# Input-Output Efficient Kernelization

Riko Jacob* Tobias Lieber† Matthias Mnich‡

Preprocessing (data reduction or kernelization) is used universally in almost every practical computer implementation that deals with NP-hard problems. By now, kernelization has emerged as an independent field of research However, the deep theoretical results in the field of kernelization have yet to make their transition to contribute in solving real world optimization problems. Solving instances of these problems often boils down to traversing graphs in a structured way, but those graphs are often *huge*: massive graphs arise naturally in many applications, such as geographic information systems, web modeling, and telecommunications research. Applications working in those domains have to process terabytes of data, yet the internal memories (main memory, RAM) of computers can only keep a fraction of these huge data sets. For example, recent crawls of social networks produce graphs on the order of 720 million nodes and 140 billion edges [2]. During the processing, the applications need to access the external storage, such as hard disks. One such access can be up $10^7$ times slower than a RAM access. Thus, when working with such large data sets, the transfer of data between internal and external memory (number of I/Os), and not the internal memory computation, is often the bottleneck. Therefore, *I/O-efficient* algorithms can lead to considerable run-time improvements, and they received considerable attention (see, e.g., the survey [3]).

Kernelization is well-understood in the *von Neumann* model which assumes uniform memory access costs. Actual machines, however, increasingly deviate from this model. While waiting for a memory access, modern microprocessors can execute a number of register operations. Even worse, in an external-memory (EM) setting where the input graphs is too big to fully fit in the internal memory, certain accesses have to be served by hard disks, thus incurring an I/O loss factor up to seven orders of magnitude. That I/O-efficient algorithms can significantly outperform internal-memory algorithms, has been thoroughly demonstrated by Ajwani, Dementiev and Meyer [1]: they designed and implemented external-memory BFS that ran on web-crawl based graphs in a few *days* rather than a few *months*.

We attempt to narrow the gap between theory and practice by introducing the concept of *I/O-efficient kernelization*. That is, we will give algorithms $\mathcal{K}$ that with $\mathcal{O}(\mathsf{sort}(n))$ I/Os compress any instance $x$ with parameter $k$ of a problem $\Pi$ to an equivalent instance $x'$ with parameter $k'$, such that $|x'|, k' \leq g(k)$ for some computable function $g$. Here, $\mathsf{sort}(N)$ is the number of I/Os required to sort $N$ data items in the I/O-Model.Thus, compared to the standard concept of kernelization as it is currently used, next to $|x'|, k'$ being small we additionally require $\mathcal{K}$ to be highly efficient by only using a small number of I/Os. Importantly, our I/O-efficient kernelization algorithms build on two very similar phenomena that hold for all practically relevant scenarios, namely that $k \ll n$ and $\mathsf{sort}(n) \ll n$ for most reasonable problem sizes. Hence, we can expect I/O-efficient kernelizations to perform more efficient than standard kernelizations with $\mathcal{K}$ only bounded by some polynomial in $n$, in particular for very large input sizes $n$.

We go on to demonstrate the relevance and wide applicability of our approach. Mainly, we prove that large classes of combinatorial optimization problems, parameterized by an objective value $k$, admit I/O-efficient compression schemes to kernels of size $\mathcal{O}(k)$, on all sparse graph classes that are characterized by forbidden (apex-)minors or forbidden topological minors. There are numerous examples of important graph classes that are covered by our results, such as planar graphs, graphs of bounded genus, or graphs of bounded maximum degree.

## References

[1] D. Ajwani, R. Dementiev, and U. Meyer. A computational study of external-memory BFS algorithms. In *SODA 2006*, pages 601–610, 2006.

[2] U. of Milano Laboratory for Web Algorithmics. Datasets. `http://law.di.unimi.it/webdata/fb-current/`, Oct 2012.

[3] J. S. Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Comput. Surv.*, 33(2):209–271, June 2001.

*Institute for Theoretical Computer Science, ETH Zürich, Switzerland, `rjacob@inf.ethz.ch`

†Institute for Theoretical Computer Science, ETH Zürich, Switzerland, `tobias.lieber@inf.ethz.ch`

‡Cluster of Excellence MMCI, Saarbrücken, Germany, `mmnich@mmci.uni-saarland.de`