Research Statement

Özgür Özkan (Oz)

My research interests broadly lie in the design and analysis of algorithms and data structures. My doctoral work focused on employing amortization as a tool to design efficient solutions for fundamental data structuring problems. In the past year, I have taken an interest in information retrieval. My current work focuses on algorithmic problems arising in information retrieval, cache-efficient algorithm design, and dynamic graph algorithms. Below, I will give a summary of my past and current research.

Distributed Index Maintenance

The query processing performance of search engines is very closely tied to user retention and revenue per user. It has been reported by Google that the injection of a 400 millisecond delay in search results caused a 0.6% drop in searches per user after just 4-6 weeks [9]. Even after the delay was removed, these users performed 0.21% less searches in the following 5 weeks. Bing reports a 1.2% loss in revenue per user with a 500 millisecond delay [54]. In this work, we focus on the efficient maintenance of the ubiquitous inverted index data structure used in search engines for answering queries.

A search engine query takes as input a sequence of terms (the query) and returns a set of documents from a collection. An inverted index of a collection of documents (where each document is a sequence of terms) stores for each term appearing in the collection a list of identifiers for documents containing the term. Each document is identified by a unique integer (docID). A query processing algorithm can then use such a list to locate the documents containing a term.

In order to save space, the lists are compressed. Better compression of the lists translates to better query processing speed due to the reduced cache misses during query processing. A common approach for improving the compression ratio is to make the lists more compressible by reordering the docIDs [5, 23].

Reordering methods have received a lot of attention in the past decade and there is a diverse set of reordering methods developed in the literature [3, 5, 6, 15, 16, 23, 28, 43, 44, 47–49, 51, 56, 62]. However, most of these methods are not scalable. Ding et al. [23] use graph sparsification to engineer scalable variants of TSP-based methods, which determine a reordering by computing a TSP-tour of a graph defined on the documents in the collection where the edges represent the similarity between documents. TSP-based approaches do not appear to be the most natural choice for this problem considering TSP is inapproximable and the sparsification process degrades the quality of the solution further.

The effects of reordering are not limited to improved compression as many reordering schemes also reduce the average number of atomic operations performed during query processing. In ongoing work, I am exploring the use of tools we developed for biased skip lists for mergeable dictionaries [40] and approximations of alternate graph problems better suited for block compressed lists to design

new reordering schemes that exploit the relationship between reordering and number of operations performed in order to find improved tradeoffs between space and query processing time.

Web-scale document collections are often dynamic and too large to fit on a single machine. We study the online version of the index maintenance problem in the distributed setting where documents arrive one at a time and have to be routed to one of the available machines. That machine then inserts the document into its inverted index. Each machine may also reorder parts of its inverted index from time to time. In this setting, in addition to space and query time, we care about the time it takes to insert a document into the collection, the time spent for reordering parts of indices if any, as well as load balancing of both documents and queries. Degrading document load balancing often improves compression ratio. Our goal is to design systems with good compression ratio without sacrificing load balancing.

It is common practice to route incoming documents randomly or based on their url (source-based routing) [46]. Recently, Lavee et al. [47] introduced a greedy routing scheme which outperforms both random routing and source-based routing in terms of compression ratio. In order to deal with load balancing issues, the choices of the routing algorithm are restricted when load becomes unbalanced, significantly degrading the compression ratio. These solutions do not utilize reordering of the index and append new documents to the end of the index (docID of the new document is inserted to the end of the appropriate lists).

Instead of altering the routing algorithm, load balancing with respect to space can be achieved via replication by maintaining multiple shards at each machine and distributing the replicas of shards to carefully chosen machines. For load balancing with respect to queries, multiple copies of a shard can be forwarded a query based on the current query load on each machine in order to reduce latencies. It is also interesting to design schemes which minimize the average cost of running a query or maximize the throughput of the system given the average query processing times and processing capacity of each machine. I address a similar optimization problem in [33]. Using replication to handle load balancing allows the routing scheme to be freed of load balancing concerns.

The dynamic nature of the document collection precludes a perfectly reordered index. In ongoing work, we introduce a framework that reorders parts of the index based on query and update frequencies.

I am currently exploring new greedy schemes which utilize more robust score functions and compression algorithms that work well together. The routing step essentially mimics a clustering method improving the performance of the reordering method. Thus, it is beneficial in terms of compression for the routing scheme, the reordering algorithm, and the compression scheme to complement each other. In the setting where reordering is not feasible, I am exploring the feasibility of allowing documents to be inserted in the middle of lists as opposed to the end. This approach will benefit from new developments in maintaining dynamic compressed arrays in the external memory model.

I am also interested in making greedy routing schemes scalable via weighted sampling. It is not clear how to assign such weights optimally. I am exploring using machine learning techniques to determine the score functions dynamically based on properties of the current distribution of the document collection. Speeding up the routing step via LSH-type hashing techniques is also interesting but this approach would benefit from the use of a more appropriate similarity measure than Jaccard similarity or intersection size.

Lastly, the index maintenance problem, as with many systems of similar scope, presents an

interesting set of cache efficiency issues. Using cache-efficient hierarchical structures to improve query processing speed, or to insert documents at locations other than the end of the index requires the development of cache-efficient data structures that work with block compressed data. This is related to my work on cache-oblivious data structures.

Cache-friendly algorithms

There is a wealth of literature on the design of cache-friendly data structures and algorithms. Two of the most well-established models are the I/O model [1] and the cache-oblivious model [31].

My work on persistence in the cache-oblivious model was the recipient of the best paper award at the 22nd Annual European Symposium on Algorithms (ESA) [19]. Persistence in data structures literature refers to the ability to query or update past states of the data structure and has a diverse set of applications including in version control systems, geometric algorithms, and functional programming. Persistence is usually discussed in the context of pointer machines which have no specific consideration of cache-efficiency [24, 29]. In this work, we introduced a general framework to convert any cache-oblivious data structure to one that supports partial persistence (can query past versions of the data structure). Our method can be used to obtain the first persistent versions of numerous cache-oblivious data structures.

In ongoing work, I am working on an extension of these results to full persistence which allows update operations in the past in addition to query operations. Besides being a natural extension, combining cache-efficiency with full persistence will lead to significant performance improvements in functional programming.

This is closely related to the study of the packed memory arrays (PMAs) which store n elements in $\Theta(n)$ space and support update and search operations. There exists PMAs with $\Theta(\log^2 n)$ complexity matching the amortized lower bounds [10, 58–60]. PMAs are fundamental data structures in external memory and warrant further study. Bender et al. [4] proved bounds for special insertion sequences for PMAs that surpass the lower bounds. I am currently working on proving bounds for PMAs parametrized by features of the operation sequences, similar to the types of bounds found in binary search tree literature (e.g. working set). Such bounds could allow us to characterize common access patterns of PMAs exhibited by various applications and circumvent the log squared lower bounds.

Advances in external memory algorithms and data structures often directly translate to performance improvements in real systems. Motivated by the index maintenance problem described above, I am interested in the design of dynamic external memory data structures for block compressed data.

Graph Algorithms

My interest in graph algorithms dates back to the beginning of my graduate studies. I am particularly interested in dynamic graph algorithms. Essentially for all reasonable dynamic graph problems in general graphs, there are significant gaps between upper and lower bounds [25, 26, 34–39].

Of particular interest among these problems is dynamic connectivity which has received the most attention in the literature. While there exists an amortized randomized lower bound in the cell probe model of $\Omega(\log n)$ [50], the best deterministic worst-case bound per operation is $\mathcal{O}(\sqrt{n})$ [25, 30], the best deterministic amortized bound is $\mathcal{O}(\log^2 n/\log\log n)$ [61], the best randomized worst-case bound is $\mathcal{O}(\log^5 n)$ [45], and the best randomized amortized bound is $\mathcal{O}((\log n(\log\log n)^3))$.

My own attempts of settling the deterministic amortized complexity of the problem was the motivation behind the development of mergeable dictionaries discussed in the next section.

I am also interested in the design and implementation of several graph algorithms and data structures used as building blocks in many problems defined on real world graphs.

Biased Data Structures

For many data structuring problems, access to knowledge about future operations could be quite useful in alleviating the cost of expensive operations. This access can be substituted with non-uniform access times to data using biased data structures. Our first attempt at applying this intuition was our result on mergeable dictionaries which maintains a dynamic collection of disjoint sets partitioning a totally ordered set under predecessor, split, and merge operations. While the worst-case complexity of the merge operation is $\Omega(n)$, where n is the number of elements in the universe, solutions with $\mathcal{O}(\log^2 n)$ amortized running time are straightforward [27].

We showed in [40] that a solution matching the $\Theta(\log n)$ lower bound is possible. The result relies on designing a biased data structure which can exploit the combinatorial structure of each set with respect to other sets in the collection in order to speed up complex merge operations.

In future work, I plan to extend the list of supported operations as well as extend the results to the case of partial orders or trees to improve the result of Sleator and Tarjan [52]. Interesting future work also includes generalization of the results of Grossi and Italiano for merging and splitting order decomposable sets in multiple dimensions [32] to handle arbitrary merge operations. This is closely related to open problems on maintaining layers of maxima and longest increasing subsequences dynamically which are of independent interest. Perhaps most interesting is the generalization and application of developed techniques to other problems that maintain a dynamic collection of subsets partitioning a universal set (e.g. dynamic connectivity, index maintenance).

Lastly, I still maintain a strong interest in continuing my dissertation work which was on fundamental data structuring problems.

Fundamental Data Structures

Binary Search Trees. Binary search trees (BSTs) are one of the most fundamental and well-studied data structures in computer science. While information theory dictates the worst-case running time of a single access in an n node BST to be $\Omega(\log n)$, much of the recent literature on BSTs is concerned with proving bounds on the overall running times of executing access sequences as a function of the access sequence. Numerous such bounds for various BSTs have been shown in the literature [2, 7, 7, 8, 11, 13, 14, 21, 53, 55]

These bounds are often referred to as properties (e.g. the dynamic finger property [17, 18]). In [22], we present a BST data structure that is $\mathcal{O}(1)$-competitive with respect to a constant number of any given BST data structures. In particular, we obtain a BST data structure that combines all known properties of BSTs.

A major open problem in data structures research, known as the *dynamic optimality conjecture*, asks whether an instance-optimal BST exists. It would be interesting to try to solve dynamic optimality via an in-depth study of the geometric view [20] of BSTs, possibly using tools from forbidden matrix theory.

Priority Queues. Another fundamental data structure with a long and rich history is the priority queue (heap). Unlike in the case of binary search trees, there exists no established model of computation for heaps that would allow us to compare them and derive lower bounds or tradeoff bounds. As our first attempt, in [42] we introduced the pure heap model which is both simple and captures the spirit of many heaps, and is meant to be a clean definition analogous to that of the well-established binary search tree (BST) model [57]. We show in [42] and subsequent work [41] that any heap in the pure heap model that has $\mathcal{O}(\log n)$ amortized-time Extract-Min and Insert operations must spend $\Omega(\log \log n)$ amortized time on the Decrease-Key operation. This bound is asymptotically tight as it matches the upper bounds known for numerous heaps that are captured by the pure-heap model. Extending the pure heap model to be more inclusive and facilitate the transfer of analogous results between BSTs and heaps would advance our understanding of these ubiquitous fundamental data structures.

Research Agenda

Numerous ongoing and planned follow ups to my previous research are mentioned above. In the short term, the focus of my research will continue to shift towards machine learning and further into information retrieval and data mining.

I am also very interested in complementing my applied research by maintaining an active theoretical research program which will also include emerging areas of study within distributed computation, parametrized algorithms, and streaming.

My goal is to collaborate with a wide range of researchers in computer science as well as in the long term engage in interdisciplinary research and develop collaborations with researchers in other disciplines, in particular, biological sciences.

References

[1] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988.

[2] Brian Allen and J. Ian Munro. Self-organizing binary search trees. *Journal of ACM*, 25(4):526–535, 1978.

[3] Diego Arroyuelo, Senén González, Mauricio Oyarzún, and Victor Sepulveda. Document identifier reassignment and run-length-compressed inverted indexes for improved search performance. In *Proceedings of the 36th International ACM SIGIR conference on research and development in Information Retrieval (SIGIR)*, pages 173–182, 2013.

[4] Michael A. Bender and Haodong Hu. An adaptive packed-memory array. *ACM Trans. Database Syst.*, 32(4), 2007.

[5] Roi Blanco and Alvaro Barreiro. TSP and cluster-based solutions to the reassignment of document identifiers. *Inf. Retr.*, 9(4):499–517, 2006.

[6] Dan Blandford and Guy Blelloch. Index compression through document reordering. In *Proceedings of the Data Compression Conference (DCC)*, pages 342–351, 2002.

[7] Prosenjit Bose, Karim Douïeb, Vida Dujmovic, and Rolf Fagerberg. An $O(\log \log n)$-competitive binary search tree with optimal worst-case access times. In *Proceedings of the 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 38–49, 2010.

[8] Prosenjit Bose, Karim Douïeb, Vida Dujmovic, and John Howat. Layered working-set trees. *Algorithmica*, 63(1-2):476–489, 2012.

[9] Jake Brutlag. Speed matters for google web search. *Google. June*, 2009.

[10] Jan Bulánek, Michal Koucký, and Michael Saks. Tight lower bounds for the online labeling problem. In *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC)*, pages 1185–1198, 2012.

[11] Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. A global geometric view of splaying. *CoRR*, abs/1503.03105, 2015.

[12] Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Greedy is an almost optimal deque. In *Proceedings of the 14th International Symposium on Algorithms and Data Structures (WADS)*, pages 152–165, 2015.

[13] Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. *CoRR*, abs/1507.06953, 2015.

[14] Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Self-adjusting binary search trees: What makes them tick? In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA)*, pages 300–312, 2015.

[15] Sharlee Climer and Weixiong Zhang. Take a walk and cluster genes: a TSP-based approach to optimal rearrangement clustering. In *Proceedings of the 21st International Conference (ICML)*, 2004.

[16] Sharlee Climer and Weixiong Zhang. Rearrangement clustering: Pitfalls, remedies, and applications. *Journal of Machine Learning Research*, 7:919–943, 2006.

[17] Richard Cole. On the dynamic finger conjecture for splay trees. part II: The proof. *SIAM Journal on Computing*, 30(1):44–85, 2000.

[18] Richard Cole, Bud Mishra, Jeanette P. Schmidt, and Alan Siegel. On the dynamic finger conjecture for splay trees. part I: Splay sorting log n-block sequences. *SIAM Journal on Computing*, 30(1):1–43, 2000.

[19] Pooya Davoodi, Jeremy T. Fineman, John Iacono, and Özgür Özkan. Cache-oblivious persistence. In *Proceedings of the 22th Annual European Symposium on Algorithms (ESA)*, pages 296–308, 2014.

[20] Erik D. Demaine, Dion Harmon, John Iacono, Daniel M. Kane, and Mihai Patrascu. The geometry of binary search trees. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–505, 2009.

[21] Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Patrascu. Dynamic optimality - almost. *SIAM Journal on Computing*, 37(1):240–251, 2007.

[22] Erik D. Demaine, John Iacono, Stefan Langerman, and Özgür Özkan. Combining binary search trees. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7965 of *Lecture Notes in Computer Science*, pages 388–399. Springer, 2013.

[23] Shuai Ding, Josh Attenberg, and Torsten Suel. Scalable techniques for document identifier assignment in inverted indexes. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 311–320, 2010.

[24] James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.

[25] D. Eppstein, Z. Galil, and G. F. Italiano. Improved sparsification. Technical Report 93-20, Department of Information and Computer Science, University of California, Irvine, 1993.

[26] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

[27] Martin Farach and Mikkel Thorup. String matching in Lempel-Ziv compressed strings. *Algorithmica*, 20(4):388–404, 1998.

[28] Moran Feldman, Ronny Lempel, Oren Somekh, and Kolman Vornovitsky. On the impact of random index-partitioning on index compression. *CoRR*, abs/1107.5661, 2011.

[29] Amos Fiat and Haim Kaplan. Making data structures confluently persistent. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA)*, pages 537–546, 2001.

[30] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput*, 14(4):781–798, 1985.

[31] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. *ACM Transactions on Algorithms*, 8(1):4, 2012.

[32] Roberto Grossi and Giuseppe F. Italiano. Efficient splitting and merging algorithms for order decomposable problems. *Inf. Comput.*, 154(1):1–33, 1999.

[33] Lisa Hellerstein, Özgür Özkan, and Linda Sellie. Max-throughput for (conservative) k-of-n testing. *Algorithmica*, pages 1–24, 2015. doi: 10.1007/s00453-015-0089-4.

[34] M. Henzinger and V. King. Fully dynamic 2-edge connectivity algorithm in polygarithmic time per operation. Technical Report SRC 1997-004a, Digital, 1997.

[35] Monika Rauch Henzinger and Valerie King. Fully dynamic biconnectivity and transitive closure. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 664–672, 1995.

[36] Monika Rauch Henzinger and Valerie King. Maintaining minimum spanning trees in dynamic graphs. In *Proceedings of the 24th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1256 of *Lecture Notes in Computer Science*, pages 594–604. Springer, 1997.

[37] Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.

[38] Monika Rauch Henzinger and Mikkel Thorup. Improved sampling with applications to dynamic graph algorithms. In *Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1099 of *Lecture Notes in Computer Science*, pages 290–299. Springer, 1996.

[39] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[40] John Iacono and Özgür Özkan. Mergeable dictionaries. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 6198 of *Lecture Notes in Computer Science*, pages 164–175. Springer, 2010.

[41] John Iacono and Özgür Özkan. A tight lower bound for decrease-key in the pure heap model. *CoRR*, abs/1407.6665, 2014.

[42] John Iacono and Özgür Özkan. Why some heaps support constant-amortized-time decrease-key operations, and others do not. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 8572 of *Lecture Notes in Computer Science*, pages 637–649. Springer, 2014.

[43] David S. Johnson, Shankar Krishnan, Jatin Chhugani, Subodh Kumar, and Suresh Venkatasubramanian. Compressing large boolean matrices using reordering techniques. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 13–23, 2004.

[44] Andrew Kane and Frank Wm Tompa. Distribution by document size. *The 11th International Workshop on Large-Scale and Distributed Systems for Information Retrieval*, 2014.

[45] Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142, 2013.

[46] Anagha Kulkarni and Jamie Callan. Document allocation policies for selective searching of distributed indexes. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management (CIKM)*, pages 449–458, 2010.

[47] Gal Lavee, Ronny Lempel, Edo Liberty, and Oren Somekh. Inverted index compression via online document routing. In *Proceedings of the 20th International Conference on World Wide Web (WWW)*, pages 487–496, 2011.

[48] Daniel Lemire, Owen Kaser, and Eduardo Gutarra. Reordering rows for better compression: Beyond the lexicographic order. *ACM Trans. Database Syst.*, 37(3):20, 2012.

[49] Scott R. McAllister, Peter A. DiMaggio Jr., and Christodoulos A. Floudas. Mathematical modeling and efficient optimization methods for the distance-dependent rearrangement clustering problem. *J. Global Optimization*, 45(1):111–129, 2009.

[50] Mihai Patrascu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput*, 35(4):932–963, 2006.

[51] Wann-Yun Shieh, Tien-Fu Chen, Jean Jyh-Jiun Shann, and Chung-Ping Chung. Inverted file compression through document identifier reassignment. *Inf. Process. Manage.*, 39(1):117–131, 2003.

[52] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci*, 26(3):362–391, 1983.

[53] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 235–245, 1983.

[54] Steve Souders. Velocity and the bottom line. O'Reilly Media, July 2009.

[55] Chengwen Chris Wang, Jonathan Derryberry, and Daniel Dominic Sleator. $O(\log \log n)$-competitive dynamic binary search trees. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 374–383, 2006.

[56] Rui-Sheng Wang, Yong Wang, Xiang-Sun Zhang, and Luonan Chen. Detecting community structure in complex networks by optimal rearrangement clustering. In *International Workshops on Emerging Technologies in Knowledge Discovery and Data Mining (PAKDD)*, volume 4819 of *Lecture Notes in Computer Science*, pages 119–130. Springer, 2007.

[57] Robert E. Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM Journal on Computing*, 18(1):56–67, 1989.

[58] Dan E. Willard. Maintaining dense sequential files in a dynamic environment (extended abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC)*, pages 114–121, 1982.

[59] Dan E. Willard. Good worst-case algorithms for inserting and deleting records in dense sequential files. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 251–260, 1986.

[60] Dan E. Willard. A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time. *Inf. Comput.*, 97(2):150–204, 1992.

[61] Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1757–1769, 2013.

[62] Hao Yan, Shuai Ding, and Torsten Suel. Inverted index compression and query processing with optimized document ordering. In *Proceedings of the 18th International Conference on World Wide Web (WWW)*, pages 401–410, 2009.