedings of the 14th World Wide Web Conference (WWW)*, pages 641–650, Chiba, Japan, 2005.

[42] J. Fogarty, R. S. Baker, and S. E. Hudson. Case studies in the use of roc curve analysis for sensor-based estimates in human computer interaction. In *Proceedings of Graphics Interface 2005*, GI '05, pages 129–136, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005. Canadian Human-Computer Communications Society.

[43] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *Annals of statistics*, pages 337–374, 2000.

[44] G. Geng, X. Jin, and C. Wang. CASIA at WSC2008. In *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2008.

[45] X.-C. Z. Guang-Gang Geng, Xiao-Bo Jin and D. Zhang. Evaluating web content quality via multi-scale features. In *Proceedings of the ECML/PKDD 2010 Discovery Challenge*, 2010.

[46] Z. Gyöngyi and H. Garcia-Molina. Spam: It's not just for inboxes anymore. *IEEE Computer Magazine*, 38(10):28–34, October 2005.

[47] Z. Gyöngyi and H. Garcia-Molina. Web spam taxonomy. In *Proceedings of the 1st International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, Chiba, Japan, 2005.

[48] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with TrustRank. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 576–587, Toronto, Canada, 2004.

[49] M. R. Henzinger, R. Motwani, and C. Silverstein. Challenges in web search engines. *SIGIR Forum*, 36(2):11–22, 2002.

[50] A. Hotho, D. Benz, R. Jäschke, and B. Krause, editors. *Proceedings of the ECML/PKDD Discovery Challenge*. 2008.

[51] G. Jeh and J. Widom. SimRank: A measure of structural-context similarity. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 538–543, 2002.

[52] Y. joo Chung, M. Toyoda, and M. Kitsuregawa. A study of web spam evolution using a time series of web snapshots. In *AIRWeb '09: Proceedings of the 5th international workshop on Adversarial information retrieval on the web*. ACM Press, 2009.

[53] C. Kohlschütter, P. A. Chirita, and W. Nejdl. Efficient parallel computation of pagerank, 2007.

[54] Z. Kou and W. W. Cohen. Stacked graphical models for efficient inference in markov random fields. In *SDM 07*, 2007.

[55] Y. Lin, H. Sundaram, Y. Chi, J. Tatemura, and B. Tseng. Splog detection using content, time and link structures. In *2007 IEEE International Conference on Multimedia and Expo*, pages 2030–2033, 2007.

[56] T. Lynam, G. Cormack, and D. Cheriton. On-line spam filter fusion. *Proc. of the 29th international ACM SIGIR conference on Research and development in information retrieval*, pages 123–130, 2006.

[57] A. Niculescu-Mizil, C. Perlich, G. Swirszcz, V. Sindhwani, Y. Liu, P. Melville, D. Wang, J. Xiao, J. Hu, M. Singh, et al. Winning the KDD Cup Orange Challenge with Ensemble Selection. In *KDD Cup and Workshop in conjunction with KDD 2009*, 2009.

[58] V. Nikulin. Web-mining with wilcoxon-based feature selection, ensembling and multiple binary classifiers. In *Proceedings of the ECML/PKDD 2010 Discovery Challenge*, 2010.

[59] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *Proceedings of the 15th International World Wide Web Conference (WWW)*, pages 83–92, Edinburgh, Scotland, 2006.

[60] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *In Proceedings of SIGIR'94*, pages 232–241. Springer-Verlag, 1994.

[61] G. Shen, B. Gao, T. Liu, G. Feng, S. Song, and H. Li. Detecting link spam using temporal information. In *ICDM'06.*, pages 1049–1053, 2006.

[62] D. Siklósi, B. Daróczy, and A. Benczúr. Content-based trust and bias classification via biclustering. In *Proceedings of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality*, pages 41–47. ACM, 2012.

[63] A. Singhal. Challenges in running a commercial search engine. In *IBM Search and Collaboration Seminar 2004*. IBM Haifa Labs, 2004.

[64] S. Webb, J. Caverlee, and C. Pu. Predicting web spam with HTTP session information. In *Proceeding of the 17th ACM conference on Information and knowledge management*, pages 339–348. ACM, 2008.

[65] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition, June 2005.

[66] B. Wu, V. Goel, and B. D. Davison. Topical TrustRank: Using topicality to combat web spam. In *Proceedings of the 15th International World Wide Web Conference (WWW)*, Edinburgh, Scotland, 2006.

[67] B. Zhou, J. Pei, and Z. Tang. A spamicity approach to web spam detection. In *Proceedings of the 2008 SIAM International Conference on Data Mining (SDM'08)*, pages 277–288. Citeseer, 2008.

Temporal Wikipedia search by edits and linkage

Julianna Göbölös-Szabó András A. Benczúr
Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SZTAKI)
{gszj, benczur}@ilab.sztaki.hu

ABSTRACT

In this paper we exploit the connectivity structure of *edits* in Wikipedia to identify recent events that happened at a given time via identifying bursty changes in linked articles around a specified date. Our key results include algorithms for node relevance ranking in temporal subgraph and neighborhood selection based on measurements for structural changes in time over the Wikipedia link graph. We measure our algorithms over manually annotated queries with relevant events in September and October 2011; we make the assessment publicly available. While our methods were tested over clean Wikipedia metadata, we believe the methods are applicable to general temporal Web collections as well.

1. INTRODUCTION

Considering a chain of events, we are often interested in the causes and effects, naturally represented by citations and links. If we want to understand what and why did happen, what other stories had influences on the event, it is worth discovering connected articles. The problem is even more interesting if we want to know how a story evolved in time. In this case we also need the information about the time of appearance of pages and links, and this can help understanding the temporal causality of the analyzed event.

In this paper we develop methods to automatically discover temporal events along important connections. For our experiments we selected Wikipedia as a clean corpus where measurements are not biased for example by date identification, yet the methods can directly be applied for any hyperlinked collection. Wikipedia is certainly the most used and best-known online encyclopedia and knowledge-base of the past decade. Almost every action or event, be it tiny or slightly remarkable, immediately appears in blog posts, news articles or sometimes even in Wikipedia articles.

Certainly not all Wikipedia modifications are triggered by headline news. People contribute to pages for various reasons. Sometimes a mistake is found and gets corrected or the editor has a special field of interest without a satisfying of coverage so she starts to add new pieces of information to the encyclopedia. In this case we expect isolated edits of a low number of editors and hence less accumulated change in the neighborhood. In addition, these pages, even being just created, will link old, stable pages and will collect incoming links with moderate speed.

This work was supported in part by the EC FET Open project "New tools and algorithms for directed network analysis" (NADINE No 288956), by the EU FP7 Project LAWA—Longitudinal Analytics of Web Archive Data, and by the grant OTKA NK 105645. The work of the first author reported in the paper has been developed in the framework of the project "Talent care and cultivation in the scientific workshops of BME" project. This project is supported by the grant TÁMOP–4.2.2.B-10/1–2010-0009.

TAIA'13 August 1, 2013, Dublin, Ireland.

.

In our experiments we use three monthly snapshots of Wikipedia from September to November 2011. We selected 20 queries for September and 15 for October and manually assessed the articles for relevant events at that time. We made the queries and the set of relevant articles public[1].

Our results complement recent results on temporal information retrieval where the main goal was to correctly date the events mentioned in Web pages. In Wikipedia we may rely on exact metadata to date addition and deletion and our main goal is to distinguish bursty, time relevant modifications from sporadic edits of past events and general, time independent information.

For ranking we use both the link structure and the content. The user can specify a query and should get a "temporally changing" subgraph of relevant articles. First we try to find the relevant articles respective to the query by a text search engine. As the next step, based on these articles, we try to find those nodes that have not only important changes according to the definition above but also their content is related to the original query. In a recursive definition reminiscent of PageRank [12] and HITS [8], we will consider the change of a page relevant if relevant changes can be observed in the neighborhood of the page as well. Finally we retrieve and present the top ranking articles and their linkage.

2. RANKING BY CHANGE AND LINKAGE

In our temporal information retrieval task, the user specifies a broad topic (e.g. Arab spring) and a date. Relevant documents should describe events that happened around the specific date involving the broad topic in question. A query with sample relevant Wikipedia articles is in Table 1.

Our ranking model combines text relevance with scores for change at the specified date that we boost by bursty changes of interlinked articles. First we identify a seed set scored by classical ranking techniques, e.g. Okapi BM25.We extend this seed set by neighboring articles that changed. These steps yield a candidate subgraph that is small enough to run subgraph scoring at query time, yet sufficiently large to contain most information relevant to the user query.

2.1 Measure of change in time

In order to discover recent events and trends, we consider changes both in linkage and content. We expect pages related to a certain event increase in content as well as connectivity of both in and out-edges after the specific event. Consequently, we measure change

As illustrated in Table 1, we measure change as the sum of the change of the logarithm of the in and out-degree as well as the absolute difference between the number of words in the article between two fixed dates t_1 and t_2 as

$$change(u) = |\log \frac{deg_{in,t_1}(u)}{deg_{in,t_2}(u)}| + |\log \frac{deg_{out,t_1}(u)}{deg_{out,t_2}(u)}| + |\log \frac{words_{t_1}(u)}{words_{t_2}(u)}|.$$

[1] `http://dms.sztaki.hu/en/download/wimmut-searching-and-navigating-wikipedia`

		Sep-Oct	Oct-Nov	Sep-Nov
Muammar Gaddafi	content	0.044	0.18	0.23
	inlink	0.55	0.12	0.68
	outlink	0.033	0.04	0.074
Death of Muammar Gaddafi	content	0	7.71	7.71
	inlink	0	4.21	4.21
	outlink	0	4.64	4.64
Battle of Sirte (2011)	content	7.78	0.79	8.56
	inlink	4.78	0.21	4.99
	outlink	4.9	0.14	5.06

Table 1: Change of articles related to Muammar Gaddafi.

2.2 Expanding the seed set

Seed expansion requires a score over the nodes that measure their relevance and freshness. In a naive solution one would specify a given number of steps and consider the entire neighborhood in this distance. As it turns out, even the one-step neighborhood is too large and hence we have to score candidate neighbors v. We use the following formula:

$$\text{score}(v) = \max_{u \in \text{ seed}} \text{BM25}(u) + \text{change}(u) + \text{change}(v). \quad (1)$$

2.3 Scores for change and relevance

We take a convex combination of the IR and change scores by a parameter α. Before combination, we transform both IR and change scores into $[0, 1]$. IR scores are normalized by the maximum while change scores are saturated by using parameter T in order to avoid extreme large values of change. The final formula becomes

$$p(u) = \alpha \cdot \frac{\text{IR(u)}}{\text{maxIR}} + (1-\alpha) \cdot \frac{\text{change}(u)}{(\text{change}(u) + T)}. \quad (2)$$

The above score forms a class of baseline ranking schemes depending on the parameters. While the dependence on T turned out to be relative low, the values of α balance between two extremities. Case $\alpha = 1$ returns the text relevance score and case $\alpha = 0$ takes only the amount of change into account. Note that even in this case, text relevance scores are involved in the seed set and the expansion process.

2.4 Personalized PageRank, random walks and electric networks

Our first algorithm scores graph nodes by PageRank [12] personalized on the IR score. In [6] an electric network based method is presented to select a subgraph connecting a set of nodes; we show that this result is a special case of our personalized PageRank method. We briefly review some useful properties of personalized PageRank from e.g. [13] and their connection to the electric network formulation of [6]. Let $\text{PPR}_p(v)$ denote the personalized PageRank of vertex $v \in V$ where $p = p(u) \in \mathbb{R}^{|V|}$ is the personalization vector. Then personalized PageRank is the solution of the system of equations

$$\text{PPR}_p(v) = (1-c) \sum_{uv \in E} \text{PPR}_p(u) \frac{w(uv)}{w(u)} + c \cdot p(v), \quad (3)$$

where $w(uv)$ denotes the edge weight normalized so that the total weight of out-edges from u is 1. PageRank is equal to the probability that a random walk of length drawn from a geometric distribution terminates at the given node:

$$\text{PPR}_p(v) = \sum_{k=0}^{\infty} c(1-c)^k \sum_{v_0, v_1, \ldots, v_k = v} p(v_0) \cdot w(v_0 v_1) \cdots w(v_{k-1} v_k). \quad (4)$$

Next we consider special personalization vectors that apply for a single node only. With an abuse of notation, PPR_u will denote personalization to a vector p with $p(u) = 1$ and 0 otherwise. For such personalization vectors, the system of equations is equivalent to

$$PPR_u(x) = (1-c) \sum_{uv \in E} PPR_v(x) \cdot w(uv) + c \cdot p(v). \quad (5)$$

The electric network formulation of [6] uses the equation

$$V(u) = \sum_v V(v) \cdot w(uv) \; \forall u \neq s, t \quad (6)$$

with boundary conditions $V(s) = 1$ and $V(t) = 0$, see [6] for details. Note that equations (5) and (6) have the same form with V_u corresponding to PPR_u except for the additive term $c \cdot p(v)$. These terms in equations (3) and (5) correspond to a universal sink S which can be added to the electric network with edge weight $w(v, S) = p(v)$. Universal sinks are also introduced in [6] with the difference that their method immediately taxes the large degree nodes while the uniform additive term in (5) taxes equally, regardless of the degree.

2.5 Personalized HITS

Our next class of graph ranking procedures are based on HITS [8]. HITS is known to be vulnerable to topic drift, the preference of nodes in a large but irrelevant clique or dense region in the neighborhood of the original topic. A few papers consider the question of personalizing HITS to reduce topic drift [2, 7] but these algorithms are rather complex.

We give a simple personalization to HITS by using a "supersource". We can think of the supersource as a new node of the graph which is connected with each node of the original graph, and the weight of an edge corresponds to the importance of the respective node in the personalization, with weight 0 also allowed. The supersource distributes a fixed amount of score in each iteration split proportional to the personalization distribution. At the end of each iteration, we normalize the authority and hub vectors, so the maximal element in the vector is 1. With the notation of a as the vector of authorities, h as the vector of hubs, c as the importance of the supersource and p as the personalization vector, we have

$$\hat{a}(v) = \sum_{uv \in E} w(uv) \cdot h(u) + c \cdot p(v), \quad a = \hat{a}/\|a\|_\infty; \quad (7)$$

$$\hat{h}(v) = \sum_{vu \in E} w(vu) \cdot a(u) + c \cdot p(v), \quad h = \hat{h}/\|h\|_\infty, \quad (8)$$

where $w(vu)$ denotes the weight of vu. We obtain personalized vectors a and h; the corresponding node scoring method will be denoted by *HitsAuth* and *HitsHub*.

3. EXPERIMENTS

Our experiments are based on three monthly Wikipedia snapshots of 2011-09-01, 2011-10-07 and 2011-11-15, with over 7M nodes and 180M edges. We selected 35 queries that are related to headline news events either from September or from October 2011. For each query we set a list of manually

	Month of change			The other month	
	NDCG	recall	MRR	recall	MRR
None	0.243	0.456	0.865	0.327	0.423
PageRank-0.9	0.258	**0.555**	0.964	0.368	0.731
HitsAuth-200	**0.315**	**0.543**	**1.101**	0.377	0.689
HitsHubs-200	0.266	**0.531**	0.953	0.378	0.527
Combined	0.268	**0.557**	0.994	0.365	0.746

Table 2: Recall@15 and MRR with the overall best parameter settings. Scores in bold show performances statistically significantly better than the baseline ($p < 0.01$).

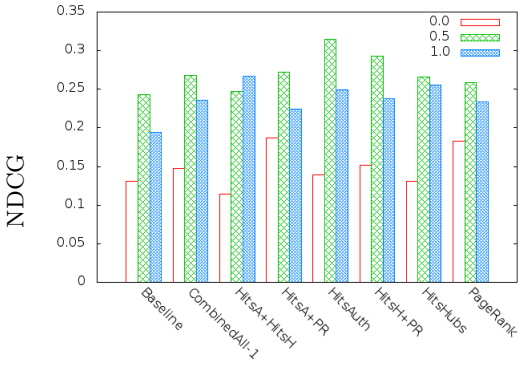

Figure 1: NDCG as a function of the values of α (0.0, 0.5 and 1.0), for different algorithms and the baseline. The top 100 BM25 score articles are expanded by another 100. Change is saturated with $T = 10$ and personalization is $c = 0.9$ for PageRank and 200 for HITS.

selected relevant articles. We made the list of queries and the assessment publicly available[2].

For all queries, we measured our methods focusing both on the change from September to October and from October to November. We expect that September events score higher in the first while October events in the second case. Results for the accuracy measures are found in Table 2.

3.1 Evaluation measures

We evaluate the performance by the Normalized Discounted Cumulative Gain (NDCG), Mean Reciprocal Rank (MRR), Early Recall, Graph Density Change and Tendency of short paths staying within the top hits. The last two measures consider the change in the connectivity of top ranked articles. We not only expect that the pages in the result set are relevant but we also show temporal evolution, growth or densification. Therefore we also measure the number of edges for the top ranked 15 articles.

We apply PageRank to measure the fraction of short paths staying inside the top 15 hits. By equation (4) if we define a personalization vector p that is identically distributed over the nodes of the selected subgraph, the sum of $\mathrm{PPR}_p(v)$ over nodes v of the subgraph gives a weighted sum of paths that terminate within the subgraph.

3.2 Retrieval performance

We overview the results of various combinations of change measures, graph ranking and the BM25 score in Table 2 with statistically significant improvements shown bold ($p < 0.01$). As best parameters we identified the following values

[2] `http://dms.sztaki.hu/en/download/wimmut-searching-and-navigating-wikipedia`

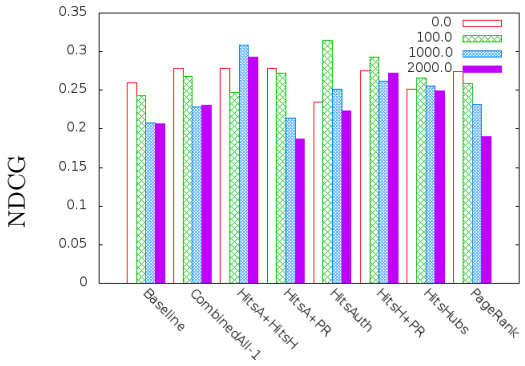

Figure 2: NDCG as the function of the size of the expansion (0, 100, 1000 and 2000), for different algorithms and the baseline. Here $\alpha = 0.5$, change is saturated with $T = 10$, and personalization is $c = 0.9$ for PageRank and 200 for HITS.

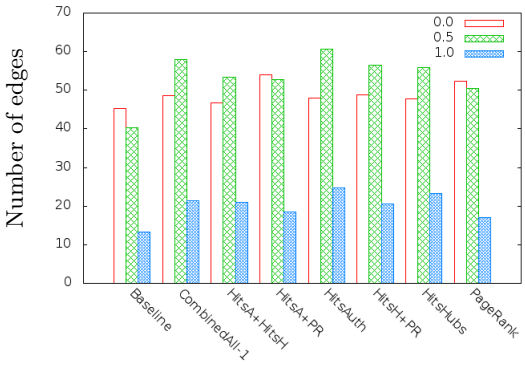

Figure 3: The number of edges among the top 15 hits in the snapshots before and after the event as well as their difference as the function of α (0.0, 0.5 and 1.0), for different algorithms and the baseline.

or ranges: seed set size of 10-100; seed set expanded by 100 more articles; change saturation $T = 10$ (it has little effect); change and IR combination ratio $\alpha = 0.5$; PageRank damping $c = 0.9$ and HITS personalization $c = 200$.

In Fig. 1 we show how NDCG is influenced by the value of α, and in Fig. 2 we show how NDCG is influenced by size of the expansion. We observe that the balanced mix of $\alpha = 0.5$ is the best choice by including both text based relevance and change measures. The importance of the temporal aspect of our queries is clear in the weaker performance of the BM25 score itself ($\alpha = 1$) while the change-only $\alpha = 0$ performs weakest.

We should be careful in expanding the seed set: best results are obtained if we extend the original top 100 articles with another 100 changing ones in the neighborhood. However, for much larger subgraphs, all algorithms show topic drift and strong personalization is needed with $c = 0.9$-0.95 for PageRank and 100-200 for HITS.

3.3 Graph density

We compare the quality and connectivity of the linkage within the top results both by counting the edges and computing the sum of personalized PageRank kept within the displayed result set in Figs. 3–4. Note that the denser sub-

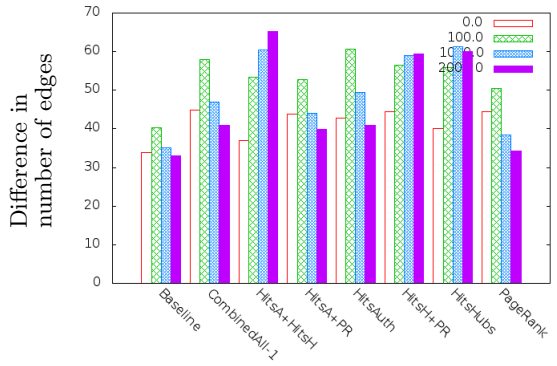

Figure 4: The difference of the number of edges among the top 15 hits between the snapshots before and after the event as the function of the size of the expansion (0, 100, 1000 and 2000), for different algorithms and the baseline.

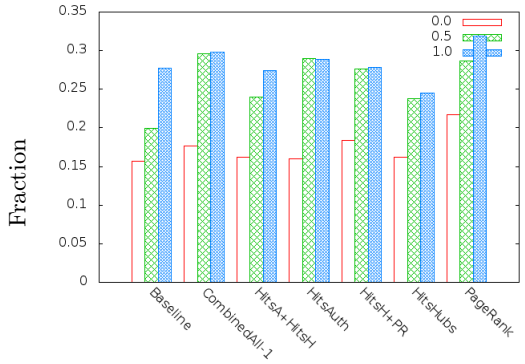

Figure 5: The fraction of short paths kept within the top 15 hits as the function of α (0.0, 0.5 and 1.0), for different algorithms and the baseline.

graphs are also of improved relevance, as seen in Section 3.2.

When comparing the effect of α (Fig. 3) and the expansion size (Fig. 4), we note that the graph algorithms tend to overfit for irrelevant lists, for example for very large expansion (1000-2000) and $\alpha = 1$ considering change only. The results show clear topic drift in these cases while good performance for low expansion. The findings are similar for the PageRank based measure of short paths staying inside the subgraph in Fig. 5.

4. RELATED RESULTS

Temporal information retrieval has mainly been considered as a task for either temporal query aspect detection or Web content dating. We are aware of no results for ranking with respect to a specified date as part of the input query. In a result with goals similar to ours for timestamped news [5], time sensitive queries are analyzed by relying on the publication date of documents. However, their goal is to identify relevant time ranges for queries, unlike in our result where we search for events in a given time related to the query.

Similar to our task is the identification of important events from the Blogosphere [10] as in the TREC Top Stories Identification task. Among others these results rely on timing and relevance as key factors but their results do not take connectivity into account. Topic detection by analyzing term bursts is first described in [9]; subsequent results rely on topic detection and tracking (TDT) achievements [3]. The general properties of the Wikigraph including degrees and their change in time is measured in [1].

As a different task, the extraction of a chronological order from free text turns out to be a difficult [11] and considered as part of the TAC Temporal Slot Filling task [14, 15]. In one application, the timeline of events related to G8 leaders is extracted [4] by starting with a query for a given politician and then identifying the date from free text. We believe that these tasks can be enhanced by our techniques.

Conclusions

We identified events in time by relying on edit dates aggregated in a neighborhood defined by hyperlinks. We proposed algorithms based on personalized HITS and PageRank that amplify changes and relevance in a given graph neighborhood. Part of our results is a query set with relevance assessment suited for the data set as well as the annotated document collection. We believe that our results find application in other related social networking and Web IR tasks.

5. REFERENCES

[1] L. Buriol, C. Castillo, D. Donato, S. Leonardi, and S. Millozzi. Temporal analysis of the wikigraph. In *Web Intelligence*, pp. 45–51. IEEE, 2006.

[2] H. Chang, D. Cohn, and A. McCallum. Learning to Create Customized Authority Lists. In *Proc. 7th ICML*, pp. 127–134, 2000.

[3] K. Chen, L. Luesukprasert, S. Chou, et al. Hot topic extraction based on timeline analysis and multidimensional sentence modeling. *IEEE TKDE*, 19(8):1016–1025, 2007.

[4] H. Chieu and Y. Lee. Query based event extraction along a timeline. In *Proc. 27th ACM SIGIR*, pp. 425–432, 2004.

[5] W. Dakka, L. Gravano, and P. Ipeirotis. Answering general time-sensitive queries. *IEEE TKDE*, 24(2):220–235, 2012.

[6] C. Faloutsos, K. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *Proc. 10th ACM SIGKDD*, pp. 118–127. ACM, 2004.

[7] K. Kim and S. Cho. Personalized mining of web documents using link structures and fuzzy concept networks. *Applied Soft Computing*, 7(1):398–410, 2007.

[8] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[9] J. Kleinberg. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, 2003.

[10] Y. Lee, H. Jung, W. Song, and J. Lee. Mining the blogosphere for top news stories identification. In *Proc. 33rd ACM SIGIR*, pp. 395–402. ACM, 2010.

[11] I. Mani and G. Wilson. Robust temporal processing of news. In *Proc. 38th ACL*, pp. 69–76, 2000.

[12] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford University, 1998.

[13] T. Sarlós, A. A. Benczúr, K. Csalogány, D. Fogaras, and B. Rácz. To randomize or not to randomize: Space optimal summaries for hyperlink analysis. In *Proc. 15th WWW*, pp. 297–306, 2006.

[14] M. Verhagen, R. Gaizauskas, F. Schilder, M. Hepple, G. Katz, and J. Pustejovsky. Semeval-2007 task 15: Tempeval temporal relation identification. In *Proc. 4th International Workshop on Semantic Evaluations*, pp. 75–80. ACL, 2007.

[15] M. Verhagen, R. Sauri, T. Caselli, and J. Pustejovsky. Semeval-2010 task 13: Tempeval-2. In *Proc. 5th International Workshop on Semantic Evaluation*, pp. 57–62. ACL, 2010.

[16] S. White and P. Smith. Algorithms for estimating relative importance in networks. *Proc 9th ACM SIGKDD*, page 266, 2003.

BUbiNG: Massive Crawling for the Masses*

Paolo Boldi
Dipartimento di Informatica
Università degli Studi di
Milano, Italy
boldi@di.unimi.it

Andrea Marino
Dipartimento di Informatica
Università degli Studi di
Milano, Italy
andrea.marino1@unimi.it

Massimo Santini
Dipartimento di Informatica
Università degli Studi di
Milano, Italy
santini@di.unimi.it

Sebastiano Vigna
Dipartimento di Informatica
Università degli Studi di
Milano, Italy
vigna@acm.org

ABSTRACT

Although web crawlers have been around for twenty years by now, there is virtually no freely available, open-source crawling software that guarantees high throughput, overcomes the limits of single-machine tools and at the same time scales linearly with the amount of resources available. This paper aims at filling this gap.

We describe BUbiNG, our next-generation web crawler built upon the authors' experience with UbiCrawler [8] and on the last ten years of research on the topic. BUbiNG is an open-source Java fully distributed crawler (no central coordination), and single BUbiNG agents using sizeable hardware can crawl several thousands pages (per agent) per second respecting strict politeness constraints, both host- and IP-based. Unlike existing open-source distributed crawlers that rely on batch techniques (like MapReduce), BUbiNG job distribution is based on modern high-speed protocols so to achieve very high throughput.

1. INTRODUCTION

A *web crawler* (sometimes also known as a *(ro)bot* or *spider*) is a system that downloads systematically a large number of web pages starting from a seed. Web crawlers are, of courses, used by search engines, but also by companies selling "Search–Engine Optimization" services, archival projects such as the Internet Archive, surveillance systems (e.g., that scan the web looking for cases of plagiarism), and by entities performing statistical studies of the structure and the content of the web, just to name a few.

*The authors were supported by the EU-FET grant NADINE (GA 288956).

The basic inner working of a crawler is surprisingly simple form a theoretical viewpoint: it is a form of traversal (for example, a breadth-first visit). Starting from a given *seed* of URLs, a set of associated pages is downloaded, their content is parsed, and the resulting links are used iteratively to collect new pages.

Albeit in principle a crawler just performs a visit of the web, there are a number of factors that make the visit of a crawler inherently different from a textbook algorithm. The first and most important difference is that the size of the graph to be explored is unknown and huge; in fact, infinite. The second difference is that visiting a node (i.e., downloading a page) is a complex process that has intrinsic limits due to network speed, latency, and *politeness*—the requirement of not overloading servers during the download. Not to mention the countless problems (errors in DNS resolutions, protocol or network errors, presence of traps) that the crawler may find on its way.

In this paper we describe the design and implementation of BUbiNG, our new web crawler built upon the experience with UbiCrawler [8] and on the last ten years of research on the topic. BUbiNG aims at filling an important gap in the range of available crawlers. In particular:

- It is a pure-Java, open-source crawler released under the Gnu GPLv3.
- It is fully distributed: multiple agents perform the crawl concurrently and handle the necessary coordination without the need of any central control; given enough bandwidth, the crawling speed grows linearly with the number of agents.
- Its design acknowledges that CPUs and OS kernels have become extremely efficient in handling a large number of threads, in particular if they are mainly I/O-bound, and that large amounts of RAM are by now easily available at a moderate cost.
- More in detail, we assume that the memory used by an agent must be *constant* in the number of discovered URLs, but that it can *scale linearly* in the number of discovered hosts. This assumption simplifies and makes several data structures more efficient.
- It is very fast: on a 64-core, 64GB workstation it can download hundreds of million of pages at more than

9 000 pages per second respecting politeness both by host and by IP, analyzing, compressing and storing more than 140 MB/s of data.

- It is extremely configurable: beyond choosing the sizes of the various data structures and the communication parameters involved, implementations can be specified by reflection in a configuration file and the whole dataflow followed by a discovered URL can be controlled by arbitrary user-defined filters, that can further be combined with standard Boolean-algebra operators.

- It guarantees that hostwise the visit is an exact breadth-first visit (albeit the global policy can be customized).

- It guarantees that politeness intervals are satisfied both at the host and the IP level, that is, that two data fetch to the same host or IP are separated by at least a specified amount of time. The two intervals can be set independently, and, in principle, customized per host or IP.

When designing a crawler, one should always ponder over the specific usage the crawler is intended for. This decision influences many of the design details that need to be taken. Our main goal is to provide a crawler that can be used out-of-the-box as an archival crawler, but that can be easily modified to accomplish other tasks. Being an archival crawler, it does not perform any refresh of the visited pages, and moreover it tries to perform a visit that is as close to breadth-first as possible (more about this below). Both behaviors can in fact be modified easily in case of need, but this discussion (on the possible ways to customize BUbiNG) is out of the scope of this paper.

We plan to use BUbiNG to provide new data sets for the research community. Datasets crawled by UbiCrawler have been used in hundreds of scientific publications, but BUbiNG makes it possible to gather data orders of magnitude larger.

2. RELATED WORKS

Web crawlers have been developed since the birth of the web. The first generation crawler dates back to the early 90s: World Wide Web Worm [24], RBSE spider [16], MOMspider [18], WebCrawler [30]. One of the main contributions of these works has been that of pointing out some of the main algorithmic and design issues of crawlers. In the meanwhile, several commercial search engines, having their own crawler (e.g., AltaVista) were born. In the second half of the 90s, the fast growth of the web called for the need of large-scale crawlers, like the crawler of Internet Archive Module [11] and the first generation of the Google crawler [9]. This generation of spiders was able to download efficiently tens of millions of pages. At the beginning of 2000, the scalability, the extensibility, and the distribution of the crawlers become a key design point: this was the case of the Java crawler Mercator [28] (distributed version of [19]), Polybot [32], IBM WebFountain [15], and UbiCrawler [8]. These crawlers were able to produce snapshots of the web of hundreds of millions of pages.

Recently, a new generation of crawlers was designed, aiming to download billions of pages, like [22]. Nonetheless, none of them is freely available and open source: BUbiNG is the first open-source crawler designed to be fast, scalable and runnable on commodity hardware.

For more details about previous works or in the main issues in the design of crawlers, we refer the reader to [29, 26, 31].

2.1 Open-source crawlers

Although web crawlers have been around for twenty years by now (since the spring of 1993, according to [29]), the area of freely available ones, let alone open-source, is still quite narrow. With the few exceptions that will be discussed below, most stable projects we are aware of (GNU wget, Ht//Dig, mngoGoSearch, to cite a few) do not (and are not designed to) scale to download more than few thousands or tens of thousands pages. They can be useful to build an intranet search engine, but not for web-scale experiments.

Heritrix [1, 27] is one of the few examples of an open-source search engine designed to download large datasets: it was developed starting from 2003 by Internet Archive [2] (a non-profit corporation aiming to keep large archival-quality historical records of the world-wide web) and it has been since actively developed. Heritrix (available under the Apache license) is a single-machine crawler, although it is of course multi-threaded, which is the main hindrance to its scalability. The default crawl order is breadth-first, as suggested by the archival goals behind its design. On the other hand, it provides a powerful checkpointing mechanism and a flexible way of filtering and processing URLs after and before fetching. It is worth noting that Internet Archive proposed, implemented (in Heritrix) and fostered a standard format for archiving web content, called WARC, that is now an ISO standard [4] and that BUbiNG is also adopting for storing the downloaded pages.

Nutch [21] is one of the best known existing open-source web crawlers; in fact, the goal of Nutch itself is much broader in scope, because it aims at offering a full-fledged search engine under all respects: besides crawling, Nutch implements features such as (hyper)text-indexing, link analysis, query resolution, result ranking and summarization. It is natively distributed (using Apache Hadoop as task-distribution backbone) and quite configurable; it also adopts breadth-first as basic visit mechanism, but can be optionally configured to go depth-first or even largest-score first, where scores are computed using some scoring strategy which is itself configurable. Scalability and speed are the main design goals of Nutch; for example, Nutch was used to collect TREC ClueWeb09 dataset[1], the largest web dataset publicly available as of today consisting of 1 040 809 705 pages, that were downloaded at the speed of 755.31 pages/s [3], but to do this they used a Hadoop cluster of 100 machines [12], so their real throughput was of about 7.55 pages/s *per machine*. This figure is not unexpected: using Hadoop to distribute the crawling jobs is easy, but not efficient, because it constrains the crawler to work in a batch fashion. It shouldn't be surprising that using a modern job-distribution framework like BUbiNG does increases the throughput by orders of magnitude.

[1]The new ClueWeb12 dataset, that is going to be released soon, was collected using Heritrix, instead: five instances of Heritrix, running on five Dell PowerEdge R410, were run for three months, collecting 1.2 billions of pages. The average speed was of about 38.6 pages per second per machine.

3. ARCHITECTURE OVERVIEW

BUbiNG stands on a few architectural choices which in some cases contrast the common folklore wisdom. We took our decisions after carefully benchmarking several options and gathering the hands-on experience of similar projects.

- The fetching logic of BUbiNG is built around thousands of identical *fetching threads* performing essentially only synchronous (blocking) I/O. Experience with recent Linux kernels and increase in the number of cores per machine shows that this approach consistently outperforms asynchronous I/O. This strategy simplifies significantly the code complexity, and makes it trivial to implement features like HTTP/1.1 "keepalive" multiple-resource downloads.
- *Lock-free* [25] data structures are used to "sandwich" fetching threads, so that they never have to access lock-based data structures. This approach is particularly useful to avoid direct access to synchronized data structures with logarithmic modification time, such as priority queues, as contention between fetching threads can become very significant.
- URL storage (both in memory and on disk) is entirely performed using byte arrays. While this approach might seen anachronistic, the Java `String` class can easily occupy three times the memory used by a URL in byte-array form (both due to additional fields and to 16-bit characters) and doubles the number of objects. BUbiNG aims at exploiting the large memory sizes available today, but garbage collection has a linear cost in the number of objects: this factor must be taken into account.
- Following UbiCrawler's design [8], BUbiNG agents are identical and autonomous. The assignment of URLs to agents is entirely customizable, but by default we use *consistent hashing* as a fault-tolerant, self-configuring assignment function.

In this section, we overview the structure of a BUbiNG agent: the following sections detail the behavior of each component. The inner structure and data flow of an agent is depicted in Figure 1.

The bulk of the work of an agent is carried out by low-priority *fetching threads*, which download pages, and *parsing threads*, which parse and extract information from downloaded pages. Fetching threads are usually thousands, and spend most of their time waiting for network data, whereas one usually allocates as many parsing threads as the number of available cores, because their activity is mostly CPU bound.

Fetching threads are connected to parsing threads using a lock-free *result* list in which they enqueue buffers of fetched data, and wait for a parsing thread to analyze them. Parsing threads poll the result list using an exponential backoff scheme, perform actions such as parsing and link extraction, and signal back to the fetching thread that that the buffer can be filled again.

As parsing threads discover new URLs, they enqueue them to a *sieve* that keeps track of which URLs have been already discovered (we do not want to download the same URL twice). A sieve is a data structure similar to a queue with memory: each enqueued element will be dequeued at some later time, with the guarantee that an element that is enqueued multiple times will be dequeued just once. URLs are added to the sieve as they are discovered by parsing. A cache sits in front of the sieve to avoid that frequently found URLs put the sieve under stress. The cache has also another important goal: it avoids that frequently found URLs assigned to another agent are retransmitted several times.

URLs that come out of the sieve are ready to be visited, and they are taken care of (stored, organized and managed) by the *frontier*, which is actually itself decomposed into several modules.

The most important data structure of the frontier is the *workbench*, an in-memory data structure that keeps track of the visit state of each host currently visited and that can check in constant time whether some host can be accessed for download without violating the politeness constraints. Note that to attain the goal of several thousands downloaded pages per second without violating politeness constraints it is necessary to keep track of the visit state of hundreds of thousands of hosts.

When a host is ready for download, its visit state is extracted from the workbench and moved to a lock-free *todo queue* by a suitable thread. Fetching threads poll the todo queue with an exponential backoff, fetch resources from the retrieved visit state[2] and then put it back on the workbench. Note that we expect that once a large crawl has started, the todo queue will never be empty, so fetching threads will never have to wait. Most of the efforts of the components of the frontier are actually geared towards avoiding that fetching threads ever wait on an empty todo queue.

The only active component (i.e., a thread) of the frontier is the *distributor*: it is a high-priority thread that processes URLs that come out of the sieve (and must therefore be crawled). Assuming for a moment that memory is unbounded, the only task of the distributor is that of iteratively dequeueing a URL from the sieve, checking whether it belongs to a host for which a visit state already exists, and then either creating a new visit state or enqueuing the URL to an existing one. If a new visit state is necessary, it is passed to a set of *DNS threads* that perform DNS resolution and then move the visit state on the workbench.

Since, however, breadth-first visit queue grows exponentially, and the workbench can use only a fixed amount of in-core memory, it is necessary to *virtualize* it, that is, writing on disk part of the URLs coming out of the sieve. To decide whether to keep a visit state entirely in the workbench or to virtualize it, and also to decide when and how URLs should be moved from the virtualizer to the workbench, the distributor uses a complex policy that is described later.

Finally, every agent stores resources in its *store* (that may possibly reside on a distributed or remote file system). The native BUbiNG store is a compressed file in the Web ARChive (WARC) format (the standard proposed and made popular by Heritrix). This standard specifies how to combine several digital resources with other information into an aggregate archive file. In BUbiNG compression happens in a heavily parallelized way, with parsing threads compressing independently pages and using concurrent primitives to pass compressed data to a flushing thread.

3.1 The sieve

[2]Possibly multiple resources on a single TCP connection using the "keepalive" feature of HTTP 1.1.

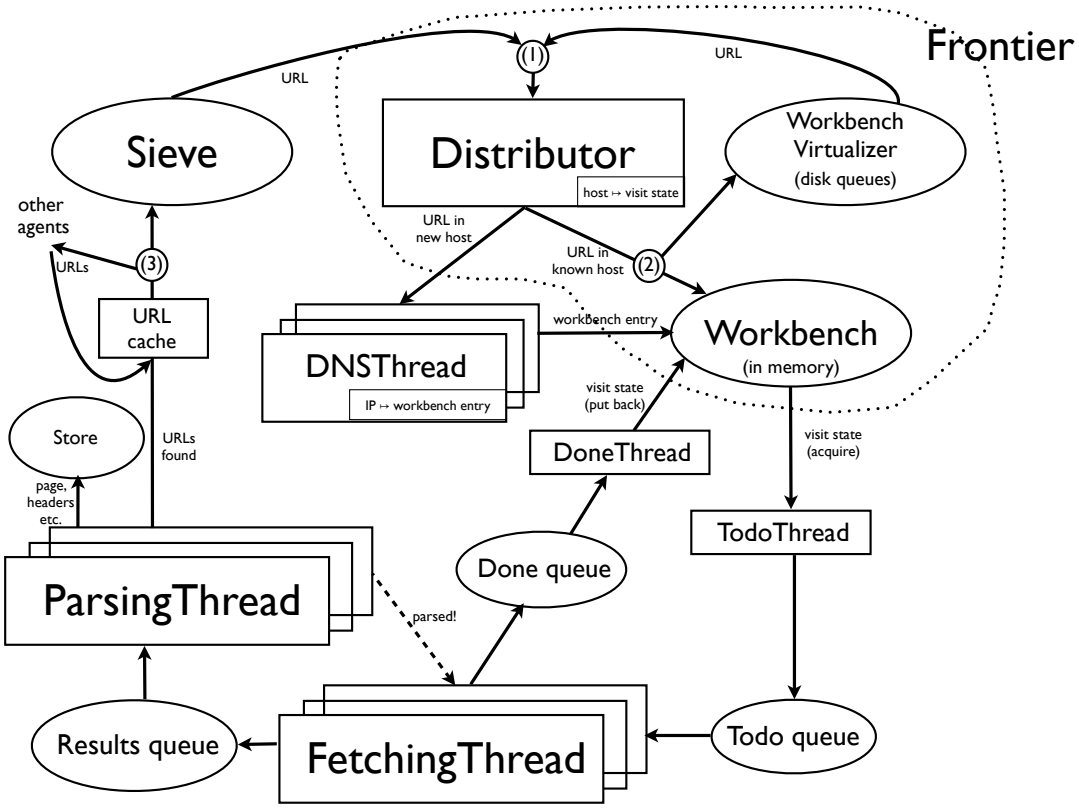

Figure 1: Overview of the architecture of a BUbiNG agent. Ovals represent data structures, whereas rectangles represent threads (or sets of threads).

A *sieve* is a queue with memory: it provides enqueue and dequeue primitives, similarly to a standard queue; each element enqueued to a sieve will be eventually dequeued later. However, a sieve guarantees also that if an element is enqueued multiple times, it will be dequeued just one time. Sieves (albeit not called with this name) have always been recognized as a fundamental basic data structure for a crawler: their main implementation issue lies in the unbounded, exponential growth of the number of discovered URLs. While it is easy to write enqueued elements to a disk file, checking that an element is not returned multiple times requires *ad-hoc* data structures, as standard dictionaries would use too much in-core memory.

The actual sieve implementation used by BUbiNG can be customized, but the default one, called `MercatorSieve`, is similar to the one suggested in [19] (hence its name). Each element known to the sieve is stored as a 64-bit hash in a disk file. Every time a new element is enqueued, its hash is stored in an in-memory array, and the element is saved in an auxiliary file. When the array is full, it is sorted and compared with the set of elements known to the sieve. The auxiliary file is then scanned, and previously unseen elements are stored for later dequeueing. All these operations require only sequential access to all files involved. Note that the output order is guaranteed to be the same of the input order (i.e., elements are appended in the same order in which they appeared the first time).

A generalization of the idea of a sieve, which adds the possibility of associating values with elements, is the DRUM (Disk Repository with Update Management) structure used by IRLBot and described in [22]. A DRUM provides additional operations that retrieve or update the values associated with elements. From an implementation viewpoint, DRUM is a Mercator sieve with multiple arrays, called *buckets*, in which a careful orchestration of in-memory and on-disk data makes it possible to sort in one shot sets of elements an order of magnitude larger than what the Mercator sieve would allow using the same in-core memory. However, to do so DRUM must sacrifice breadth-first order: due to the inherent randomization of key placement in the buckets, there is no guarantee that URLs will be crawled in breadth-first order, not even per host. Finally, the tight analysis in [22] about the properties of DRUM is unavoidably bound to the single-agent approach of IRLBot: for example, the authors conclude that a URL cache to reduce the number of insertions in the DRUM is not useful, but the same cache reduces significantly network transmissions. Once the cache is in place, the Mercator sieve becomes much more competitive.

There are several other implementations of the sieve logic currently used. A quite common choice is to use an explicit queue and a *Bloom filter* [7] to remember enqueued elements. Albeit popular, this choice has no theoretical guarantee: while it is possible to decide *a priori* the maximum number of pages that will ever be crawled, it is very difficult to bound in advance the size of the *discovered* URLs, and this number is essential in sizing the Bloom filter. If

the discovered URLs are significantly more than expected, several pages are likely to be lost because of false positives. A better choice is to use a dictionary of fixed-size *fingerprints* obtained from URLs using a suitable hash function. The disadvantage is that the structure would no longer use constant memory.

3.2 The workbench

The *workbench* is an in-memory data structure that contains the next URLs to be visited, and can check in constant time whether a URL is ready for download without violating politeness limits. It is one of the main novel ideas in BUbiNG's design, and it is one of the main reasons why we can attain a very high throughput.

First of all, URLs associated with a specific host[3] are kept in a structure called *visit state*, containing a FIFO queue of the next URLs to be crawled for that host along with a `next-fetch` field that specifies the first instant in time when a URL from the queue can be downloaded, according to the per-host politeness configuration. Note that inside a visit state we only store a byte-array representation of the path and query of a URL: this approach significantly reduces object creation, and provides a simple form of compression by prefix omission.

Visit states are further gathered by IP address in *workbench entries*; every time the first URL for a given host is found, a new visit state is created and then the IP address is determined (by one of the *DNS threads*): the new visit state is either put in a new workbench entry (if no known host was as yet associated to that IP), or in an existing one.

A workbench entry contains a queue of visit states prioritized by their `next-fetch` field. In other words, a workbench entry contains all visit states associated with the same IP, along with an IP-specific `next-fetch`, containing the first instant in time when the IP address can be accessed again, according to the per-IP politeness configuration. The *workbench* is the queue of all workbench entries, prioritized on the `next-fetch` field of each entry *maximized* with the `next-fetch` field on the top element of its queue of visit states. In other words, the workbench is a priority queue of priority queues of FIFO queues.

We remark that due to our choice of priorities *there is a host that can be visited without violating host or IP politeness if and only if the first URL of the top visit state of the top workbench entry can be visited*. Moreover, if there is no such host, the delay after which a host will be ready is given by the priority of the top workbench entry minus the current time.

The workbench acts as a *delay queue*: its dequeue operation waits, if necessary, until a host is ready to be visited. At that point, the top entry E is removed from the workbench and the top visit state is removed from E. The visit state and the associated workbench entry act as a *token* that is virtually passed between BUbiNG's components to guarantee that no component is working on the same workbench entry at the same time (in particular, this forces both kinds of politeness). In practice, as we mentioned in the overview, dequeueing is performed by a high-priority thread, the *todo thread*, that constantly dequeues visit states from the workbench and enqueue them to a lock-free *todo queue*, which is then accessed by fetching threads. This approach, besides avoiding contention by thousands of threads on a relatively slow structure, makes the number of visit states that are ready for downloads easily measurable: it is just the size of the todo queue. The downside is that, in principle, using very skewed per-host or per-IP politeness delays might cause the order of the todo queue not to reflect the actual priority of the visit state contained therein.

3.3 Fetching threads

A *fetching thread* is a very simple thread that iteratively extracts visit states from the todo queue. If the todo queue is empty, a standard exponential backoff procedure is used to avoid polling the list too frequently, but the design of BUbiNG aims at keeping the todo queue nonempty and avoiding backoff altogether.

Once a fetching thread acquires a visit state, it tries to fetch the first URL of the visit state FIFO queue. If suitably configured, a fetching thread can also iterate the fetching process on more URLs for a fixed amount of time, so to exploit the "keepalive" feature of HTTP 1.1.

Each fetching thread has an associate *fetch data* instance in which the downloaded data are buffered. Fetch data include a transparent buffering method that keeps in memory a fixed amount of data and dumps on disk the remaining part. By sizing the fixed amount suitably, most requests can be completed without accessing the disk, but at the same time rare large requests can be handled without allocating additional memory.

After a URL has been fetched, the fetch data is put in the *results* queue so that one of the parsing threads will parse it. One the parsing is over, the parsing thread will signal back so the fetching thread will be able to start working on a new URL. Once a fetching thread has to work a new visit state, it puts the current visit state on a *done queue*, from which it will be dequeued by a suitable thread that will put it back on the workbench together with its associated entry.

Most of the time, a fetching thread is blocked on I/O, which makes it possible to run thousands of them in parallel. Indeed, the number of fetching threads determines the amount of parallelization BUbiNG can achieve while fetching data from the network, so it should be chosen as large as possible, compatibly with the amount of bandwidth available and with the memory used by fetch data.

3.4 Parsing threads

A *parsing thread* iteratively extracts from the results queue fetch data that have been previously enqueued by a fetching thread. Then, the content of the HTTP response is analyzed and possibly parsed. If the response contains an HTML page, the parser will produce a set of URLs that will be first checked against the URL cache, and then, if not already seen, either sent to another agent, or enqueued to the sieve (given that the maximum number of URLs per host has not been exceeded).

During the parsing phase, a parsing thread computes a signature of the content of the response. In the case of HTML pages, some heuristic is used to collapse near-duplicates (e.g., most HTML attributes are stripped). The signature is stored

[3]Every URL is made [6] by a scheme (also popularly called "protocol"), an authority (a host, possibly a port number, and possibly some user information) and a path to the resource, possibly followed by a query (that is separated from the path by a "?"). BUbiNG's data structures are built around the pair scheme+authority, but in this paper we will use the more common word "host" to refer to it.

in a Bloom filter [7] and it is used to avoid crawling several times the same page (or near-duplicate pages).[4] Finally, the content of the response is saved to the store.

The number of parsing threads should be equal to the number of available cores.

3.5 DNS threads

DNS threads are used to solve host names of new hosts: a DNS thread continuously dequeues from the list of newly discovered visit states and *resolves* its host name, adding it to a workbench entry (or creating a new one, if the IP itself is new), and putting it on the workbench.

The number of DNS threads is limited by the kind of DNS service the crawler relies upon. In our experience, it is essential to run a local recursive DNS server to avoid the bottleneck of an external server.

3.6 The workbench virtualizer

The workbench virtualizer is a sequence of k on-disk URL queues (at the beginning $k = 1$), called *virtual queues*; the last virtual queue is called *overflow queue*. This design is inspired by the BEAST module of IRLbot [22], albeit it is more geared towards maintaining the visit order as close as possible to a breadth-first visit, rather than using prioritization.

Conceptually, all URLs that have been extracted from the sieve but have not yet been fetched are enqueued in the workbench visit state they belong to, in the exact order in which they came out of the sieve. Since, however, we aim at crawling with an amount of memory that is *constant* in the number of discovered URLs, part of the queue must be written on disk. Each virtual queue contains a fraction of URLs from each visit state, in such a way that the overall URL order respects, *per host*, the original breadth-first order.

Virtual queues are consumed as the visit proceeds, following the natural per-host breadth-first order. As fetching threads download URLs, the workbench is partially freed and can be filled with URLs coming from the virtual queues. When all virtual queues preceding the overflow queue have been exhausted, the number of virtual queues is increased (usually doubled) and the content of the overflow queue is redistributed on the set of new queues based on an estimate of the future time at which each URL will be needed.

3.7 The distributor

The *distributor* is a high-priority thread that orchestrates the movement of URLs out of the sieve, and loads as necessary URLs from virtual queues into the workbench.

As the crawl proceeds, URLs get accumulated in workbench visit states at different speeds, both because hosts have different responsiveness and because websites have different sizes and branching factors. Moreover, the size occupied by the workbench has a (configurable) limit that cannot be exceeded, as one of the central design goals of BUbiNG is that the amount of central memory occupied cannot grow unboundedly in the size of the discovered URLs, but only in the number of hosts discovered. Thus, filling the workbench blindly with URLs coming out of the sieve would soon result in having in the workbench only URLs belonging to a limited number of hosts.

[4]In a post-crawl phase, there are several more sophisticated approaches that can be applied, like *shingling* [10], *simhash* [14], *fuzzy fingerprinting* [17, 13] and others, like [23].

The *front* of a crawl, at any given time, is the number of visit states that is ready for download by politeness constraints. The front size determines the overall throughput of the crawler—because of politeness, the number of distinct hosts currently being visited is the crucial datum that establishes how fast or slow the crawl is going to be.

One of the two forces driving the distributor is, indeed, that *the front should always be large enough so that no fetching thread has ever to wait.* To attain this goal, the distributor enlarges dynamically the *required front size*: each time a fetching thread has to wait, although the current front size is larger than the current required front size, the latter is increased. After a warmup phase, the required front size stabilizes to a value that depends on the kind of host visited and on the amount of resources available. At that point, it is impossible to have a faster crawl given the resources available, as all fetching threads are continuously downloading data. Increasing the number of fetching threads, of course, may cause an increase of the required front size.

The second force driving the distributor is the (somewhat informal) requirement that *we try to be as close to a breadth-first visit as possible.* Note that this force works in an opposite direction with respect to enlarging the front—URLs that are already in existing visit states should be in principle visited *before* any URL in the sieve, but enlarging the front requires dequeueing from the sieve to find new hosts.

The distributor is also responsible for filling the workbench with URLs coming either out of the sieve, or out of virtual queues (circle numbered (1) in Figure 1). Once again, staying close to a breadth-first visit requires loading URLs in virtual queues, but keeping the front large might require reading URLs from the sieve to discover new hosts.

The distributor balances these two forces by keeping an eye on the *limbo*—the set of visit states that currently have URLs in virtual queues, but few (or no) no URLs in memory:

- if the limbo is large, the distributor will try to read from the virtual queues, in the hope that the front (the number of hosts currently being visited) can increase at the expense of the limbo size;
- if the limbo is small, the distributor will rather read from the sieve, hoping to find new sites to make the front larger.

The limbo is considered to be large when it contains more than a small fraction (typically, 1%) of the visit states with some URLs in virtual queues. If there is no room in the workbench, or the front is already large enough, the distributor just waits. The overall behavior is depicted in Figure 2.

Note that if the distributor takes the decision to read from the virtual queues (i.e., to read the first URL of the first non-empty queue), the URL is always put in the workbench (and will be later fetched). On the other hand, if the workbench reads a URL from the sieve it can be either put in the workbench or written in a virtual queue, depending on the estimate of the future time at which the URL will be needed. If the distributor decides to write the URL on a virtual queue, another URL will have to be taken either from the sieve or from the virtual queues, and so on until the workbench is full again or until the front is large enough.

3.8 Configuration and Heuristics

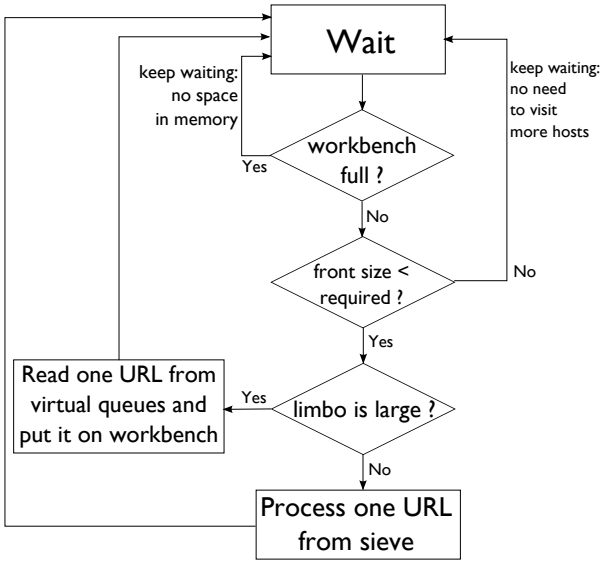

Figure 2: How the distributor interacts with the sieve, the workbench and the workbench virtualizer.

To make BUbiNG capable of a versatile set of tasks and behaviors, every crawling phase (fetching, parsing, following the URLs of a page, scheduling new URLs, storing pages) is controlled by a *filter*, a Boolean predicate that determines whether a given resource should be accepted or not. Filters can be configured both at startup and at runtime allowing for a very fine-grained control.

The type of objects a filter considers is called the *base type* of the filter. In most cases, the base type is going to be a URL or a fetched page. More precisely, a *prefetch* filter is one that has a BUbiNG URL as its base type (typically: to decide whether a URL should be scheduled for later visit, or should be fetched); a *postfetch* filter is one that has a fetched response as base type and decides whether to do something with that response (typically: to parse it, to store it, etc.).

Even if it is relatively easy to write a filter, BUbiNG contains a number of filters ready to be used. The prefetch filters include, for instance filters that accept only URLs whose host ends with a certain string, or URLs whose path ends with one of a given set of suffixes. The postfetch filters include, for instance, filters accepting certain contain types, or streams that appear to be binary.

Filters can be composed by means of Boolean operators with a short-circuit semantics. Additionally, we provide a parser that makes it possible creating filters by reflection. An example of a textual description of a composed filter is:

```
(HostEndsWith(foo.bar) and not
    ForbiddenHost(http://xxx.yyy/list-of-hosts))
or NoMoreSlashThan(10)
```

One filter, in particular, accepts only URLs whose path does not contain too many *duplicate segments*. Indeed, it is not uncommon to find URLs generated by badly configured servers that look like `http://.../foo/bar/foo/bar/...`. Our filter will not accept URLs containing a sequence of consecutive segments appearing more times than a given threshold. The implementation uses ideas from [20] to simulate a suffix-tree visit on a suffix array, and the approach of [33], for the linear-time detection of tandem arrays using suffix trees: the resulting code is one order of magnitude faster than regular expressions. We observe that, for the same purpose of avoiding bad URLs (of different kinds), it would be interesting to add a filter implementing the DUSTER (Different URL's with Similar Text) technique [5].

3.9 Distributed crawling

BUbiNG crawling activity can be distributed by running several agents over multiple machines. All agents are identical instances of BUbiNG, without any explicit leadership, similarly to UbiCrawler [8]: all data structures described above are part of each agent.

URL assignment to agent is entirely configurable. By default, BUbiNG uses just the host to assign a URL to an agent, which avoids that two different agents can crawl the same host at the same time. Moreover, since most hyperlinks are relative, each agent will be himself responsible for the large majority of URLs found in a typical HTML page [29]. Assignment of hosts to agent is performed using *consistent hashing* [8].

Communication of URLs between agents is handled by the message-passing methods of the JGroups Java library; in particular, to make communication lightweight URLs are by default distributed using UDP. More sophisticated communications between the agents rely on the TCP-based JMX Java standard remote-control mechanism, which exposes most of the internal configuration parameters and statistics. Most of the crawler structures are indeed modifiable at runtime, including, for instance, the number of parsing, fetching and DNS threads.

4. EXPERIMENTS

Testing a crawler is a delicate, intricate, arduous task: on one hand, every real-world experiment is obviously influenced by the hardware at one's disposal (in particular, by the available bandwidth). Moreover, real-world tests are difficult to repeat many times with varying parameters: you will either end up disturbing the same sites over and over again, or choosing to visit every time a different portion of the web, with the risk of introducing artifacts in the evaluation. Given these considerations, we ran two kinds of experiments: one batch was performed *in vitro* with a HTTP proxy[5] simulating network connections towards the web and generating fake HTML pages (with a configurable behavior that includes delays, protocol exceptions etc.), and another group of experiments were performed *in vivo*.

4.1 In vitro experiments

To verify the robustness of BUbiNG when varying some basic parameters, such as the number of fetching threads or the IP delay, we have run some *in vitro* simulations on a group of four machines sporting 64 cores and 64 GB of core memory. In all experiments, the number of parsing and DNS threads has been fixed and set respectively to 64 and 10. The size of the workbench has been set to 512MB, while the size of the sieve has been set to 256MB. Every *in vitro* experiment was run for 90 minutes.

Fetching threads. The first thing we wanted to test was that increasing the number of fetching threads produces a

[5]The proxy software is distributed along with the rest of BUbiNG.

Crawler	Machines	Resources (Millions)	Resources/s		Speed in MB/s	
			overall	per agent	overall	per agent
Nutch (ClueWeb09)	100 (Hadoop)	1 200	430	4.3	10	0.1
Heritrix (ClueWeb12)	5	2 300	300	60	19	3.9
IRLBot	1	6 380	1 790	1 790	40	40
BUbiNG (Milano)	3	650	2 200	735	96	32
BUbiNG (Pisa)	1	100	2 500	2 500	71	71
BUbiNG (Pisa)	4	37	5 400	1 350	168	42
BUbiNG (iStella)	1	115	3 700	3 700	135	135
BUbiNG (*in vitro*)	4	1 000	36 600	9 150	584	146

Table 1: Comparison between BUbiNG and the main existing open-source crawlers. Resources are HTML pages for ClueWeb09 and IRLBot, but include other data types (e.g., images) for ClueWeb12. For reference, we also report the throughput of IRLbot [22], although the latter is not open source. Note that ClueWeb09 was gathered using a heavily customized version of Nutch.

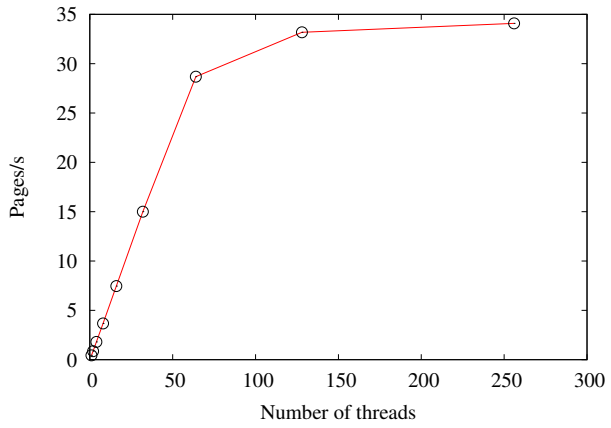

Figure 3: The average number of pages per second with respect to the number of threads using a simulated slow connection. Note the linear increase in speed until the plateau, due to the limited (300) number of threads of the simulator.

better usage of the network, and hence a larger number of requests, until the bandwidth is saturated. The results of this experiment are shown in Figure 3 and have been obtained by having the proxy simulate a network that saturates quickly, using no politeness delay. The behavior visible in the plot tells us that the increase in the number of fetching threads produces a linear increase in the number of requests until the available (simulated) bandwidth is reached; after that, the number of requests stabilizes to a plateau. Also this part of the plot tells us something: after saturating the bandwidth, we do not see any decrease in the throughput, witnessing the fact that our infrastructure does not cause any hindrance to the crawl.

Politeness. The experiment described so far uses a small number of fetching threads, because the purpose was to show what happens before saturation. Now we show what happens under a heavy load. Our second *in vitro* experiment keeps the number of fetching threads fixed but increases the amount of politeness, as determined by the IP delay. The IP (respectively host) delay is a lower bound of the time between two successive requests to the same IP (respectively host). In our simulations we varied the IP delay and always set the host delay to be eight times the IP delay. We plot BUbiNG's throughput as the IP delay (hence the host delay) increases in Figure 4 (top): to maintain the same throughput, the front size (i.e., the number of hosts being visited in parallel) must increase, as expected. Moreover, this is independent on the number of threads (of course, until the network is saturated). In the same Figure we show that the average throughput is essentially independent from the politeness (and from the number of fetching threads) and the same is true of the CPU load. Even if this could seem surprising, this is the natural consequence of the following two observations:

- even with a small amount of fetching threads, BUbiNG always tries to fill the bandwidth and to maximize the computational resources;
- even varying the IP and host delay, BUbiNG modifies the number of hosts under visit in order to tune the interleaving between their processing.

Raw speed. Finally, we wanted to test the raw speed of a cluster of BUbiNG agents. We thus ran four agents using a larger workbench (2 GB) and 1000 fetching threads, IP delay 500 ms and host delay 4 s. We ran the agents until we gathered one billion pages, averaging 36 600 pages per second on the whole cluster. We also ran the same test on a single machine, obtaining essentially the same per-machine speed, showing that BUbiNG scales linearly with the number of agents in the cluster.

4.2 In vivo experiments

We performed a number of experiments *in vivo* at different sites. The main problem we had to face is that a single BUbiNG agent on sizable hardware can saturate a 1 Gb/s geographic link, so, in fact, we were not able to perform any test in which the network was not capping the crawler. Due to resource constraints, we decided to perform medium-size experiments on a variety of architectures and network connections. In the final version of the paper, we will report data for longer-running experiments.

A first, longer experiment was performed at our university (Milano): three BUbiNG agents using the same hardware of the *in vitro* experiments gathered 650 million pages from domains of the EU, but the connection was capped at 250 Mb/s. A second set of short-running experiments was

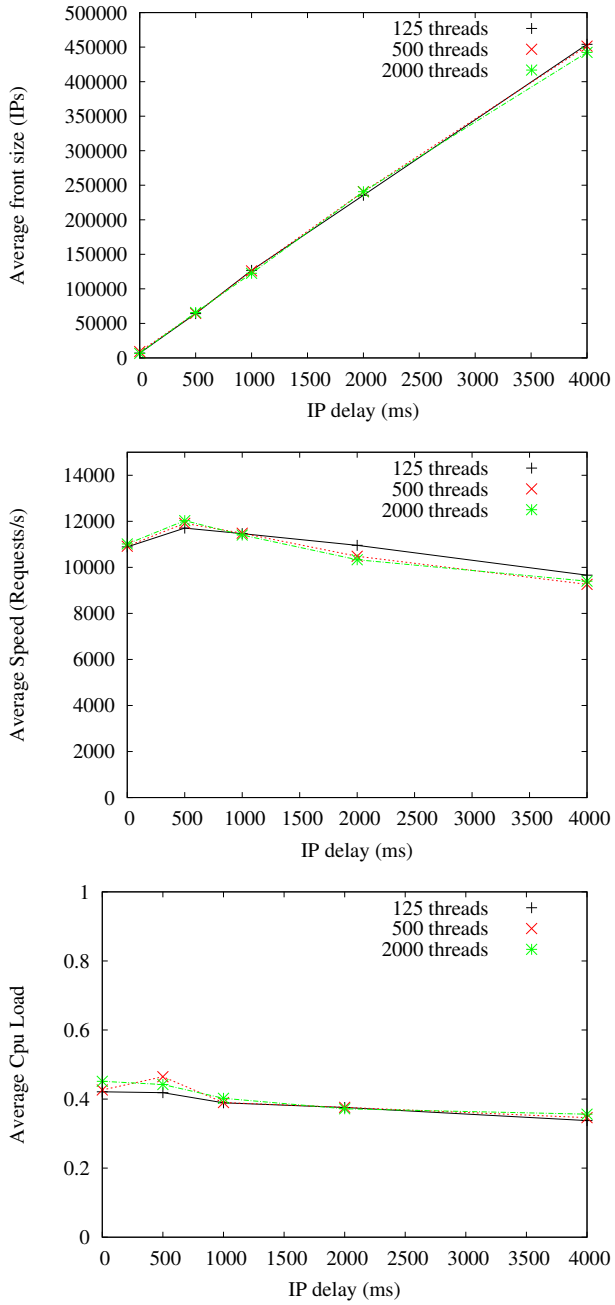

Figure 4: The average size of the front, the average number of requests per second, and the average CPU load with respect to the IP delay (the host delay is set to eight times the IP delay). Note that the front adapts linearly to the growth of the IP delay, and, due to the essentially unlimited bandwidth of the simulator, the number of fetching threads is irrelevant.

performed at Università di Pisa, using slower hardware (24-core, 24 GB RAM) capped at 1 Gb/s. Finally, iStella, an Italian commercial search engine provided us with a 48-core, 512 GB RAM machine capped at 1 Gb/s.

The results confirm the knowledge we have gathered with our *in vitro* experiment: in the iStella experiment we were able to saturate the 1 Gb/s link using a single BUbiNG agent. The two experiment at Pisa show a single, small-sized agent being able to download 2 500 pages per second, using about 2/3 of the available bandwidth, and three agents, saturating the 1 Gb/s link, downloading 5 400 pages per second. Finally, the long-running experiment at our university, albeit slow in comparison, shows the steadiness of BUbiNG after a large number of pages have been downloaded (see Figure 5).

5. COMPARISON

When comparing crawlers, many measures are possible, and depending on the task at hand, different measures might be suitable. For instance, crawling all types of data (CSS, images, etc.) usually yields a significantly higher throughout than crawling just HTML, as HTML pages are often rendered dynamically, sometimes causing a significant delay, whereas most other types are served statically. The crawling policy has also a huge influence on the throughput: prioritizing by indegree (as IRLBot does [22]) or alternative importance measure shifts most of the crawl on sites hosted on powerful servers with large-bandwidth connection. Ideally, crawler should be compared on a crawl with given number of pages in breadth-first fashion from a fixed seed, but some crawlers are not available to the public, which makes this goal unattainable.

In this section, as a tradeoff, we briefly give some very simple comparison with recent crawls made for the ClueWeb project: ClueWeb09 and ClueWeb12. The data used in this comparison are those available in [12] along with those found at `http://lemurproject.org/clueweb09/` and `http://boston.lti.cs.cmu.edu/crawler/crawlerstats.html`: notice that the data we have about those collections are sometimes slightly contradictory. We report the throughput declared by IRLBot [22], too, albeit the latter is not open source.

The results of the comparison are shown in Table 1: they show quite clearly that the speed of BUbiNG is several times that of IRLBot and one to two orders of magnitude greater than that of Heritrix or Nutch. While the comparison with the ClueWeb09 crawl is somewhat unfair (the hardware was "retired search-engine hardware"), it shows the inherent slowness of batch, Hadoop-based crawlers. The comparison with ClueWeb12 is more interesting, as the hardware used was recent and very similar to the one used in the Milano and in the *in vitro* experiment, sporting 64 GB of core memory.

All in all, our experiments show that BUbiNG's design provides a very high throughput: indeed, from our comparison, the highest throughput. The fact that the throughput can be scaled linearly just by adding agents makes it by far the fastest crawling system publicly available.

6. CONCLUSIONS

In this paper we have presented BUbiNG, our next-generation distributed open-source Java crawler. BUbiNG is order of magnitudes faster than existing open-source crawlers, scales

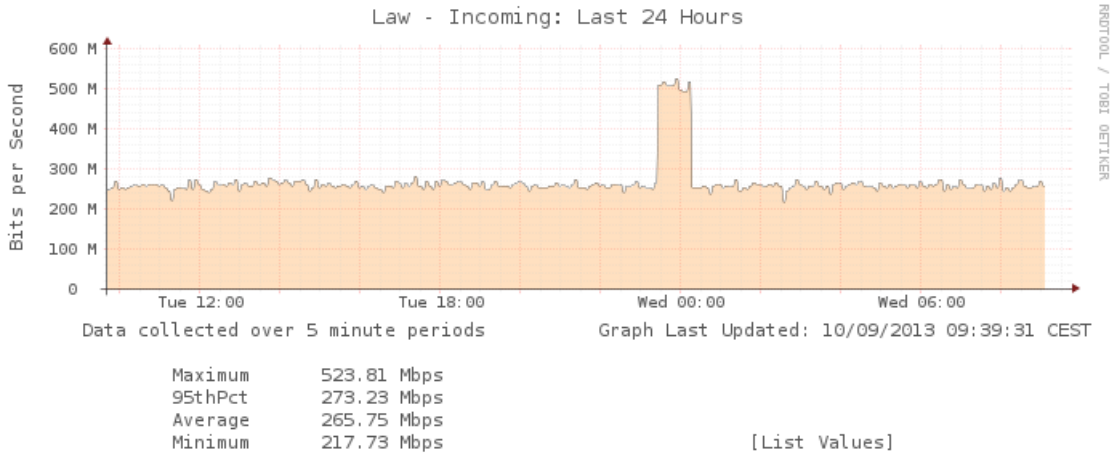

Figure 5: Network usage as reported by FlowViewer for the last 24 hours of the Milano experiment. Note that constant network usage. The peak around midnight is due to an internal data transfer.

linearly with the number of agents, and will provide the scientific community with a reliable tool to gather large data sets: this is the reason why, in the first place, the development of BUbiNG was financed in the framework of the EU-FET grant NADINE (New Algorithms for DIrected NEtworks).

Future work on BUbiNG includes integration with spam-detection software, policies for IP/host politeness throttling based on download times and site branching speed, and integration with different stores like HBase, HyperTable and similar distributed storage systems.

Acknowledgments. We thank our university for providing bandwidth for our experiments (and being patient with bugged releases). We thank Giuseppe Attardi, Antonio Cisternino and Maurizio Davini for providing the hardware, and the GARR Consortium for providing the bandwidth for the Pisa experiments. Finally, we thank Domenico Dato and Renato Soru for providing the hardware and bandwidth for the iStella experiment.

7. REFERENCES

[1] Heritrix web site. `https://webarchive.jira.com/wiki/display/Heritrix/Heritrix`.

[2] Internet archive website. `http://archive.org/web/web.php`.

[3] The clueweb09 dataset. `http://lemurproject.org/clueweb09/`, 2009.

[4] Iso 28500:2009, information and documentation - warc file format. `http://www.iso.org/iso/catalogue_detail.htm?csnumber=44717`, 2009.

[5] Ziv Bar-Yossef, Idit Keidar, and Uri Schonfeld. Do not crawl in the dust: different urls with similar text. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 111–120, New York, NY, USA, 2007. ACM.

[6] et al. Berners-Lee. Uniform resource identifier (uri): Generic syntax. `http://www.ietf.org/rfc/rfc3986.txt`, 2005.

[7] Burton H. Bloom. Space-time trade-offs in hash coding with allowable errors. *Comm. ACM*, 13(7):422–426, 1970.

[8] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[9] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the 7th International Conference on World Wide Web*, 1998.

[10] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *Selected papers from the sixth international conference on World Wide Web*, pages 1157–1166, Essex, UK, 1997. Elsevier Science Publishers Ltd.

[11] M. Burner. Crawling towards eternity: Building an archive of the world wide web. *Web Techniques*, 2(5), 1997.

[12] Jamie Callan. The lemur project and its clueweb12 dataset. Invited talk at the SIGIR 2012 Workshop on Open-Source Information Retrieval.

[13] Soumen Chakrabarti. *Mining the web - discovering knowledge from hypertext data*. Morgan Kaufmann, 2003.

[14] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[15] Jenny Edwards, Kevin McCurley, and John Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 106–113, New York, NY, USA, 2001. ACM.

[16] D. Eichmann. The RBSE spider: balancing effective search against web load. In *Proceedings of the first World Wide Web Conference*, Geneva, Switzerland, May 1994.

[17] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet L. Wiener. A large-scale study of the evolution of web pages. In *Proceedings of the Twelfth Conference on World Wide Web*, Budapest, Hungary, 2003. ACM Press.

[18] R. Fielding. Maintaining distributed hypertext infostructures: Welcome to momspider. In *Proceedings of the 1st International Conference on the World Wide Web*, 1994.

[19] Allan Heydon and Marc Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, April 1999.

[20] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In Amihood Amir and Gad M. Landau, editors, *CPM*, volume 2089 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2001.

[21] R. Khare, D. Cutting, K. Sitaker, and A. Rifkin. Nutch: A flexible and scalable open-source web search engine. *Oregon State University*, 2004.

[22] Hsin-Tsang Lee, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov. Irlbot: Scaling to 6 billion pages and beyond. *ACM Trans. Web*, 3(3):8:1–8:34, July 2009.

[23] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 141–150, New York, NY, USA, 2007. ACM.

[24] Oliver A. McBryan. GENVL and WWWW: Tools for taming the web. In *Proceedings of the first World Wide Web Conference*, pages 79–90, 1994.

[25] Maged M. Michael and Michael L. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, PODC '96, pages 267–275. ACM, 1996.

[26] Seyed M Mirtaheri, Mustafa Emre Dincturk, Salman Hooshmand, Gregor V Bochmann, Guy-Vincent Jourdan, and Iosif-Viorel Onut. A brief history of web crawlers. In *CASCON*, 2013.

[27] Gordon Mohr, Michele Kimpton, Micheal Stack, and Igor Ranitovic. Introduction to Heritrix, an archival quality web crawler. In *Proceedings of the 4th International Web Archiving Workshop (IWAW'04)*, September 2004.

[28] Marc Najork and Allan Heydon. High-performance web crawling. In James Abello, Panos M. Pardalos, and Mauricio G. C. Resende, editors, *Handbook of massive data sets*, pages 25–45. Kluwer Academic Publishers, 2002.

[29] Christopher Olston and Marc Najork. Web crawling. *Foundations and Trends in Information Retrieval*, 4(3):175–246, 2010.

[30] Brian Pinkerton. Finding what people want: Experiences with the WebCrawler. In Anonymous, editor, *Proceedings of the 2nd International World Wide Web*, volume 18(6) of *Online & CDROM review: the international journal of*, Medford, NJ, USA, 1994. Learned Information.

[31] Denis Shestakov. Current challenges in web crawling. In *ICWE*, pages 518–521, 2013.

[32] Vladislav Shkapenyuk and Torsten Suel. Design and implementation of a high-performance distributed web crawler. In *In Proc. of the Int. Conf. on Data Engineering*, pages 357–368, 2002.

[33] Jens Stoye and Dan Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. *Theor. Comput. Sci.*, 270(1-2):843–856, 2002.