Incremental Web-Site Boundary Detection Using Random Walks

Ayesh Alshukri, Frans Coenen, and Michele Zito

Department of Computer Science,
University of Liverpool, Ashton Building,
Ashton Street, L69 3BX Liverpool, UK
{a.alshukri,coenen,michele}@liverpool.ac.uk

Abstract. The paper describes variations of the classical k-means clustering algorithm that can be used effectively to address the so called Web-site Boundary Detection (WBD) problem. The suggested advantages offered by these techniques are that they can quickly identify most of the pages belonging to a web-site; and, in the long run, return a solution of comparable (if not better) accuracy than other clustering methods. We analyze our techniques on artificial clones of the web generated using a well-known preferential attachment method.

Keywords: Web Site Boundary Detection, Random Walk Techniques, Web Site Clustering.

1 Introduction

Web-site Boundary Detection (WBD) is an important activity [8,21,22] with respect to such applications as: web archiving [23] [1] , web search and information retrieval [20]. While also being important in the study and analysis of the web [5] , which includes; content authorship [8], link structure analysis [17], and web content accessibility studies [6].

The principal objective of the WBD problem is, given a start web-page, the *target (or seed, or home) web-page*, to automatically identify all pages that are related to this web-page in such a way that they can be considered to form a *web-site*. In this paper we will follow [4] in postulating that web-sites are collections of pages describing their authors intentions. Therefore they are formed by groups of web-pages that can be seen as similar to an (number of) initial seed(s). We assume that web-pages can be described by a set of measurable features. Additionally web-sites are characterized by some global structural features encoded by their hyper-link distribution. Given all this, web-site boundary detection can be reduced to a special type of clustering problem whereby the saught web-site is obtained by grouping highly connected pages described by similar feature vectors.

Such an approach carries, however, a number of issues related to the nature of the web. In particular the simplistic idea of gathering all the data first and then performing the clustering, in this context, is highly inefficient as the initial

P. Perner (Ed.): MLDM 2011, LNAI 6871, pp. 414–427, 2011.

fetching stage will then dominate the overall computation time by several orders of magnitude [14,19]. Also, it is well-known that the web is a very dynamic environment, where pages get created and deleted continuously [11,13].

In this paper we suggest that more effective results can be obtained by resorting to iterative clustering methods that do not need to have (local memory) access to the whole data set in order to start clustering the pages: the clustering process takes place while the pages are fetched from the internet. We refer to such methods as *incremental* methods. Our empirical investigation is centered on the k-means clustering heuristic. It is well known that k-means depends heavily on the sequence in which data is presented [9,12,24,25]. We claim that methods based on fetching the various web-pages by moving across the web-graph in a random manner, following the web hyperlink structure and clustering the pages on the fly, perform better than other k-means variants in the context of the WBD problem. In fact such methods, often, don't even need to explore the whole dataset to provide quite acceptable solutions for the WBD problem. We argue that such algorithms quickly identify most of the pages belonging to the web-site under consideration and, in the long run, return a solution of comparable (if not better) accuracy than other incremental variants of k-means. Our analysis is based on studying the performance of various clustering algorithms using artificial datasets generated using the preferential attachment web model proposed by Kumar's *et al.* [15]. We also argue that the properties of this model, and the resulting performance figures, suggest the possibility that our randomized clustering methods would out-perform all other k-means variants considered here, even on real web data.

The rest of the paper is organized into three broad sections. In Section 2 we define the WBD problem, both in general and in the specific setting analyzed in this paper, and review the main definitions related to Random Walks and the k-means clustering algorithm. In Section 3 we report on our experiments by describing one after the other our data sets, our performance measures, and our results. Finally Section 4 is devoted to a discussion of our results.

2 Preliminaries

All algorithms considered in this paper will work on a portion of the world-wide web. We may think of this as a graph $G = (V, E)$, where V is a collection of web-pages, and the set E keeps track of all directed links between pairs of elements of V. Without loss of generality assume that G is connected. Each page has a numerical feature vector associated with it. Technically this can be determined once the page has been downloaded from the internet. Thus, there exists a function $f : V \to \mathbb{R}^k$, for some fixed integer $k \geq 1$, such that for any $P \in V$, $f(P)$ is an ordered sequence of k real numbers characterizing the page P. In what follows we will denote the elements of V using italicized capital letters (rather than the usual $u, v, \ldots$). We will use the terms page, web-page, node, and vertex interchangeably, but in all cases we will actually be referring to the pair formed by an element of V and its corresponding feature vector. Given a

specified web-page $P \in V(G)$, our purpose is to define a set $\mathcal{W} = \mathcal{W}(P) \subseteq V(G)$ called the *web-site* of P, containing all pages of G that are similar to P.

In the work described in this paper we focus on iterative clustering processes founded on the template presented in Table 1. The graph G is assumed to model a portion of the web. For the purpose of our experiments (see Section 3.1) we may assume that it resides on some secondary memory storage device and bits of it are retrieved as needed. If the algorithms were to be used on the real web G they would be distributed at different sites across the internet and its content would have to be downloaded through http requests. Note that the well known k-means [9,12,24,25] clustering method fits this template, however the template can equally be used to represent a host of different algorithms. In the forthcoming sections we provide details about the proposed iterative clustering processes that will be considered in the rest of this paper.

Table 1. Clustering algorithm template

```
Algorithm clustering_template (P)
  W = {P}; N = {};
  set up the process internal state;
  repeat
    select a page Q from G;
    add Q to W or N;
    update the process state;
  until convergence;
  return W;
```

2.1 *k*-Means Clustering

The classical k-means clustering algorithm is obtained from the template given in Table 1 by assuming that the state of the process contains information about the (two) clusters centroids, that the pages in G are inspected one at a time in some given order (which may vary over subsequent iterations) and that the state update is only performed after a complete sweep of the data has been performed. Also, it is assumed that the graph $G = (V, E)$ is not initially available. It is retrieved from the web incrementally as different pages get requested inside the process main loop. The loop is then further repeated until the system state does not change from one sweep of the graph to the next one.

This paper considers the two variants of this process, named BF-means and DF-means, obtained by assuming that the order in which the pages of G are examined is determined by the way in which they are retrieved from the web, and is given by a fixed and bound breadth-first (resp. depth-first) traversal of the web starting at P, including all pages up to a certain bounded distance from P.

2.2 Clustering Based on Random Crawling

The careful reader will have realised that many different processes fit the template described above. The order in which the pages are considered needn't be

fixed or deterministic. The way in which the web-graph is explored can be quite arbitrary. Furthermore the process state may be updated every time a new page is considered, rather than at the end of a whole sweep of the given dataset. Finally the periodical state update might include major modification of the retrieved web-graph content. In the remainder of this section we describe two possible processes of this type: (i) Naive Random Walk (NRW) clustering and the Advanced Random Walk (ARW) clustering. The first is appealing because of its simplicity, the second will turn out to deliver the best results. Other variants were tested but results of these are omitted from this paper mainly because they are not significantly different from those of the two methods considered.

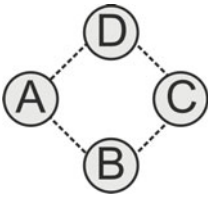

Fig. 1. Example web dataset displayed as a graph; vertices indicate web pages, dashed edges indicate hyperlinks (directions omitted so as to maintain clarity)

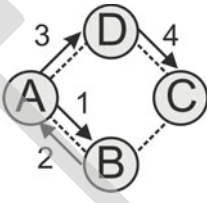

Fig. 2. Sample run of NRW (numbered directed edges indicate progress of the random walk)

Naive Random Walk (NRW) Clustering. Given an arbitrary graph G, the sequence of vertices visited by starting at a vertex and then repeatedly moving from one vertex to the next by selecting a neighbour of the current vertex at random is termed a *random walk* on the graph G (see for instance [18]). Random walks arise in many settings (e.g. the shuffling of a deck of cards, or the motion of a dust particle in the air) and there's a vast literature related to them (the interested reader is referred to the classical [10], or the very recent [2] and references therein). In particular they can be used [3] as a means for exploring a graph.

The process NRW is the variant of the clustering_process in Table 1 based on the idea of performing a random walk on G. A pure random walk is not easily simulated (at least initially) if we want to keep the constraint that the process does not require access to the full dataset to start off with. The sequence of pages to be (re-)clustered is thus given by the order in which they are visited by performing a random walk on the *known portion* of G. Initially the walk has to choose a neighbour of P. In general, given the current page Q, the page to be visited next is selected at random among those pointed by a link of Q and the set of pages seen so far that point to Q. For example, with reference to the example graph given in Figure 1, the first four steps of the process might be as shown in Figure 2, with clustering performed at every step. Note that the walk can revisit nodes multiple times, even before having completed a full sweep of G. It is therefore convenient to re-compute the centroids after each step of the walk,

rather than at the end of a sweep. It is well-known [3] that any random walk on a finite connected graph eventually visits all vertices in it. Thus, in principle, the process could run until convergence as in the standard k-means algorithm. It will turn out, however, that stopping the process after a given maximum number of "steps" (MaxIterations) is more effective and still results in good quality clusters. Some pseudo code describing the NRW technique is presented in Table 2. The decision on whether to add Q to W or the *noise cluster* N is based on the computation of the Euclidean distance between Q's feature vector and the centroids of the two clusters $W \setminus \{Q\}$ and $N \setminus \{Q\}$.

Table 2. Pseudo code for RW

```
Algorithm NRW (P)
  W = {P}; N = {};
  set Q to P; set a counter to one;
  set up the process internal state;
  repeat
    redefine Q to be a random neighbour of Q in G;
    add Q to W or N;
    increase the counter;
    update the process state;
  until counter goes past MaxIterations;
  return W;
```

Advanced Random Walk (ARW) clustering. Random walks, as defined above, are examples of so called Markov stochastic processes [10]. The evolution of a process of this type is fully determined by its current state: in the example in Figure 1, assuming the graph is completely known to the walk, every time we visit vertex A we have a 50-50 chance of moving to B or D and no chance at all to visit vertex C next.

The process NRW, strictly speaking, already breaks this framework. In this section we describe a slight more refined process (ARW) that moves even further from a pure random walk process. The process traverses the graph vertices in a random order in a similar manner as in the case of NRW. However in the case of ARW, all edges are weighted (initially such weights are all equal to one) and these weightings are reduced by a factor ϕ whenever an edge is discovered from the current vertex (the "to" vertex) to the previous vertex (the "from" vertex) where the vertex pair are not considered to be similar (according to a predetermined similarity threshold). With respect to the experiments reported later in this paper a ϕ value of 9 was used, i.e. we divided the edge weight of dissimilar edges by $\phi = 9$ each time we used one such edge. Additionally the graph structure may be changed as a result of a traversal by the addition of further edges. Immediately after visiting a page for the first time, the algorithm computes the similarity between the "new" vertex and all previously visited vertices. If the similarity is above a predefined threshold an edge is added between

the corresponding vertices. The additional edges included in this manner are then taken into account when considering the next vertex (page) to visit.

Table 3. Pseudo code for ARW

```
Algorithm ARW (P)
  W = {P}; N = {};
  set Q to P; set a counter to one;
  set up the process internal state;
  repeat
    { Secondary edges }
    if first visit to Q then
      for all nodes R seen so far do
        if R and Q are similar then
          add a "secondary edge" (R, Q) to G; set w(R, Q) to one;
        end if
      end for
    end if

    { Similarity Reduction }
    if Q and its predecessor are NOT similar then
      set w(Q,pred(Q)) to w(Q,pred(Q))/φ;
      set w(pred(Q), Q) to w(pred(Q), Q)/φ;
    end if

    redefine Q to be a random neighbour of Q in G;
    add Q to W or N;
    increase the counter;
    update the process state;
  until counter goes past MaxIterations;
  return W;
```

The effect of all this is that, as the walk progresses, a number of links are added to the graph between similar vertices and thus the walk is more likely to remain within a certain group of pages, while the reduction of edge weights between non-similar vertices will further reduce the chance of visiting unrelated vertexes in succession. Some pseudo code for ARW is presented in Table 3. In that description pred(Q) is the page visited immediately before Q. As in the case of the NRW algorithm the choice of Q's neighbours is made from among all pages R such that either (R, Q) or (Q, R) is in E.

3 Experiments

In this section we describe a number of experiments used to evaluate the processes described in Section 2.

3.1 Data Set

In a first attempt to assess the quality of our algorithms we tested them on a number of artificial data sets. Graphs $G = (V, E)$ containing a particular set of pages similar to some chosen element of V were put together by first creating artificial host graphs using the well established web-graph model proposed by Kumar *et al.* [15]. One of the generated host graphs was then selected to be the target graph. The graphs were then represented using a standard feature vector representation that included noise words randomly selected from a "bag" of noise words.

A model of the web. We will use the graph process described in [15], but with case 3. below derived from [16]. Given a positive integer m, the process generates a synthetic model of the world wide web starting from a graph $G_0^{K,m}$ having a single vertex with m links pointing to itself. Then, for $t \geq 1$, $G_t^{K,m}$ is derived from $G_{t-1}^{K,m}$ according to the following procedure (here α, β, and ν are real numbers between zero and one):

1. With probability $\alpha \times \beta$ add a new vertex to $G_{t-1}^{K,m}$ with m links pointing to itself.
2. With probability $\alpha \times (1 - \beta)$ choose a random edge in $G_{t-1}^{K,m}$ and make its source point to P.
3. With probability $(1-\alpha) \times \beta$, pick a random copying vertex Q_c; a new vertex P will point to m vertices chosen as follows:
   uar with probability ν, choose a random vertex Q and add (P, Q) to the graph.
   pa with probability $1 - \nu$, add (P, R) to the graph, where R is a random neighbour of Q_c.
4. with the remaining probability $(1 - \alpha) \times (1 - \beta)$ no new vertex is generated and a random edge is added to $G_{t-1}^{K,m}$.

In our experiments we used values of m mirroring the fact that the average page on the world wide web contains some 40-50 links [7]. Also $\alpha = 0.01$, $\beta = 0.9$ and $\nu = 0.2$. So, most of the time, new vertices will be generated to which 50 neighbours will be linked according to the procedure described in 3 above. The resulting graph shared many features with the real web, in particular: (i) in and out degree distribution, (ii) the diameter, and (iii) the presence of small bipartite "cliques". For the purpose of our experiments, we generate $G_T^{K,m}$ and then remove $G_0^{K,m}$ from it. Because of the copy mechanism by which we add edges to $G_{t-1}^{K,m}$, the single page in $G_0^{K,m}$ contains a large number of links and is linked by a large number of "younger" pages. Removing it from the graph used for our experiments makes the resulting graph, which we denote by K_T more realistic.

Generating clustery data. The use of a synthetic version of the web has many advantages. However Kumar's graphs have one disadvantage: the lack of the "clustery" nature of the real web. To complete the definition of our artificial instances we need to identify a web-site $\mathcal{W}$ within K_T (and then complete the

definition of G by adding some noise cluster, $\mathcal{N}$ to that). To this end we performed the following steps:

1. Given K_T, we picked a random node X from this graph. Such node will represent the home-page in $\mathcal{W}$.
2. To create the set of pages in $\mathcal{W}$ we then performed a breadth-first crawl of K_T starting from X, up to a certain maximum depth. The nodes visited by such process are then added to $\mathcal{W}$ with some fixed probability p.
3. The noise cluster $\mathcal{N}$ contains all nodes reachable from X that have not been added to $\mathcal{W}$ in the previous step.
4. The graph G is then defined as the subgraph of K_T induced by the set $\mathcal{W} \cup \mathcal{N}$.

For the experiments reported in this paper we generated datasets of two types, which we term A and B. Both sets were generated from copies of K_T. Sets of type A are derived from graphs generated using $m = 50$, keeping in the set $\mathcal{W}$ all nodes at a distance of at most three links from the initial page X. Sets of type B were generated using $m = 30$ and $p = 0.5$ and breadth-first exploration up to distance five. Table 4 reports some graph-theoretic statistics of the resulting graph G. Note that the two clusters have roughly the same number of elements but the class cluster $\mathcal{W}$ has many more internal links than links pointing to the noise cluster (modelling the fact that the elements of $\mathcal{W}$ represent homogeneous web-pages). Furthermore $\mathcal{W}$ is *popular* in the sense that many links from the noise cluster $\mathcal{N}$ point back to $\mathcal{W}$. There are two main differences between the graphs in set A and those in set B. First, typically graphs of type B contain fewer edges and, individually, their nodes contain fewer out-links. Second, for graphs in set A the class $\mathcal{W}$ coincides with the set of all pages at distance at most three from X (also known as the *ball of radius three around* X). For the graphs in B this is not the case. The web-site of X, although much bigger than in the case of A, is typically more "stringy". Only half of the neighbours of a vertex in $\mathcal{W}$ are in the cluster.

Given a particular graph structure G, we may generate several instances of this structure by changing the feature vectors that we associate with the various pages (vertices). In general, for any page $P \in \mathcal{W}$ (resp. in $\mathcal{N}$, the feature vector $f(P)$ can be chosen from the superposition of two k-dimensional normal multivariate distributions, having different means, $\mu_{\mathcal{W}}$ and $\mu_{\mathcal{N}}$, and equal deviations σ. The vectors associated with the elements of $\mathcal{W}$ may be chosen from the $N_k(\mu_{\mathcal{W}}, \sigma)$ distribution, those for $\mathcal{N}$ from $N_k(\mu_{\mathcal{N}}, \sigma)$. Varying the relative position of the means results in datasets of differing difficulty.

For our experiments we simulated such process in a very simple way. Given P in $V(K_T)$, $f(P)$ contains $k = 20$ integer numbers, corresponding to the number of occurrences of the elements of a pool of words S in a bag associated with P. The pool S is split in two disjoint groups of equal size, $S_{\mathcal{W}}$ and $S_{\mathcal{N}}$. The bag of P is defined by sampling, independently, k elements of S. If P is in $\mathcal{W}$, each chosen word has a chance $\pi = 0.7$ (resp. $1 - \pi$) of belonging to $S_{\mathcal{W}}$ (resp. $S_{\mathcal{N}}$), and is selected uniformly at random with replacement, from the chosen group. If $P \in \mathcal{N}$, the selection is symmetrically biased towards $S_{\mathcal{N}}$.

Table 4. Data set graph-theoretic statistics. *Intra edges* are links connecting two pages in the same cluster. *Inter edges* are links connecting pages in different clusters.

Type	Name	Cluster $\mathcal{W}$			Cluster $\mathcal{N}$		
		Size	Intra edges	Inter edges	Size	Intra edges	Inter edges
A	G1	181	2443	53	176	470	2353
	G2	124	1581	48	182	591	2335
	G3	102	978	89	128	374	1463
	G4	149	2046	77	109	259	1665
B	G1	208	4783	46	201	699	5006
	G2	262	6616	80	199	623	5570
	G3	254	5956	76	212	706	5554
	G4	293	6108	128	237	766	5500

3.2 Evaluation Criteria

We used a number of measures to compare the performance of the various clustering methods considered in this paper. Given an instance (G, f) generated as described in Section 3.1, the output (W, N) of a particular clustering algorithm $\mathcal{A}$ on (G, f) can be expressed as:

$$W = W_t \cup W_f \qquad N = N_t \cup N_f$$

where W_t (resp. W_f) is the collection of class items (noise items) that are correctly (resp. incorrectly) identified as being within the target web site, and, similarly, N_t (resp. N_f) is the collection of noise items (resp. items in W) that are correctly (resp. incorrectly) identified as noise. The *accuracy* $\gamma = \gamma(\mathcal{A}, (G, f))$ of algorithm $\mathcal{A}$ on input (G, f) is given by the expression:

$$\frac{|W_t| + |N_t|}{|\mathcal{W}| + |\mathcal{N}|}$$

which measure the proportion of correctly classified items.

We argue that in this context the accuracy only tells half of the story. Some of our clustering methods are deterministic others are randomized. As we mentioned above randomized graph crawls are only likely to visit the whole graph, eventually, but they may fail to do so, in some particular cases. Even more importantly, the clustering problems we are dealing with are asymmetric; in the sense that, one may argue, grouping together all pages in the cluster $\mathcal{W}$ is much more important than even discovering the pages in $\mathcal{N}$. Hence a (randomized) algorithm $\mathcal{A}$ may achieve poor accuracy, just because it does not see many noise pages but, on the other hand, maybe considered good because it classifies correctly all the pages in $\mathcal{W}$. We therefore resort to a second measure that focuses on $\mathcal{W}$. In what follows the *class ratio* $\omega = \omega(\mathcal{A}, (G, f))$ of algorithm $\mathcal{A}$ on input (G, f) will denote the quantity $|W_t|/|\mathcal{W}|$.

For completeness we will also track two obvious *coverage* parameters:

$$\chi_{\mathcal{W}} = \frac{|W_t| + |N_f|}{|\mathcal{W}|} \qquad \chi_{\mathcal{N}} = \frac{|W_f| + |N_t|}{|\mathcal{N}|},$$

the number of steps $\varsigma = \varsigma(\mathcal{A}, (G, f))$ (i.e. iterations of the main algorithm loop) performed and the average CPU time to complete a single step, denoted by $\theta = \theta(\mathcal{A}, (G, f))$ calculated as the total execution time to terminate a particular run divided by how many steps the algorithm took to terminate.

3.3 Results

Table 5 shows the average performance, over 50 distinct runs, of all algorithms considered in this paper on the eight datasets mentioned in Section 3.1. For completeness, Figure 3 all the way up to Figure 10 provide more detailed plots showing how the various quantities change as the algorithms run.

Table 5. Table shows the average performance of each technique, run 50 times on each graph in sets A and B (Each run has a varying word distribution of class and noise pages). Accuracies, class ratios and coverages are reported as percentages.

	Algorithm	γ	ω	χ_W	χ_N	θ	ς
A	BF-means	63.76	61.7	100	100	1.1577	1071
	BF-means (1)	56.9	53.75	100	100	1.1527	715
	DF-means	79.07	78.63	100	100	0.9596	2467
	DF-means (1)	73.18	51.38	100	100	1.3354	715
	NRW	92.76	92.09	99.74	88.67	0.0868	100000
	ARW	92.11	91.61	99.67	70.32	0.1060	100000
B	BF-means	71.08	67.84	100	100	1.4275	1591
	BF-means (1)	64.06	56.71	100	100	1.9449	1061
	DF-means	89.24	86.8	100	100	1.6542	1591
	DF-means (1)	83.36	76.68	100	100	2.3369	1061
	NRW	87.38	83.57	97.85	63.29	0.1085	100000
	ARW	92.05	87.85	95.88	45.90	0.1308	100000

In our experiments we compared the two random walk methods (NRW, ARW) described in Section 2.2 with two versions of each of the deterministic variants of k-means (BF,DF-means) described in Section 2.1: in each case we considered the full process as well as a version that only performs one scan of the data (denoted in table by (1) in Table 5). The latter has been included to show, in a sense, the quality of the solution produced by the given clustering algorithm if one relaxes its termination condition. A process that can only "see" each page once is faster than the fully-fledged k-means heuristic, but returns much poorer solutions.

The results in the table show that in most cases the methods based on random walks perform, on average, better than the deterministic ones. In most cases they provide better accuracy and class ratio. The results for such methods are

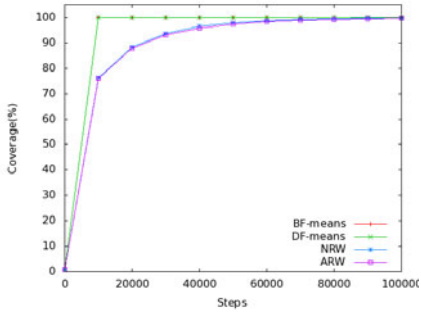

Fig. 3. Set A: Class Coverage of NRW, ARW, BF-means and DF-means

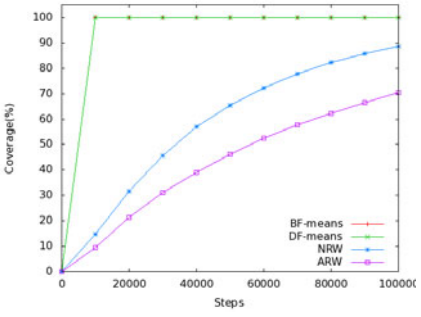

Fig. 4. Set A: Noise Coverage of NRW, ARW, BF-means and DF-means

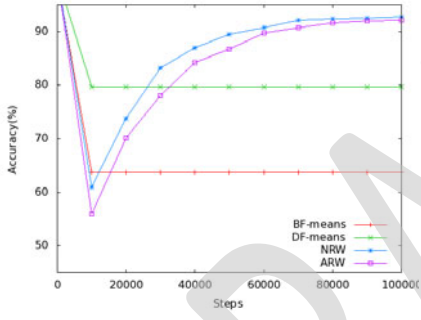

Fig. 5. Set A: Accuracy of NRW, ARW, BF-means and DF-means

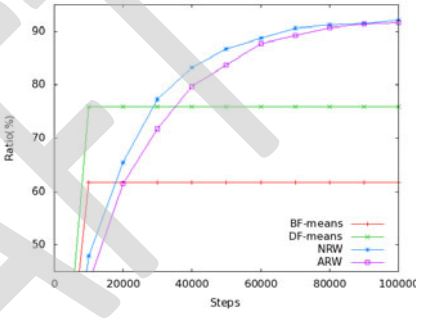

Fig. 6. Set A: Class Ratio of NRW, ARW, BF-means and DF-means

particularly good on graphs in set A, and even on the "stringy" datasets of type B, which are easier for BF-means and DF-means, they perform reasonably well. Figures 6 and 10 provide even more interesting results for the class ratios: both random walk based heuristics cover more than 70% of the "good" pages in less than 50000 steps.

Of course one may object that this comes at a price. Table 5 clearly indicates that the randomized algorithms in question struggle to visit all pages in the given data set. Furthermore, the random walk methods seem hugely slower than the deterministic ones. All deterministic methods complete their processing on average in less than 3000 steps, whereas all randomized processes considered are run for 100000 steps. However we argue that, for the problem at hand, these issues may not be very important. In the context of web-site boundary detection the main aim is to identify a web-site containing a particular homepage. It may thus be quite satisfactory to get a clustering that correctly classifies 95% of the "good" web-pages even if 40% of the noise pages are not visited at all. The relatively high running times are also not too meaningful, as the average running time per step (parameter θ in Table 5) gives a better indicator of the overall run time.

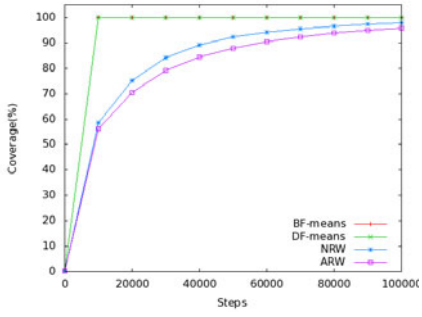

Fig. 7. Set B: Class Coverage of NRW, ARW, BF-means and DF-means

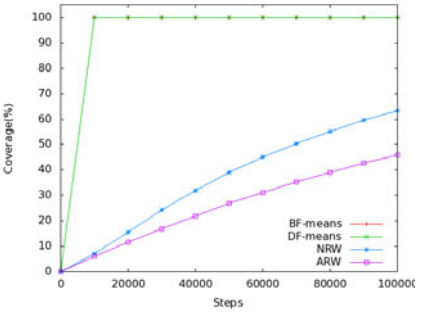

Fig. 8. Set B: Noise Coverage of NRW, ARW, BF-means and DF-means

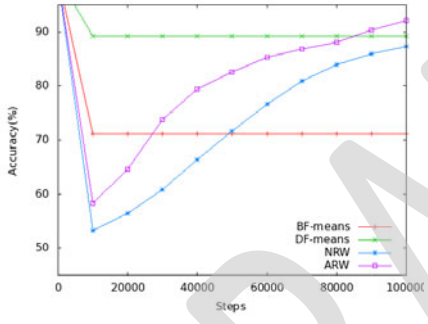

Fig. 9. Set B: Accuracy of NRW, ARW, BF-means and DF-means

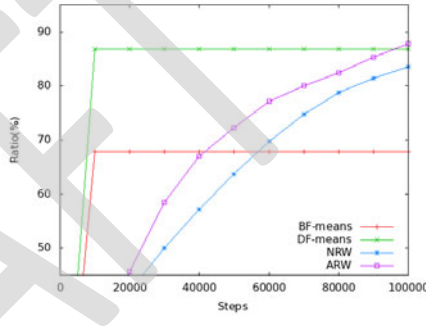

Fig. 10. Set B: Class Ratio of NRW, ARW, BF-means and DF-means

4 Discussion

The results in Section 3.3 show that, at least on the artificial data used here, simple randomized methods that combine k-means clustering abilities with graph exploration methods that carefully use the graphical structure of the data outperform a clustering method that does not take the graph structure into consideration. In particular the overall best method is ARW, which performs consistently well across the two sets. The best performing deterministic method was DF-means.

In set A the structure of the graph is such that web-pages associated with the class are located at a certain distance (number of links) from the start page, while noise pages are located beyond this distance . This type of graph is more difficult for BF-means in particular to cluster, as the initial conditions of the walk are affected by the amount of class items that are seen without any relativity to noise items. Noise items are not seen till later in the walk, after all class items have been seen. This also effects DF-means, but that process has a chance to recover slightly by walking to a deeper depth to visit some noise items

before backtracking, which then in turn helps the clustering make the distinction between class and noise items earlier in the walk.

In set B however, the structure is such that there is a mixture of noise and class items. When BF and DF-means crawl the graphs in the initial stages, this mixture helps improve the initial conditions for the clustering in both cases. As a distinction can be made between class and noise earlier in the walk, thus this improves the overall performance of DF-means.

Looking at the performance of ARW on both set A and B, it can be seen from the history of the walk there is some consistencies between the two sets. For example, the coverage is quite consistent between both graphs (Set A: figs 3 & 4. Set B: figs 7 & 8). The coverage of class and noise is steady in both cases. What this shows is that ARW is much less susceptible to variation in the graph structure that would normally effect performance, as is shown by the poor performance of BF-means and DF-means on set A in contrast to set B. The performance of ARW is more consistent for both set A and B. Massive differences in accuracy or class ratio are not seen when comparing the performance of the random walk methods on both sets.

References

1. Abiteboul, S., Cobéna, G., Masanes, J., Sedrati, G.: A first experience in archiving the french web. In: Agosti, M., Thanos, C. (eds.) ECDL 2002. LNCS, vol. 2458, pp. 1–15. Springer, Heidelberg (2002)
2. Aldous, D., Fill, J.: Reversible markov chains and random walks on graphs. Monograph in preparation (2002)
3. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovasz, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: Proceedings of the 20th Annual Symposium on Foundations of Computer Science, pp. 218–223. IEEE Computer Society, Washington, DC, USA (1979)
4. Alshukri, A., Coenen, F., Zito, M.: Web-Site Boundary Detection. In: Perner, P. (ed.) ICDM 2010. LNCS, vol. 6171, pp. 529–543. Springer, Heidelberg (2010)
5. Bharat, K., Chang, B.-W., Henzinger, M.R., Ruhl, M.: Who links to whom: Mining linkage between web sites. In: Proceedings of the 2001 IEEE International Conference on Data Mining, ICDM 2001, pp. 51–58. IEEE Computer Society, Washington, DC, USA (2001)
6. Broder, A.Z., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomikns, A., Wiener, J.: Graph structure in the web. Computer Networks 33(1-6), 309–320 (2000)
7. Broder, A.Z., Najork, M., Wiener, J.L.: Efficient url caching for world wide web crawling. In: Proceedings of the 12th International Conference on World Wide Web, pp. 679–689. ACM, New York (2003)
8. Dmitriev, P.: As we perceive: finding the boundaries of compound documents on the web. In: Proceeding of the 17th International Conference on World Wide Web, WWW 2008, pp. 1029–1030. ACM, New York (2008)
9. Dunham, M.H.: Data Mining: Introductory and Advanced Topics. Prentice Hall PTR, Upper Saddle River (2002)
10. Feller, W.: Introduction to probability theory and its applications, vol. 1. WSS (1968)

11. Gomes, D., Silva, M.J.: Modelling Information Persistence on the Web. In: 6th International Conference on Web Engineering, pp. 193–200. ACM Press, New York (2006)
12. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco (2001)
13. Koehler, W.: Web page change and persistence A four-year longitudinal study. Journal of the American Society for Information, 162–171 (2002)
14. Kroeger, T.M., Long, D.D.E., Mogul, J.C.: Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems Monterey, p. 2 (December 1997)
15. Kumar, R.: Trawling the Web for emerging cyber-communities. Computer Networks 31, 1481–1493 (1999)
16. Kumar, R., Raghavan, P., Rajagopalan, S., Sivakumar, D., Tomkins, A., Upfal, E.: Stochastic models for the Web graph. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, pp. 57–65. IEEE Computer Society, Washington, DC, USA (2000)
17. Liu, B.: Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data. Springer, Heidelberg (2007)
18. Lovász, L.: Random walks on graphs: A survey. Combinatorics Paul Erdos is Eighty 2, 1–46 (1994)
19. Padmanabhan, V.N., Mogul, J.C.: Using predictive prefetching to improve World Wide Web latency. ACM SIGCOMM Computer Communication Review 26 (July 1996)
20. Pokorn, J.: Web Searching and Information Retrieval. Computing in Science and Engineering 6(4), 43–48 (2004)
21. Rodrigues, E.M., Milic-Frayling, N., Fortuna, B.: Detection of Web Subsites: Concepts, Algorithms, and Evaluation Issues. In: IEEE/WIC/ACM International Conference on Web Intelligence, pp. 66–73. IEEE Computer Society, Los Alamitos (2007)
22. Schneider, M.S., Kirsten, F., Michele, K., Gina, J.: Building thematic web collections: challenges and experiences from the september 11 web archive and the election 2002 web archive. In: Digital Libraries, ECDL, pp. 77–94 (2003)
23. Senellart, P.: Identifying Websites with Flow Simulation. In: Lowe, D.G., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 124–129. Springer, Heidelberg (2005)
24. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Pearson International Edition (2006)
25. Witten, I.H., Frank, E.: Data Mining: practical machine learning tools and techniques. Morgan Kaufman, San Francisco (2005)