

PhD Thesis Proposal

Managing Massive Graphs

Cecilia Hernández

Advisors: Gonzalo Navarro gnavarro@dcc.uchile.cl

Mauricio Marín mmarin@yahoo-inc.com

December, 2009

Universidad de Chile, Chile

chernand@dcc.uchile.cl

Abstract. Many real graphs conform today some of the largest data sets. Some of the best representatives of these graphs are the web graph, the interconnection network graph, the telephone call-graph, social networks, and query log graphs. Managing and finding relevant information on large graphs are challenging problems in current research. The need to deal with massive graphs has increased the interest in different research areas, such as compact data structures, data streaming, graph mining, secondary storage, and distributed algorithms.

In this thesis, we propose to study the management of massive graphs in memory, secondary storage and over distributed systems for graph applications such as the web graph, social networks and query log graphs. We specifically aim to define competitive compressed graph representations with application-driven search capabilities, providing storage and search time tradeoffs. We also propose the study of strong and weak data stream models and algorithms for constructing, storing and mining compressed large graphs in order to provide space/time algorithmic guarantees.

1 Introduction

There are many massive real graph-based applications that are of special interest, including the web, social networks and query log graphs. In particular, one of the largest known graphs is the web graph, which Google reported (in its Official Blog) to have more than one trillion pages and many more edges in 2008. The web graph is a directed graph, where nodes represent web pages and the edges correspond to hyperlinks between pages. These data is crucial for search engines ranking algorithms, such as PageRank [10] and HITS [43], and also for spam detection [4]. On the other hand, social networks are undirected graphs, where nodes represent people and edges are represented by the relationship between people. Social networks, such as Flickr and Facebook, have become very popular and continue to experience user population growth, therefore its understanding and analysis is important to improve and envision new collaboration systems [16, 47]. These networks also offer an opportunity to study their dynamics and data management at a scale. Metrics, such as node centrality, outdegree and

indegree centrality, proximity prestige, and rank prestige are used for analyzing social networks [42]. Also, query log graphs have been used to infer properties of the queries, obtain semantic relations found in the graph [33], and to improve spam detection [15]. Usually, a query log graph is modeled as a directed bipartite graph with query and URL sets. There are many other interesting applications modeled by graphs including scientific citation, Internet interconnection network, and gene sequences.

While most of the data can be stored in secondary memory, such as disks, one of the main challenges for managing massive data sets include the high cost of I/O operations for current computer hardware, which it is usually optimized for sequential access. This has been the main research motivation for proposing graph representations in compressed form, trying to allow algorithms to have more data available in main memory and minimizing random access to disks. In the context of the web graph, there are many proposals to compress the graph so that the direct neighbors of nodes are efficiently retrieved [9, 13, 55]. However, not much has been done to support both direct and reverse navigation on the graph. Ranking algorithms such as PageRank and HITS need such operations. PageRank gives high ranking scores to important pages based on the importance of the pages that point to them, and HITS assigns a hub score and an authority score to each page. In HITS, a page is a good hub if it points to good authorities and a page is a good authority if it is pointed by good hubs. As far as we know, there are only a couple proposals that supports both direct and reverse access to nodes [11, 20], but neither of them exploit the data stream model. In general, a common performance metric for measuring compression of the web graph is the *bpe* (bits per edge).

In addition to the I/O high costs, processing large data sets in polynomial upper bound order might need a computational time that is not possible in practice. Expensive I/O operations and algorithm time requirements have motivated an increased interest in studying efficient ways to store/retrieve data from secondary storage devices. A well-studied model to deal with data sets that do not fit in memory is the external memory. External memory algorithms [57] can access disks only in units of blocks (each with at most B items) and performance metrics are based on the number of disk accesses for read/write operations. However, external memory algorithms allow random access to disks. More recently there has been an increased attention in the data stream model. Data stream models impose sequential access to data on disk, which provides a higher throughput than performing random access. This model assumes a limited working memory, usually of polylogarithmic order with respect to the input size, and allows one or a few algorithm passes over the data stream. In general, graph problems are difficult for the strongest data stream model [49], but recently more relaxed data stream models [5, 25, 29] have provided good results for solving some graph problems.

Although data stream models, in theory, facilitates the study of algorithms in massive data sets, these models by themselves are insufficient [4]. For example, web data is so large that no single processor can make even a single pass over

the data in a reasonable amount of time. Therefore, computations need to be distributed over multiple machines. In addition, in the web scenario, data is collected, gathered and stored on many machines to be able to keep track of the web activity, such as the web crawlers and query logs. This distribution carries several challenges in theory and practice. In theory, the data stream model is sequential and there is a need of designing distributed versions for the algorithms. Whereas in the practice context, distributed algorithms have to deal with synchronization, data distribution, load balancing, scalability, etc.

In this thesis, we propose to study the management of large graphs with the goal of providing graph representations to support low space requirements as well as fast navigation algorithms. We aim to manage these graphs, not only in main memory, but also on secondary storage and in a distributed system. We plan to consider large graphs like the web, social networks and query log graphs.

2 Related Work

This section discusses how the research community has addressed the main challenges described in the previous section. First, we discuss some graph compression approaches (mostly for web graphs) aiming to process/retrieve more data in main memory and consequently reduce the access to secondary storage. These approaches include graph properties such as power law distributions, locality and similarity; community structures and grammar-based techniques. Then, we describe compact data structures, which have provided good compression with search capabilities results, and we discuss, in more detail, the grammar-based technique. Afterwards, we review proposed secondary storage models such as external memory algorithms and data streaming for dealing with massive data sets and graphs. Finally, we present how these challenges have been addressed in the context of distributed systems.

2.1 Graph Compressed Representations And Navigation

Graph algorithms are the basis for discovering information in large graphs. In the context of the Web, Donato et al. [27] show that several web mining techniques used to discover the structure and evolution of the web graph, such as weakly and strongly connected components, depth-first search and breath-first search, are based on classical graph algorithms. In a later work Donato et al. [26] present an experimental study of the statistical and topological properties of the web graph. Other proposals use graph algorithms to detect spam farms [34, 54]. Saito et al. [54] present a method for spam detection based on classical graph algorithms such as identification of weak and strong components, maximal clique enumeration and minimum cuts. On the other hand, Gibson et al. [34] propose large dense subgraph extraction as a primitive for spam detection. Their algorithm used efficient heuristics based on the idea of *shingles* [12] (most used to estimate similarity among web pages) together with frequency itemset mining

approximation. There are other techniques, based on graphs algorithms, aiming to extract small subgraphs or small communities for mining purposes [32,44,52].

One of the most studied graphs is the web graph and compressing it has been an active research area [9,13,55]. Some graph properties that have been used for achieving compression are skewed or power-law distribution, locality and similarity. Skewed distribution, in simple terms, says that the probability that a node x has j links is $1/j^\alpha$. For inlinks (incoming links of node x) α is approximately 2.1 and for outlinks (outgoing links of node x) α is about 2.7 [13]. Locality says that, usually, most of the links of a page on a web site point to other pages on the same site. Lastly, similarity says that many pages that are from the same web site tend to have many links in common, or in other words that web pages that are close in lexicographic order tend to share many outlinks.

Locality motivates to order the URLs lexicographically, so that the outlinks for a page are close to previous pages in the outlink list. One of the first proposals that use locality together with gap encoding was proposed by Randall et al. [53] to achieve compression of the web graph. Similarity motivates compression by using the fact that if nodes in the graph are ordered by their corresponding URL, then consecutive ordered adjacency lists have great overlap in outlinks (reference coding).

One of the most competitive approaches [9] (known as the Webgraph framework) in terms of compression and access times exploits power-law distributions, locality and similarity allowing to encode adjacency lists by referencing to a previous adjacency list plus a list of differences. They developed ς codes, which are well suited for compressing power-law distributed data with small exponents. They provide good compression rates for the web graph, its transposed and good access times. However, this approach depends on node ordering and a couple of parameters, such as the window size and the maximum reference count. The window size corresponds to the number of previous rows in the adjacency matrix considered when compressing a row x . Larger window size provides better and slower compression. The maximum reference count is the maximum length of a reference chain, where a reference chain corresponds to the chain formed in the compression process. When compressing a row x of the adjacency matrix, the compression is done with respect to a previous row y , and row y could have been compressed differentially with respect to some other row z , generating a reference chain. This reference chain is important for keeping decompression process fast. If reference chain is too long, decompression is slow.

Such work relies on some URL ordering, which require a sorting pass over the data items. More recently Boldi, Santini and Vigna have explore and evaluated other ordering methods [8], like gray ordering. A simple n -bit gray code is an arrangement of 2^n such that any two successive bit vectors have only one bit different. They showed that using gray ordering provides better results for the transpose of the web graph.

Another approach considers discovering communities on the web graph which are used to represent the web graph in a compressed form. Buehrer et al. [14] identifies directed bipartite cliques as communities by discovering dense sub-

graphs in large graphs [34] together with an approximation for frequent pattern mining. For each bipartite clique, their edges are replaced by a virtual node and its respective edges. A virtual node is a connecting node between the two sets in the bipartite clique. Even though, the work has opportunities for direct and reverse navigation, Buehrer et al. [14] do not provide an implementation and evaluation for supporting such queries. This approach reduces considerably the number of edges in the web graph and, in addition, can potentially answer some interesting questions about the nodes in the graph, such as, is a page x part of a community? or in how many communities does a page x participate in?

In general, ordering-based approaches are sensitive and do not include community discovery into the compressed structure to support queries related to this information. Moreover, Boldi and Vigna [9] relies in a window size parameter that makes difficult to such discovery.

Besides using ordering techniques and communities for compression, other interesting methods have been explored. Claude and Navarro [19,21] have proposed compressing the web graph by using Re-Pair [45], a grammar-based compression technique, providing faster navigation times than the WebGraph framework, for the same space availability.

On the other hand, reverse navigation (retrieving the nodes that point to a node) is usually performed by representing the transpose of the graph in addition to the graph itself. Boldi et al. [8] showed that gray ordering outperforms other ordering techniques for the transpose of the web graph. However, they do not provide both direct and reverse navigation on the same structure. A proposal that supports both direct and reverse navigation [11] represents the adjacency matrix taking advantage of its sparseness providing similar times for direct and reverse queries. More recently, Claude and Navarro [20] present a compact web graph representation using the grammar-based compression technique Re-Pair, with compact data structures, such as bitmaps with rank/select fast operations, wavelet trees [39], and compact tree representations. As a result, they provide the smallest space for a compact web graph representation supporting direct and reverse graph queries.

Some recent works on compressing social networks [6, 17] have showed compression opportunities, although in less degree than in the web graph. Chierichetti et al. [17] approach is based on the webgraph framework [9] exploiting in addition link reciprocity. Whereas, Barbay et al. [6] exploits link reciprocity in the adjacency matrix and uses compact data structures, such as binary relations [7], wavelet trees and basic tools based on bitmaps with rank/select capabilities [50] for compression and query resolution.

2.2 Compact Data Structures

In the last decade, there has been an active research area for defining and implementing compact data structures to improve the space and time requirements. A compact data structure provides the same abstraction as its classical counterpart, using little space and supporting interesting queries without having to expand the whole data structure. In general, compact data structures are attractive for

managing large data sets, because they require less space and therefore more data can be maintained in main memory, procrastinating the need to access the disk. These structures have been successfully used for defining compressed self-indexes for large text collections, compressed graph representations and compressed spatial representations with query capabilities. Ferragina et al. [31] provides a good survey on the theory and practice experiences.

Many compact data structures use a basic tool based on a bitmap supporting rank/select/access query primitives [50]. A $rank_B(b, i)$ operation on the bitmap B counts the number of times bit b appears in the prefix $B[1, i]$. A $select_B(b, i)$ operation returns the position of the i -th occurrence of bit b in B . Lastly, an $access_B(i)$ operation retrieves the value for $B[i]$. A solution that requires $n + o(n)$ bits and provides constant time for rank/select/access queries was proposed by Clark [18]. In later work, Rao et al. (RRR) improved the required space to $nH_0(B) + o(n)$ bits (where H_0 represents the zero-order empirical entropy).

Compact data structures for an arbitrary sequence S with $b \in \Sigma$ are an extension of the basic tool described above. In this context, the wavelet tree [39] supports rank/select/access queries in $O(\log \sigma)$ time and require $n \log \sigma + o(n) \log \sigma$ bits when using Clark bitmap tool, or $nH_0(S) + o(n) \log \sigma$ bits when using RRR for bitmaps. Another structure, proposed by Golynski et al. [36], requires $n \log \sigma + o(n \log \sigma)$ bits and support *rank/access* queries in $O(\log \log \sigma)$ time and $O(1)$ for *select*, or alternatively, $O(1)$ time for *access*, $O(\log \log \sigma)$ for *select* and $O(\log \log \sigma \log \log \sigma)$ for *rank*.

2.3 Re-Pair Compression for Web Graphs

Re-Pair [45] is a grammar-based compression algorithm consisting of repeatedly finding the most frequent pair of symbols in a sequence of integers and replacing it with a new symbol, until no more replacements are convenient. Considering as input an integer sequence S , the algorithm iterates until there is no more available memory for the dictionary R or the replacements do not improve compression. The algorithm iterates over the following three steps. (1) It identifies the most frequent pair ab in S . (2) It adds the rule $r \rightarrow ab$ to a dictionary R , where r is a new symbol that is not in S . (3) It replaces any occurrence of ab for r in S . The output of the compression is the compressed sequence C and the dictionary R . The sequence C is formed both with terminal (original symbols in S) and nonterminal symbols. In order to obtain the original symbols, nonterminal must be decompressed using the information stored in the dictionary R .

Re-Pair has been used for compressing the web graph providing competitive results in terms of compression, i.e. bpe (bits per edge), querying direct neighbors [19, 21] and both direct and reverse neighbors [20]. Claude and Navarro [21] proposed an approximation version for Re-Pair that require little space plus the sequence to avoid the space requirement for keeping track of all pairs needed to be replaced. The web graph representation concatenates all the pages adjacency lists with the goal of allowing direct navigation in optimal time. In this case, the compressed representation of the web graph has direct access for each node to the compressed form of its direct neighbors.

The approximation approach is flexible in terms of memory requirements in the sense that if the whole sequence does not fit in main memory, it is possible to retrieve the data from disk sequentially. Assuming there is an available memory of size M and the sequence T of size $|T|$, with $M > |T|$, the approach consists of having a closed hash table H , where $|H| = M - |T|$. In order to find the k most frequent pairs in the sequence T , the algorithm reads T sequentially inserting unique pairs ab of symbols into H along with a counter that represents the number of occurrences the pair has in T . Pairs ab are the keys in H and are represented by its position in T . If at some point during the T traversal table H goes beyond a load factor, the traversing algorithm does not insert more items in H , and only increments counters of existent pairs as needed. Once the traversing algorithm is done the k most frequent pairs are obtained from H and maintained in a heap. In the next step of the Re-Pair algorithm the k most frequent items are replaced in T (some care is needed to avoid problems when some replacements may invalidate others). After replacing the k most frequent pairs in T , the size of the sequence decreases, then, it is possible to store more pairs in H . The overall time cost for executing the approximation Re-Pair algorithm, just described, is linear, achieving good compression rates. More specifically, the running average time is $O(\lceil r/k \rceil (n + n \log k))$, for k most frequent items and $h = |H| = O(n)$ and r replacements. Therefore, this approximation allows a space/time tradeoff by choosing k . When there is not enough memory available, T is stored on disk and H in memory. The worst-case I/O cost of the algorithm is $O(\lceil r/k \rceil (n/B + k)) = O(\lceil r/k \rceil (n/B + r + k))$, where B is the block size.

2.4 Data Streaming

In order to address massive data sets the research community has recently placed attention in the data stream model [48, 49]. Data streaming has been used to solve different problems like statistic computations (histograms [35, 40], quantiles [37]) and data mining like the *heavy hitters* problem (frequent item problem) [23]. More recently researchers have proposed more relaxed data stream models [5, 25, 29], especially attractive for dealing with classic graph problems.

In the traditional data stream model, the input is a stream of data items, and the streaming algorithm must process data items sequentially using limited memory space, usually of polylogarithmic order with respect to the input size. The algorithm must process the data stream, ideally in one pass, but for complex problems few passes are allowed.

In the context of mining data streams, the frequent items problem (also known as the *heavy hitters problem*) has been studied. The problem consists of finding the items that occur more frequently in a data stream. There are many applications that can use the solution of this problem, items might represent packets in a computer network, queries on a search engine, symbols in an integer or string sequence, etc.. Carmode et al. [23] describe the most important algorithms for this problem. They provide implementations and performance evaluation comparison among the main algorithms and showed that the best methods can

be implemented by using little working memory (tens of kilobytes) providing high rates of items (millions) per second on commodity computer systems.

The semi-streaming model, introduced by Feigenbaum et al. for graph problems [28], allows a space usage of $O(n \text{ polylog } n)$, for n nodes of a graph, defining the space restriction based on the number of nodes instead of the number of edges. Such model has been called a *sweet-spot* for dense graphs [49]. The *streaming with sorting* model, proposed by Aggarval et al. [5] allows the model to have temporary streams with an additional sorting algorithm pass. A later work, Demetrescu et al. [25] showed that having temporary streams (*write-stream*) during each pass is sufficient to provide effective space/passes tradeoffs for some classical graph problems.

Some of the interesting graph problems that have been considered in the semi-stream data model are the unweighted and weighted matching problems. In this case, Feigenbaum et al. [28] provide algorithms for finding unweighted bipartite graphs with a $2/3-\varepsilon$ approximation matching algorithm in $O((\log(1/\varepsilon))/\varepsilon)$ passes, with $0 < \varepsilon < 1/3$. The algorithm processes each edge in $O(1)$ time in each pass, although finding the bipartition takes an amortized time proportional to the number of edges and nodes of the graph. The space required by the algorithm is $O(n \log n)$. For the weighted matching problem, where each edge has a weight, the same work provides an algorithm that is at least $1/6$ of the optimal size and uses only one pass over the stream, requiring $O(n \log n)$ space. Other graph problems studied under the stream-sort [5] and write-stream [25] data stream models include the connectivity problem in undirected connectivity.

Another interesting graph problem is finding approximate neighborhood functions. A basic *neighborhood function* is the number of pairs of nodes within a specified distance h . This function is used to measure the connectivity of the graph, subgraph similarity and node connectivity (which can be of importance for relevance or spam detection). Palmer et al. [52] provide a scalable and fast data mining tool that finds approximate neighborhood functions. Their tool include algorithms that process edge stream sequentially, avoiding graph traversals. They do not provide a data stream algorithm explicitly, but instead an external memory algorithm. The algorithms are attractive because they use low storage requirements, access edges sequentially, provide oportunities for exploiting paralellism, and are scalable.

2.5 Distributed Systems

Given that there is no single machine capable of keeping massive data sizes in memory, distributed frameworks and models get an important rol for managing large data sets. Distributed technologies such as MapReduce [24] and Hadoop-MapReduce [3] are currently used to process several terabytes of data at a time. Hadoop-MapReduce is basically an open source version of the MapReduce paradigm, and it is being used by industry (being Yahoo one of the main contributors) and academia [1].

The MapReduce model allows the execution of distributed algorithms defined mostly as three functions. (1) a local function that takes a single data item and

output a message. (2) an aggregation function to combine pairs of messages and (3) sometimes a third function for post processing. The local function is applied to input data items independently and in parallel, and the aggregation function can be applied to pairs of messages in any order. Local functions (Mappers) are executed on local data items and basically map data items to $\langle key, value \rangle$ pairs combining them to a set of intermediate $\langle key, value \rangle$ pairs. Aggregation functions (Reducers) reduce a set of intermediate values which share a key to a smaller set of values.

An important feature of the MapReduce paradigm that makes it different from previous models of parallel computation is that it interleaves sequential and parallel processing. Parallel processing occurs when multiple mappers execute over different local data items and reducers execute over different intermediate data generated by mappers. The sequential part happens because reducers must wait for mappers before executing. To simplify fault tolerance, in the MapReduce model, Mappers materialize their output to disk before the Reducers can consume it, therefore the MapReduce model has been mostly used for computing batch jobs. A recent work proposes a modified MapReduce architecture that supports online aggregation and continuous queries keeping fault tolerance [22].

Feldman et al. [30] study the relationship between distributed algorithms on these platforms and streaming computation. The question they want to answer is whether there are limitations on the possible computations when using multiple machines compared with streaming on a single machine. They introduced an algorithm model for massive, unordered, distributed (MUD) computation implemented by these systems. They show that order-invariant streaming algorithms can also be computed by MUD algorithms with similar space/time costs. In the MUD model, the Reduce function considers as an input a data stream that uses a working memory of polylogarithmic size.

More recently, Karloff et al. [41] propose a different model of computation for MapReduce, limiting the number of machines and memory per machine to be sublinear with respect to the input size. However, this model allows random access to the reducer input data items and does not restrict its working memory. This model requires that both map and reduce functions execute in time polynomial on the local input size, limits the working memory for both functions to $O(n^{1-\epsilon})$ and requires to have a number of rounds of $R = O(\log^i n)$. They claim to achieve a more powerful model and provide algorithms for computing the Minimum Spanning Tree (MST) and undirected connectivity in large graphs.

Even though, the MapReduce paradigm has several virtues such as simplicity, fault tolerance and low managing costs, its simplicity makes difficult to express some computations needed by complex applications. Different application requirements have motivated researchers to develop and propose other distributed models. For instance Sawzal [38] and Pig [51] are programming models on top of MapReduce. More recently, Google announced Pregel [46], a computational model for addressing graph algorithms in a distributed environment, which is inspired by Bulk Synchronous Parallel model (BSP) [56]. However, not much details about Pregel have been published yet. Likewise, Hama [2] is a distributed

framework for processing matrix and graph data on Hadoop, but there is no publication available yet.

On the other hand, the BSP model is based on the idea of supersteps. Each superstep consists of three stages: parallel computation on local data, communication through point-to-point messages among processors, and a synchronization stage where all processors wait and synchronize at the end of the superstep. One of the advantages of BSP over the MapReduce model is that BSP allows to preserve data locality in consecutive supersteps where each processor may communicate to others as necessary while other processors preserve data locality. In contrast with BSP, the MapReduce model does not preserve data locality in consecutive operations, which might limit performance and scalability of complex applications.

3 Thesis Proposal

The main goal of this thesis is to improve the space requirements and information access time of large graphs considering either memory, secondary storage and distributed systems.

In the following subsections we address current problems described in the introduction, we provide the specific objectives, set of activities and expected results of the research.

3.1 Compact Graph Representation and Retrieval with Communities

Some large graph applications of great interest today are the web graph, social networks and query log graphs. In particular, as discussed in previous section, compressing the web graph on memory has been an active research topic. Many proposals use the web graph properties such as power law distributions, locality and similarity. Another approach has been the use of grammar-based technique such as Re-Pair and recently another approach considers community structures of the web graph. Most of the compressed graph representation support only direct neighbor retrieval, requiring an additional representation for the reverse neighbor access. In this context, not much has been done for supporting, implementing and evaluating both direct and reverse neighbors and queries that answer specific community questions using the same compressed graph representation. On the other hand, compact data structures have been used for grammar-based but not with community-based compression. Therefore, we believe that using compact data structures can enhance both compression and provide algorithm guarantees for retrieving reverse/direct neighbors as well as answering specific community related queries.

Therefore, our specific objective is to study the compression rate and access time for community related queries including direct/reverse neighbors for web and social network large graphs, using compression based on communities and compact data structures to reduce the amount of memory and provide query

access time guarantees. The set of activities to consider to accomplish this specific objective are:

- To analyze and evaluate different community discovery tools such as [26] and [14] for finding bipartite cliques for web graphs and social networks.
- To implement and evaluate graph representations using compact data structures such as wavelet trees [39] and Golynski [36].
- To design, implement and evaluate alternative compact graph representations that exploits runs and other possible regularities on the input sequence representing communities.
- To perform a theoretical analysis for compression rate as well as computational time tradeoffs for large graphs.
- To do a empirical performance evaluation in terms of space/time requirements among the different representations using compact data structures.
- To analyze the solutions based on other metrics such as scalability and/or applicability to models that include secondary storage or distributed systems.

Expected Results At the end of this stage, we expect to have a compact graph representation that enhance the compression achieved by using community structures, and at the same time provides good times for community based query resolution including direct/reverse neighbors.

3.2 Grammar-based Compressed Graphs with Streaming

Since managing large data sets is a real challenge today, and we are not aware of any proposal that compress graphs using data streaming algorithms, we think there are interesting opportunities to improve the management of large graphs considering secondary storage. An interesting compressed web graph representation that supports both direct/reverse neighbor access is based on Re-Pair [20]. Such algorithm need to iteratively estimate the most frequent items of the sequence. On the other hand, there are efficient data streaming algorithms that require little working memory (some tens of kB) and provide a high rate processing (millions per second) for solving the problem of heavy hitters (frequent item set) [23].

Therefore, our specific objective is to study the use of the heavy hitter data streaming algorithms [23] to compute and construct a compressed graph representation using Re-Pair for direct/reverse navigation for the web graph.

In order to achieve this objective we consider the following activities.

- To design and implement streaming heavy hitter algorithms for constructing a compressed graph representation using Re-Pair.
- To perform a theoretical analysis for this solution in terms of space/time requirements.
- To evaluate the impact of using the heavy hitter algorithms based on the number of frequent items, space requirements, computational time and precision.

- To perform a performance analysis and tradeoffs in terms of space/time requirements as well as parameter sensitivity (for example, number of frequent items) for this solution compared with previous proposal without streaming [20].

Expected Results We expect to propose a compact graph representation for the web graph considering streaming algorithms to improve the access time and memory requirement for accessing data from the secondary storage.

3.3 Graph Compressed Representation with Streaming on a Distributed Environment

We want to address the need of designing distributed algorithms for massive graphs, which sizes at some point are not possible to process in a single machine. We will study the feasibility of using distributed computing models with streaming. We plan to use relaxed parallel models such as BSP and more restrictive paradigms like MapReduce. BSP offers features that are attractive for preserving data locality and for applications that require recursion or iterative operations; and MapReduce offers simplicity and fault tolerance. The research community, fairly recently, has been interested in computational models based on MapReduce and Streaming [30, 41], and has been working on distributed frameworks to address complex applications including graph algorithms [2, 46].

In this case, our main specific objectives are:

1. To determine the feasibility of combining data streaming algorithms with parallel models such as BSP and MapReduce. In particular, we want to determine if it is possible to use the heavy hitter data streaming algorithms [23] with approximate Re-Pair [19, 21] for compressing the web graph and social networks, such as Flickr and scientific citation, in a distributed setting.
2. To analyze whether those approaches can be used using a previous proposed distributed data streaming model [30] or more relaxed MapReduce computational model [41].
3. To determine the impact (in terms of performance/costs, flexibility and scalability) of using a distributed environment for compressing a large graph using data streaming with the BSP and MapReduce models.
4. To determine the feasibility of a search algorithm to retrieve direct/reverse neighbors on the distributed compressed web graph in order to support basic operations required by metrics used in social network applications [42], and web spam detection using query log graphs [15].

In order to achieve the specific objectives we envision the following activities:

- To define the operation setting for the distributed environment and input graphs considering partitioning and number of machines with their corresponding capacities.

- To define and analyze the needed interfaces to combine the heavy hitter data streaming algorithms with BSP model and MapReduce paradigm for implementing graph compression using Re-Pair.
- To design a distributed query algorithm in order to search direct/reverse neighbors on distributed compressed graph representation.
- To perform a theoretical analysis in terms of space/time requirements and communication costs.
- To implement the interfaces defined on previously.
- To evaluate the implementation prototype in terms of metrics such as compression rate, construction time, number of rounds needed and bandwidth usage for different configuration parameters such as the memory availability, and number of frequent items to use.
- To evaluate the solutions in terms of simplicity, scalability and flexibility.
- To implement and evaluate the distributed query algorithm in terms of communication costs and scalability.

Expected Results We expect to have a distributed algorithms using heavy hitter algorithms with BSP and MapReduce models for compressing large graphs, with a performance analysis (theretical and practical) based on memory used, disk access, communication costs and comparison with respect to the sequential case defined in previous objective.

4 Derivables

The main contributions of this thesis will be the followings:

- Theoretical and practical analysis in the area of massive graph compression with search capabilities on memory, secondary storage and distributed systems.
- Public algorithm implementations to leave them available either for industry and academic communities.
- We plan to publish our main results in high-level international conferences and/or in ISI journals.

5 Conclusion

Managing massive graphs is a challenging problem nowadays. Current graph sizes are so large that managing them do not only require to consider main memory, but also secondary storage and distributed systems. Finding relevant information and storing these data efficiently has motivated the study of compression using compact data structures, graph properties and data mining approaches. It has also motivated the study of efficient secondary storage models and distributed models and algorithms.

The main goal of our research is to contribute with compressed graph representations that support search capabilities for massive graphs, considering memory, secondary storage and distributed systems. We plan to consider applications based on the web, social network and query log graphs. We specifically aim to provide on memory, compressed graph representation with query capabilities using community structures found on graphs and compact data structures. We also plan to manage on memory and secondary storage compressed graph representation with search, using grammar-based compression and data streaming algorithms. Finally, we want to study the feasibility and performance of designing/implementing compressed graph representations with search capabilities on a distributed setting using data streaming algorithms and compression techniques.

References

1. Yahoo!! partners with four top universities to advance cloud computing systems and application research.yahoo! press release,2009. In <http://research.yahoo.com/news/2743>, 2009.
2. The apache hama project. In <http://incubator.apache.org/hama/>, 2010.
3. D. Cutting A. Bialecki, M. Cafarella and O. O'Malley. Hadoop: a framework for running applications on large clusters built of commodity hardware. In *Wiki at* <http://lucene.apache.org/hadoop/>, 2005.
4. Gagan Aggarwal, Nir Ailon, Florin Constantin, Eyal Even-Dar, Jon Feldman, Gereon Frahling, Monika Rauch Henzinger, S. Muthukrishnan, Noam Nisan, Martin Pál, Mark Sandler, and Anastasios Sidiropoulos. Theory research at google. *SIGACT News*, 39(2):10–28, 2008.
5. Gagan Aggarwal, Mayur Datar, Sridhar Rajagopalan, and Matthias Ruhl. On the streaming model augmented with a sorting primitive. In *FOCS*, pages 540–549. IEEE Computer Society, 2004.
6. J. Barbay, F. Claude, and G. Navarro. Compact rich-functional binary relation representations. In *Proc. 9th Latin American Symposium on Theoretical Informatics (LATIN)*, LNCS, 2010. To appear.
7. Jérémy Barbay, Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Adaptive searching in succinctly encoded binary relations and tree-structured documents. In Moshe Lewenstein and Gabriel Valiente, editors, *CPM*, volume 4009 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2006.
8. Paolo Boldi, Massimo Santini, and Sebastiano Vigna. Permuting web graphs. In Konstantin Avrachenkov, Debora Donato, and Nelly Litvak, editors, *WAW*, volume 5427 of *Lecture Notes in Computer Science*, pages 116–126. Springer, 2009.
9. Paolo Boldi and Sebastiano Vigna. The webgraph framework i: compression techniques. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *WWW*, pages 595–602. ACM, 2004.
10. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.
11. N. Brisaboa, S. Ladra, and G. Navarro. K2-trees for compact web graph representation. In *Proc. 16th International Symposium on String Processing and Information Retrieval (SPIRE)*, LNCS 5721, pages 18–30. Springer, 2009.
12. Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.

13. Andrei Z. Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet L. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000.
14. Gregory Buehrer and Kumar Chellapilla. A scalable pattern mining approach to web graph compression with communities. In Marc Najork, Andrei Z. Broder, and Soumen Chakrabart, editors, *WSDM*, pages 95–106. ACM, 2008.
15. Carlos Castillo, Claudio Corsi, Debora Donato, Paolo Ferragina, and Aristides Giannis. Query-log mining for detecting spam. In Carlos Castillo, Kumar Chellapilla, and Dennis Fetterly, editors, *AIRWeb*, ACM International Conference Proceeding Series, pages 17–20, 2008.
16. Meeyoung Cha, Alan Mislove, and P. Krishna Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *WWW*, pages 721–730. ACM, 2009.
17. Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki, editors, *KDD*, pages 219–228. ACM, 2009.
18. D. Clark. Compact pat trees. In *Ph.D. Thesis, University of Waterloo*, 1996.
19. F. Claude and G. Navarro. Fast and compact web graph representation. In *Tech. Report TR/DCC-2008-3. Dept. of Comp. Sci., University of Chile*, 2008.
20. F. Claude and G. Navarro. Extended compact web graph representations. In *Book Chapter*, LNCS, 2010. To appear.
21. Francisco Claude and Gonzalo Navarro. A fast and compact web graph representation. In Nivio Ziviani and Ricardo A. Baeza-Yates, editors, *SPIRE*, volume 4726 of *Lecture Notes in Computer Science*, pages 118–129. Springer, 2007.
22. Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmelegy, and Russell Sears. Mapreduce online. Technical Report UCB/EECS-2009-136, EECS Department, University of California, Berkeley, Oct 2009.
23. Graham Cormode and Marios Hadjieleftheriou. Finding the frequent items in streams of data. *Commun. ACM*, 52(10):97–105, 2009.
24. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.
25. Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. In *SODA*, pages 714–723, 2006.
26. Debora Donato, Luigi Laura, Stefano Leonardi, Ulrich Meyer, Stefano Millozzi, and Jop F. Sibeyn. Algorithms and experiments for the webgraph. *J. Graph Algorithms Appl.*, 10(2):219–236, 2006.
27. Debora Donato, Stefano Leonardi, Stefano Millozzi, and Panayiotis Tsaparas. Mining the inner structure of the web graph. In AnHai Doan, Frank Neven, Robert McCann, and Geert Jan Bex, editors, *WebDB*, pages 145–150, 2005.
28. Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005.
29. Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, 2008.
30. Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Clifford Stein, and Zoya Svitkina. On distributing symmetric streaming computations. In Shang-Hua Teng, editor, *SODA*, pages 710–719. SIAM, 2008.

31. P. Ferragina, R. González, G. Navarro, and R. Venturini. Compressed text indexes: From theory to practice. *ACM Journal of Experimental Algorithmics (JEA)*, 13:article 12, 2009. 30 pages.
32. Gary William Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. In *KDD*, pages 150–160, 2000.
33. Alexandre P. Francisco, Ricardo A. Baeza-Yates, and Arlindo L. Oliveira. Clique analysis of query log graphs. In Amihoud Amir, Andrew Turpin, and Alistair Moffat, editors, *SPIRE*, volume 5280 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2008.
34. David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 721–732. ACM, 2005.
35. Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *STOC*, pages 389–398, 2002.
36. Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *SODA*, pages 368–373, 2006.
37. Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD Conference*, pages 58–66, 2001.
38. Robert Griesemer. Parallelism by design: data analysis with sawzall. In Mary Lou Soffa and Evelyn Duesterwald, editors, *CGO*, page 3. ACM, 2008.
39. Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850, 2003.
40. Sudipto Guha, Nick Koudas, and Kyuseok Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. Database Syst.*, 31(1):396–438, 2006.
41. Siddharth Suri Howard Karloff and Sergei Vassilvitskii. A model of computation for mapreduce. In *SODA*, LNCS, 2010. To appear.
42. Musial Katarzyna, Kazienko Przemyslaw, and Bródka Piotr. User position measures in social networks. In *SNA-KDD '09: Proceedings of the 3rd Workshop on Social Network Mining and Analysis*, pages 1–9, New York, NY, USA, 2009. ACM.
43. Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.
44. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the web for emerging cyber-communities. *Computer Networks*, 31(11-16):1481–1493, 1999.
45. N. Jesper Larsson and Alistair Moffat. Offline dictionary-based compression. In *Data Compression Conference*, pages 296–305, 1999.
46. Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In Srikanta Tirthapura and Lorenzo Alvisi, editors, *PODC*, page 6. ACM, 2009.
47. Alan Mislove, Massimiliano Marcon, P. Krishna Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In Constantine Dovrolis and Matthew Roughan, editors, *Internet Measurement Conference*, pages 29–42. ACM, 2007.
48. J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.
49. S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

50. Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1), 2007.
51. Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In Jason Tsong-Li Wang, editor, *SIGMOD Conference*, pages 1099–1110. ACM, 2008.
52. Christopher R. Palmer, Phillip B. Gibbons, and Christos Faloutsos. Anf: a fast and scalable tool for data mining in massive graphs. In *KDD*, pages 81–90. ACM, 2002.
53. Keith H. Randall, Raymie Stata, Janet L. Wiener, and Rajiv Wickremesinghe. The link database: Fast access to graphs of the web. In *DCC*, pages 122–131. IEEE Computer Society, 2002.
54. Hiroo Saito, Masashi Toyoda, Masaru Kitsuregawa, and Kazuyuki Aihara. A large-scale study of link spam detection by graph algorithms (s). In *AIRWeb*, volume 215 of *ACM International Conference Proceeding Series*, 2007.
55. Torsten Suel and Jun Yuan. Compressing the graph structure of the web. In *Data Compression Conference*, pages 213–222, 2001.
56. Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.
57. Jeffrey Scott Vitter. External memory algorithms and data structures. *ACM Comput. Surv.*, 33(2):209–271, 2001.