Practical Representations for Web and Social Graphs

Francisco Claude
University of Waterloo
Waterloo, Canada
fclaude@cs.uwaterloo.ca

Susana Ladra
Universidade da Coruña
A Coruña, Spain
sladra@udc.es

ABSTRACT

In this paper we focus on representing Web and social graphs. Our work is motivated by the need of mining information out of these graphs, thus our representations do not only aim at compressing the graphs, but also at supporting efficient navigation. This allows us to process bigger graphs in main memory, avoiding the slowdown brought by resorting on external memory.

We first show how by just partitioning the graph and combining two existing techniques for Web graph compression, k^2-trees [Brisaboa, Ladra and Navarro, SPIRE 2009] and RePair-Graph [Claude and Navarro, TWEB 2010], exploiting the fact that most links are intra-domain, we obtain the best time/space trade-off for direct and reverse navigation when compared to the state of the art. In social networks, splitting the graph to achieve a good decomposition is not easy. For this case, we explore a new proposal for indexing MP_K linearizations [Maserrat and Pei, KDD 2010], which have proven to be an effective way of representing social networks in little space by exploiting common dense subgraphs. Our proposal offers better worst case bounds in space and time, and is also a competitive alternative in practice.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Graphs and networks; E.4 [**Coding and Information Theory**]: Data compaction and compression; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Experimentation, Performance

1. INTRODUCTION

Graphs are a natural way of modeling connections among Web pages in a network or people according to a criteria like friendship, co-authorship, etc. Many algorithms that compute and infer interesting facts out of these networks work directly over these graphs. Some examples of this are Connected components, HITS [16], PageRank [21], spam-detection[22], among others. In social networks, graph mining also enables the study of populations' behavior. Successful graph mining not only enables segmentation of users but also prediction of behavior. Link analysis and graph mining remains an area of high interest and development.

These human-generated graphs are growing at an amazing pace, and their representation in main memory, secondary memory, and distributed systems are getting more and more attention. Furthermore, space-efficient representations for these graphs have succeeded at exploiting regularities that are particular to the domain. In the case of Web graphs the main properties exploited are the locality of reference, skewed in/out-degree, and similarity of adjacency lists among nodes of the same domain. In social networks, there is a tendency towards forming cliques in the network.

Hence, representing, compressing and indexing graphs become crucial aspects for the performance and in-memory processing when mining information from those graphs. In this work we focus on obtaining space-efficient in-memory representations for both, Web graphs and social networks. We follow the same notation across this paper. When talking about a graph $G = (V, E)$ we call $n = |V|$ the number of nodes and $m = |E|$ the number of edges.

For representing Web graphs, we present a structure that successfully combines the k^2-tree, proposed by Brisaboa et al. [7], and the RePair representation proposed by Claude and Navarro [11]. This new structure splits the graph in $t+1$ pieces, the first t ones correspond to sub-graphs formed by groups of domains. The last piece contains all the edges that point from one of the t sub-graphs to another one. This combination allows us to obtain the best cases for the k^2-tree, since most of the compression is gained inside domains and the query time is good when the matrix is dense.

Our second structure proposed corresponds to a new indexing technique for the MP_K linearization proposed by Maserrat and Pei [20]. This new indexing technique improves the time to obtain the in/out-neighbors of a node v from $O(\sum_{u \in N_v} deg(u))$ [1] to $O(\lg \lg n)$ per element retrieved, while obtaining a space that is bounded by the previous indexing proposal [20] plus lower order terms.

The experimental results show that for Web graphs we obtain a structure that improves upon the state of the art, achieving the best trade-off when we require both, direct and reverse navigation. In the domain of social networks,

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

[1] N_v corresponds to the neighbors of v and $deg(u)$ to the number of incoming/outgoing links to/from u.

our proposal improves upon previous results showing that it constitutes a competitive index for social networks. An interesting property of this last structure is that any practical advances in *rank/select* indexes for large alphabets [13, 9] would improve our representation. Our implementations are available at `http://webgraphs.recoded.cl/`.

2. RELATED WORK

In this section we describe the related work on Web graph and social networks representations, focusing on compact solutions that achieve little space while retaining basic navigational functionality. The first subsection covers Web graphs and presents in more detail the k^2-tree [7]. The next subsection presents results on social networks and relevant details about the representation proposed by Maserrat and Pei [20].

2.1 Compact Representations of Web Graphs

The best trade-off, for many years, was offered by the proposal of Boldi and Vigna [5]. They presented a highly engineered framework for managing Web graphs, exploiting the regularities of Web graphs, and at the same time offering fast access to any adjacency list. Later, Boldi et al. [4] proposed an improved method based on their original method. An alternative that permits a fast access to direct lists was proposed by Apostolico and Drovandi [1]. This representation depends only on their topology of the graph.

Another proposal, competing in the time/space trade-off, is the one by Claude and Navarro [11]. This approach represents the graph as a text, and then, by applying grammar based compression, they achieve a representation whose space consumption is similar to the one by Boldi and Vigna while the access time is considerably reduced. The particular method used is an approximate version of the well known Re-Pair compression method [11]. Claude and Navarro extended their work to support direct and reverse navigation [10]. This was later improved by Brisaboa et al. [7], who represent a binary matrix by using a succinct representation for trees [15] for a special kind of tree inspired in quad-trees. Their structure, called k^2-tree, achieves the smallest space reported so far when representing both graphs, direct and reverse. During the rest of this section, we focus on k^2-trees, providing information required in the rest of the article.

The k^2-tree method represents the adjacency matrix of the graph using a non-balanced k^2-ary tree. The method consists of a subdivision of the adjacency matrix into k rows and k columns of sub-matrices of size n^2/k^2. Each of the resulting k^2 sub-matrices will be a child of the root node and its value will be 1 iff in the cells of the sub-matrix there is at least one 1. A 0 child means that the sub-matrix has all 0s and hence the tree decomposition ends there. Once the level 1, with the children of the root, has been built, the method proceeds recursively for each child with value 1.

This construction allows us to represent the whole adjacency matrix with just two bit arrays: T (tree) and L (leaves). T stores all the bits of the k^2-tree except those in the last level. The bits are placed following a level-wise traversal. L stores the last level of the tree. Additionally, an auxiliary structure is created over T that allows the navigation through the compact representation of the tree.

To find the direct (reverse) neighbors of a node of the graph, the k^2-tree needs to locate which cells in a certain row (column) of the adjacency matrix have a 1. This is done with a top-down traversal that can be performed efficiently over the compact representation of the tree, requiring $O(1)$ time per node traversed in the tree.

While alternative compressed graph representations are limited to retrieving direct, and sometimes reverse, neighbors of a given node, the k^2-tree allows for more sophisticated forms of retrieval, such as range searches or checking whether two Web pages are connected or not without extracting the adjacency lists of the pages involved in the query.

2.2 Compact Representation of Social Networks

A Web graph can be regarded as an special case of social network, which describes a network of interconnected pages. Most Web graph compression techniques exploit properties that are exclusive from this particular class, and thus they do not extend well for other social networks.

There has been several attempts at approximating this reordering [4, 8], which turns to be NP-hard. The best result with this approach corresponds to the one by Chierichetti et al. [8]. They extend Boldi and Vigna's WebGraph framework [5] to compress social networks. They exploit the reciprocity exhibited in social networks to improve compression, using a scheme called *backlink* compression, and sort the nodes following a shingle-based ordering using the Jaccard coefficient to measure the similarity between nodes.

Another line is to exploit regularities such as clustering in the graph. One of the best results achieved, proposed by Maserrat and Pei [20], follows this approach. They propose a transformation called MP_K linearization and show how to index them in order to support direct and reverse navigation. Given a graph $G = (V, E)$, the MP_K linearization corresponds to a sequence S of node identifiers that satisfies the following property: For any $e = (u, v) \in E$, there exists a substring of length K in S, such that u and v appear in it.

This sequence, augmented with $2K$ bits per item, is enough to represent the whole graph. The structure proposed by Maserrat and Pei [20] works by storing this $2K$ bits per item in plain form, we call this string N, and storing an alternative sequence S_2, where $S_2[i] = p$ iff $p = \max(\min\{j | S[i] = S[j], j > i\}, \min\{j | S[i] = S[j], j \geq 0\})$. They add dummy nodes at the beginning in order to determine when a traversal of the list of occurrences of a given node finishes. Also, these dummy nodes allow us to start a traversal for a given node in $O(1)$. S is not represented.

3. K²-PARTITIONED

The main proposal of this paper consists in a simple combination of two existing Web graph compression techniques, the approach by Brisaboa et al. [7] and the one by Claude and Navarro [10, 11], which obtains striking practical space/time results, as we show in Section 5.

In order to combine both techniques, we partition the graph into $t+1$ pieces, $G_1, G_2, \ldots, G_{t+1}$. The first t of them correspond to non-overlapping squares along the diagonal of the binary matrix defining the graph. The last partition corresponds to all the ones outside the region defined by $G_1 \cup G_2 \ldots \cup G_t$.

As a simple example, consider a graph with t domains. A valid partition would consider every domain as defining a square in the matrix. The last graph generated contains all the links to external domains in the graph. In fact, this is what the partition scheme aims at, yet we want to be able

to join small domains in order to amortize the extra cost of representing a separate graph.

This partition has many interesting advantages over handling the original graph. First, the representation of Brisaboa et al. works faster over smaller graphs, since the height of the tree is smaller; in addition, if the sub-graphs are roughly defined by domains, we do not expect to follow many empty paths through the trees. Besides being faster, we do not expect to lose too much compression, since most of the properties exploited are among nodes in the same domain, and we will enforce complete domains to land on the same sub-graph. Finally, we expect the last sub-graph (including all links not considered in the other sub-graphs) to be sparse, and thus not really an issue in terms of space.

3.1 Partitioning the Graph

We partition the graphs guided by the domains. The partition procedure receives a parameter M, and generates the splitting in a greedy fashion. It starts adding the alphabetically smaller domain to the current sub-graph, and continues adding domains to this partition, in alphabetic order, until at least M nodes are contained in the sub-graph (the size of the square is at least M^2). Then it creates a new sub-graph, and continues with the next domain in alphabetical order. Each of the sub-graphs generated by this process is stored independently, and then, all remaining edges are added to a matrix with the same size of the original graph ($n \times n$). This graph corresponds to G_{t+1}.

3.2 Internal Links

For the representation of the internal links we use the k^2-tree representation presented in [17]. It consists in a modification of the original algorithm where the last level of the k^2-tree is not represented in plain form, but compacted using the variable-length encoding scheme *Directly Addressable Codes* (DACs) [6]. This enables fast direct access to any leaf sub-matrix represented at the last level of the k^2-tree.

More concretely, we create a k^2-tree per each sub-graph G_j, $1 \leq j \leq t$, with the same parameters of k_L, where k_L is the k used in the last level. Then, we create a common vocabulary with the different $k_L \times k_L$ leaf sub-matrices that appear in the last level of the t partitions and compute their frequencies. We sort the vocabulary by frequency, and assign an integer value to each $k_L \times k_L$ sub-matrix: shorter integers are assigned to more frequent sub-matrices. Then, we compress the last level of each k^2-tree by representing the integers associated to those sub-matrices using DACs, which have been proved successfully for this scenario.

Hence, the representation of the internal links consists of an array of t k^2-trees, one per each sub-graph G_j, with $j \leq t$; the common vocabulary of leaf sub-matrices; and an array containing the offsets for each partition, that is, the node identifier of the first node of each sub-graph.

When a query involving internal links must be solved, a binary search is performed in the array of offsets to determine the sub-graph where the sought node is located, and then the query is translated to the corresponding k^2-tree.

Given that the k^2-tree plays the main role in representing the links of the graph, since most links are intra-domain, we call our combination k^2-partitioned.

3.3 External Links

We considered two representations for the graph representing the external links, both based on the approximate version [11] of the Re-Pair compression method [18].

The first one corresponds to the one proposed by Claude and Navarro [10], that supports direct and reverse navigation. The second is a simple modification of the original proposal by Claude and Navarro [11], that compresses both, the direct and the reverse graph, separately, but sharing the vocabulary among them, saving space over the naive representation that compresses both graphs independently.

The way this second approach works is the following: We compress the graphs $G_{t+1} = ([n] = \{1, 2, \ldots, n\}, E)$ and $G^R_{t+1} = ([n], E^R)$ by compressing the fictitious graph $G'_{t+1} = ([2n], E \cup S(E^R))$, where $S(E^R) = \{(n+i, j) | (i, j) \in E^R\}$. This allows us to compress both graphs at the same time, and generates only one dictionary, thus saving space. The query for direct neighbors does not change, and the reverse neighbors can be obtained by adding n to the node identifier and querying the direct neighbors in G'_{t+1}.

In our experiments we will use the second version, which proved to be faster and did not add much extra space.

4. INDEXING $\mathrm{MP_K}$ LINEARIZATIONS

As we have said, Web graph compression methods can be used for compressing social networks. However, the proposed method, k^2-partitioned, cannot be as successfully applied to social networks, since it is not usually possible to separate intra-host and inter-host links as clearly as for Web graphs. Hence, in this paper we study the MP_K linearizations to represent social networks.

It is a simple exercise to replace the pointers to the next occurrence of each symbol in the MP_k linearization by an index for sequences supporting rank, select and access.

The *rank* operation for symbol v and position p counts the number of occurrences of v in S up to position p (included). The *select* operation for a symbol v and position j computes the position where the j-th occurrence of v appears in S.

Using rank and select we can iterate over the occurrences of v in S by using $select_S(v, j)$ for $j = 1 \ldots rank_S(v, |S|)$. If we use the structure proposed by Barbay et al. [2], retrieving each position where v occurs takes $O(1)$ time and accessing a position takes $O(\lg \lg n)$ time.

To retrieve the reverse neighbors, a similar approach is valid, but we need to check the $2K$ adjacent positions to each occurrence of v to determine whether they point to v or not.

There are many proposals we can use to represent S offering different time/space trade-offs [2, 13, 19, 12, 14]. We can also improve the N_v term by paying $\lg(3)$ extra bits per symbol in S, by storing three different versions for the identifier of each node. This allows us to scan just the nodes relevant to each query and the time for direct neighbors becomes $O(\lg \lg n)$ per element retrieved. The time for reverse neighbors becomes $O(K \lg \lg n)$ per element retrieved. Note that we always know which part of the field in N we want to access in order to check, so we can always check if a node points to us in $O(1)$ given that we know both positions. When K is smaller than the word size w ($w = \Theta(\log n)$), we can do slightly better using tables to obtain the pointers to our position when retrieving the reverse neighbors, the $O(K)$ factor becomes $O(1 + K^2 / \lg n)$.

Table 1: Description of the Web graphs used.

File	Pages	Links	Size (MB)
EU (2005)	862,664	19,235,140	77
UK (2002)	18,520,486	298,113,762	1,208
Arabic (2005)	22,744,080	639,999,458	2,528

THEOREM 1. *If an MP_K linearization (S, N) is represented with the string representation proposed by Barbay et al [2], we obtain an index that allows us to retrieve the direct neighbors of a node in $O(N_v \lg\lg n)$ time. The reverse neighbors can be computed in $O(N_v K \lg\lg n)$ time. The space requirement is $|S|H_0(S)+o(|S|)(1+H_0(S))+2K|S|$ bits, where H_0 corresponds to the zero-order entropy. By using $O(|S|)$ extra bits we can obtain $O(\lg\lg n)$ time per element retrieved for direct neighbors and $O(\min(K, 1+K^2/\lg n)\lg\lg n)$ time per element for the reverse ones.*

5. EXPERIMENTAL RESULTS

5.1 Experiments on Web Graphs

We ran several experiments over some Web graphs from the *WebGraph* project, some of them gathered by UbiCrawler [3]. These data sets are made available to the public by the members of the *Laboratory for Web Algorithmics*[2] at the *Univerita Degli Studi Di Milano*. Table 1 gives the main characteristics of the graphs used. The last column gives the size of a plain adjacency list representation of the graphs (using 4-byte integers).

The machine used in our tests is a 2GHz Intel®Xeon® (8 cores) with 16 GB RAM, and 580 GB Disk (SATA 7200rpm). It runs Ubuntu GNU/Linux with kernel version 2.6.24-28-server SMP (64 bits). We used the gcc compiler version 4.2.4 with `-O9` optimizations. The space achieved by our structures is measured in bits per edge (bpe), dividing the total space of the structure by the number of edges (i.e., links) in the graph. Time results measure average CPU user time per neighbor retrieved: We compute the time to search for the neighbors of all the web pages of the graph (in random order) divided by the total number of links retrieved.

We show in Section 5.1.2 the results of our technique, which outperforms the rest of the methods of the state of the art. These improved results are mainly caused by the partitioned representation of the internal links of the graphs. Hence, we first show some partial results when indexing only intra-host links, which helps us understand the benefits of our proposal.

5.1.1 Internal Links

In this section we show the advantages of representing the internal links separated by domains, using one k^2-tree for each sub-graph instead of using the original k^2-tree representation over the whole adjacency matrix of the web graph. Hence, we compare the compressed representation of the complete web graph obtained by the k^2-tree technique against the compressed representation of just the internal links using the proposed k^2-partitioned approach.

Table 2 illustrates the compression properties obtained by the two techniques, "k^2-tree all" and "k^2-partitioned" over the two larger web graphs described, using $M = 10000$ for the partition of Arabic and $M = 1000$ for UK. We also include a row for each web graph, "k^2-tree inlinks", which shows the time performance when we use the original k^2 technique representing the whole graph to retrieve just the internal links. This can be simulated by querying the neighbors of each web page within the range of page identifiers corresponding to its same domain, hence, we report the times to answer that range query for each web page.

Notice that we are comparing the space results for the original k^2-tree, which represents all the links of the web graph, with the results for k^2-partitioned when representing just the internal links. These intra-domains links constituted around the 90% of the total number of links depending on the graph. The space required by k^2-partitioned to represent those internal links is around 80-90% the space required by the original k^2-tree technique to represent the complete set of links. However, to compute the total space required by k^2-partitioned we must add the space of the external links, which may be less compressible than internal links. This causes that the total compression for k^2-partitioned approach will be close to the compression of the k^2-tree technique but not better, as we will see in Section 5.1.2.

The strength of k^2-partitioned approach consists in representing the internal links by building a separate k^2-tree for each sub-graph, as they become shorter. Moreover, it solves the main problem of k^2-tree approach, where the branches generated by external links may cause several unsuccessful top-down traversals over the tree. This behavior can be observed in Table 2. The average time to retrieve the internal links of a page is considerably faster for k^2-partitioned than retrieving all the links of a page using the original k^2-tree. The times obtained are around 5-20 times faster, as the average number of nodes traversed during the retrieval of the neighbors lists is one order of magnitude larger when using the original k^2-tree than when using k^2-partitioned. k^2-partitioned also obtains better times than retrieving just the internal links from the original k^2-tree.

Hence, excluding external links from the k^2-tree representation decreases the number of traversed nodes significantly, and thus the retrieval time. The experiments demonstrate the imperativeness of storing the external links separately.

5.1.2 Overall Representation

We now compare our combined representation, k^2-partitioned, with the most competitive solutions of the state of the art. Note that in this section we are indexing and searching the complete Web graph, using a k^2-tree approach for the intra-host links and a RePair approach for the inter-host links.

Figure 1 shows the space/time trade-off for retrieving direct neighbors over different graphs (reverse neighbors behave similarly). We measure the average time efficiency in μs/edge as before. We compare our compact representation, k^2-partitioned, with the original k^2-tree and the fastest proposal in [10] that computes both direct and reverse neighbors (`Re-Pair GMR`), as well as the original representation in [11] (`Re-Pair (dir+rev)`). A variant of Re-Pair GMR labeled `Re-Pair GMR (2)` is also included, where *access* operation is solved in constant time and *select* in time $O(\log\log n)$. Thus, `Re-Pair GMR` is faster for reverse neighbors (using constant-time select), and `Re-Pair GMR (2)` is faster on direct neighbors (using constant-time access). We also include

[2]http://law.dsi.unimi.it/

Table 2: Space time results of k^2-partitioned compared with k^2-tree method (just internal links).

Graph	Method	Links (%)	Space (bpe)	Time (μs/e)	Avg. trav. nodes
UK	k^2-tree all	100%	2.78	14.031	1,527.65
	k^2-tree inlinks	92.51%	-	1.174	71.64
	k^2-partitioned	92.51%	2.27	0.749	65.49
Arabic	k^2-tree all	100%	2.47	6.163	1,193.30
	k^2-tree inlinks	89.72%	-	1.217	139.78
	k^2-partitioned	89.72%	2.11	0.817	110.64

the `WebGraph (dir+rev)` alternative from Boldi and Vigna, using version 2.4.2 and the variant strictHostByHostGray, and Apostolico and Drovandi [1] `AD(dir+rev)`, version 0.2.1. In these two techniques, both the graph and its transpose are represented in order to achieve reverse navigation as well. Notice that `AD(dir+rev)` technique performs a reordering of the node identifiers based on the Breadth First Search (BFS) of the graph instead of the lexicographic order. This permutation of identifiers is not accounted for in the space results reported next. However, this mapping should be stored if we want to recover the graph with the original node identifiers.

We can observe that our new representation k^2-partitioned competes successfully with the other compression methods of the literature, especially for larger graphs. It achieves very compact spaces, smaller than the rest of the techniques except for the k^2-tree, which can obtain slightly better spaces at the expense of degrading its time efficiency in orders of magnitude. Hence, our proposed technique achieves the best space/time trade-off for Web graph representation when direct and reverse navigation are required.

5.2 Experiments on Social Networks

In this section we focus on compressing social networks. As k^2-partitioned makes heavy use of the `URL`s of the nodes, something non-existing in social networks, we consider the standard k^2-tree as a baseline for comparing space results.

The method implemented by Maserrat and Pei [20] is not available, thus we implemented our own linearization heuristic. Ours is much simpler than theirs, so we do not obtain the same numbers, still the index shows that it can compete against the other indexing method while offering a time guarantee. The heuristic we used to generate the MP_K linearization corresponds to simply adding the node with more in/out-going links from the last K nodes added so far. In case of a tie, we just chose one node at random. The value of K remains constant through the whole process.

Table 3 shows the social networks we used for our experiments. In Figure 2 we present the time/space tradeoff for our implementation, and we include the results for the WebGraph framework, `AD`, and k^2-trees. For the WebGraph framework, we add the space of the direct and reverse graphs compressed with standard parameters. In order to compare with the previous indexing method for MP_K linearizations, we implemented the index as proposed in [20]. Figure 2 includes these results as `MPk original`. We also include the results for the WebGraph framework, Apostolico and Drovandi technique and the k^2-trees.

We used wavelet trees without pointers [14, 9] to represent the *rank* and *select* data structure[3]. The time per element retrieved is around 8.5 to 20.33 microseconds. This is comparable with other Web graph representations. When compared to the original indexing for MP_K linearizations, we offer a new tradeoff where our version is slower in practice, but also smaller. For `Ljournal-2008` our approach is 4.96 times slower than the original proposal. Smaller ones are better exploited by the original indexing, since a *rank/select* operation does require more work than just traversing short lists; results for this case omitted due to lack of space.

[3]Available in libcds at http://libcds.recoded.cl

Crawl	Nodes	Edges	K
Web-Stanford	281,904	2,312,497	8
Web-Google	916,428	5,105,039	7
Ljournal-2008	5,363,260	79,023,142	4

Table 3: Data sets used for MP_k linearization.

6. CONCLUSIONS

We have shown two new compressed graph representations. The first one, k^2-partitioned, designed to represent Web graphs, provides the best space/time trade-off when compared to the state of the art on real crawls. Its simplicity contrasts with the remarkable results obtained. Our second proposal achieves competitive space compared to the results from Maserrat and Pei [20], and provides a time bound when retrieving an adjacency list that depends only on properties of the query node itself.

Both methods are designed to work in main memory, and they do not offer good access locality, thus are not recommendable for secondary memory. As previously mentioned [11], these compressed representations can also improve distributed systems from an economical point of view, since their smaller memory footprint allows us to represent the same data using fewer computers. In the case of k^2-partitioned, this is particularly interesting, since the partition of the graph could allow different computers to handle different domains, obtaining a natural division. This, of course, depends on the access patterns required by the system, when it is desirable to distribute the load across all computers.

Acknowledgments

The authors would like to thank Nieves Brisaboa, Ian Munro and Gonzalo Navarro for useful comments.

This work was funded in part by NSERC and David R. Cheriton Scholarships program (Francisco Claude), and by Ministerio de Ciencia e Innovación (grants TIN2009-14560-C03-02 and CDTI CEN-20091048) and Xunta de Galicia (grant 2010/17) (Susana Ladra).

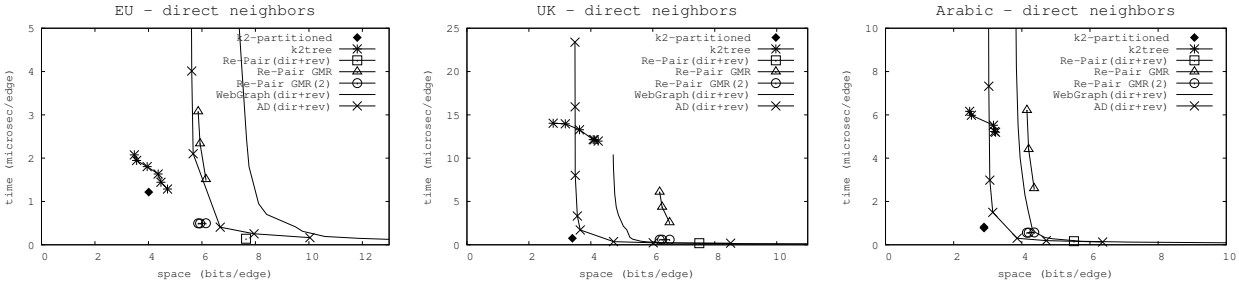

Figure 1: Space/time trade-off to retrieve direct neighbors for several Web graphs.

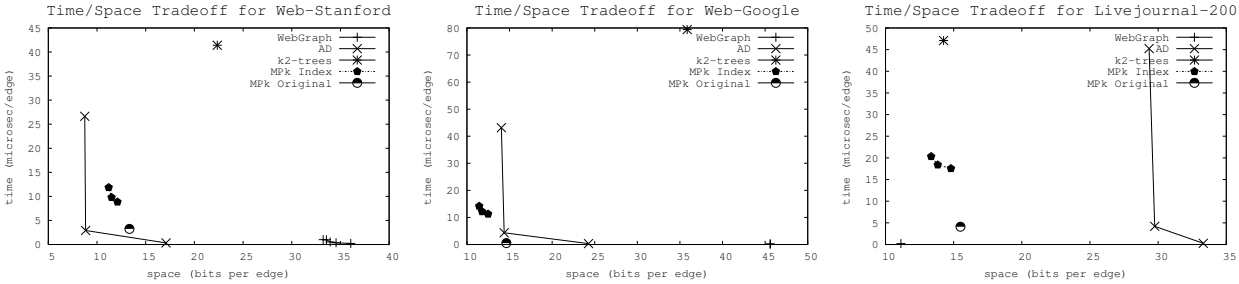

Figure 2: Space/time trade-off to retrieve direct neighbors for several social networks.

7. REFERENCES

[1] A. Apostolico and G. Drovandi. Graph compression by BFS. *Algorithms*, 2(3):1031–1044, 2009.

[2] J. Barbay, T. Gagie, G. Navarro, and Y. Nekrich. Alphabet partitioning for compressed rank/select and applications. In *Proc. ISAAC*, LNCS 6507, pages 315–326. Springer, 2010. Part II.

[3] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[4] P. Boldi, M. Santini, and S. Vigna. Permuting Web graphs. In *Proc. WAW*, pages 116–126, 2009.

[5] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proc. WWW*, pages 595–601, 2004.

[6] N. Brisaboa, S. Ladra, and G. Navarro. Directly addressable variable-length codes. In *Proc. SPIRE*, LNCS 5721, pages 122–130. Springer, 2009.

[7] N. Brisaboa, S. Ladra, and G. Navarro. K2-trees for compact web graph representation. In *Proc. SPIRE*, LNCS 5721, pages 18–30. Springer, 2009.

[8] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *Proc. KDD*, pages 219–228. ACM, 2009.

[9] F. Claude and G. Navarro. Practical rank/select queries over arbitrary sequences. In *Proc. SPIRE*, LNCS 5280, pages 176–187, 2008.

[10] F. Claude and G. Navarro. Extended compact web graph representations. In *Algorithms and Applications (Ukkonen Festschrift)*, LNCS 6060, pages 77–91, 2010.

[11] F. Claude and G. Navarro. Fast and compact web graph representations. *ACM Transactions on the Web*, 4(4):article 16, 2010.

[12] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms*, 3(2):article 20, 2007.

[13] A. Golynski, I. Munro, and S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proc. SODA*, pages 368–373, 2006.

[14] R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *Proc. SODA*, pages 841–850, 2003.

[15] G. Jacobson. Space-efficient static trees and graphs. In *Proc. SFCS*, pages 549–554, 1989.

[16] J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The Web as a graph: Measurements, models, and methods. In *Proc. COCOON*, LNCS 1627, pages 1–17, 1999.

[17] S. Ladra. *Algorithms and Compressed Data Structures for Information Retrieval*. PhD thesis, Department of Computer Science, University of A Coruña, 2011.

[18] J. Larsson and A. Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000.

[19] V. Mäkinen and G. Navarro. Rank and select revisited and extended. *Theoretical Computer Science*, 387:332–347, 2007.

[20] H. Maserrat and J. Pei. Neighbor query friendly compression of social networks. In *Proc. KDD*, pages 533–542. ACM, 2010.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[22] H. Saito, M. Toyoda, M. Kitsuregawa, and K. Aihara. A large-scale study of link spam detection by graph algorithms. In *Proc. AIRWeb*. ACM Press, 2007.