My Publications until 2020

Andrej Sajenko

This is a survey of my publications arranged by topics.

Space-efficient (Graph) Algorithms

Sorting and Ranking of Self-Delimiting Numbers with Applications to Tree Isomorphism [8]

Assume that an N-bit sequence S of k self-delimiting numbers is given as input. We present space-efficient algorithms for sorting, dense ranking and (competitive) ranking S on the word RAM model with word size $\Omega(\log N)$ bits. Our algorithms run in $O(k + \frac{N}{\log N})$ time and use $O(N)$ bits. The sorting algorithm returns the given numbers in sorted order, stored within a bit-vector of N bits, whereas our ranking algorithms construct data structures that allow us subsequently to return the (dense) rank of each number x in S in constant time if the position of x in S is given together with x.

As an application of our algorithms above we give an algorithm for tree isomorphism, which runs in $O(n)$ time and uses $O(n)$ bits on n-node trees. The previous best linear-time algorithm for tree isomorphism uses $\Theta(n \log n)$ bits.

Simple 2^f-Color Choice Dictionaries [5]

A c-color choice dictionary of size $n \in \mathbb{N}$ is a fundamental data structure in the development of space-efficient algorithms that stores the colors of n elements and that supports operations to get and change the color of an element as well as an operation choice that returns an arbitrary element of that color. For an integer $f > 0$ and a constant $c = 2^f$, we present a word-RAM algorithm for a c-color choice dictionary of size n that supports all operations above in constant time and uses only $nf + 1$ bits, which is optimal if all operations have to run in $o(n/w)$ time where w is the word size.

In addition, we extend our choice dictionary by an operation union without using more space.

Linear-Time In-Place DFS and BFS in the Restore Model [6]

We present an in-place depth first search (DFS) and an in-place breadth first search (BFS) that runs on a word RAM in linear time such that, if the adjacency arrays of the input graph are given in a sorted order, the input is restored after running the algorithm. To obtain our results we use properties of the representation used to store the given graph and show several linear-time in-place graph transformations from one representation into another.

Extra Space during Initialization of Succinct Data Structures and Dynamical Initializable Arrays [4]

Many succinct data structures on the word RAM require precomputed tables to start operating. Usually, the tables can be constructed in sublinear time. In this time, most of a data structure is not initialized, i.e., there is plenty of unused space allocated for the data structure. We present a general framework to store temporarily extra buffers between the real data so that the data can be processed immediately, stored first in the buffers, and then moved into the real data structure after finishing the tables. As an application, we apply our framework to Dodis, Pătraşcu, and Thorup's data structure (STOC 2010) that emulates c-ary memory and to Farzan and Munro's succinct encoding of arbitrary graphs (TCS 2013). We also use our framework to present an in-place dynamical initializable array.

Space-efficient FPT Algorithms

FPT-space Graph Kernelizations [7]

Let n be the size of a parametrized problem and k the parameter. We present polynomial-time kernelizations for Cluster Editing/Deletion, Path Contractions and Feedback Vertex Set that run with $O(\texttt{poly}(k)\log n)$ bits and compute a kernel of size polynomial in k. By first executing the new kernelizations and subsequently the best known polynomial-time kernelizations for the problem under consideration, we obtain the best known kernels in polynomial time with $O(\texttt{poly}(k)\log n)$ bits.

Our kernelization for Feedback Vertex Set computes in a first step an approximated solution, which can be used to build a simple algorithm for undirected s-t-connectivity (USTCON) that runs in polynomial time and with $O(\texttt{poly}(k)\log n)$ bits.

Space-Efficient Vertex Separators for Treewidth [3]

For n-vertex graphs with treewidth $k = O(n^{1/2-\epsilon})$ and an arbitrary $\epsilon > 0$, we present a word-RAM algorithm to compute vertex separators using only $O(n)$ bits of working memory. As an application of our algorithm, we give an $O(1)$-approximation algorithm for tree decomposition. Our algorithm computes a tree decomposition in $c^k n(\log\log n)\log^* n$ time using $O(n)$ bits for some constant $c > 0$.

We finally use the tree decomposition obtained by our algorithm to solve Vertex Cover, Independent Set, Dominating Set, MaxCut and q-Coloring by using $O(n)$ bits as long as the treewidth of the graph is smaller than $c' \log n$ for some problem dependent constant $0 < c' < 1$.

Temporal Graphs

Two Moves per Time Step Make a Difference [1]

A temporal graph is a graph whose edge set can change over time. We only require that the edge set in each time step forms a connected graph. The temporal exploration

problem asks for a temporal walk that starts at a given vertex, moves over at most one edge in each time step, visits all vertices, and reaches the last unvisited vertex as early as possible. We show in this paper that every temporal graph with n vertices can be explored in $O(n^{1.75+\varepsilon})$ time steps for arbitrary $\varepsilon > 0$ provided that either the degree of the graph is bounded in each step or the temporal walk is allowed to make two moves per step. This result is interesting because it breaks the lower bound of $\Omega(n^2)$ steps that holds for the worst-case exploration time if only one move per time step is allowed and the graph in each step can have arbitrary degree. We complement this main result by a logarithmic inapproximability result and a proof that for sparse temporal graphs (i.e., temporal graphs with $O(n)$ edges in the underlying graph) making $O(1)$ moves per time step can improve the worst-case exploration time at most by a constant factor.

Multistage Problems on a Global Budget [2]

Time-evolving or temporal graphs gain more and more popularity when studying the behavior of complex networks. In this context, the multistage view on computational problems is among the most natural frameworks. Roughly speaking, herein one studies the different (time) layers of a temporal graph (effectively meaning that the edge set may change over time, but the vertex set remains unchanged), and one searches for a solution of a given graph problem for each layer. The twist in the multistage setting is that the solutions found must not differ too much between subsequent layers. We relax on this already established notion by introducing a global instead of the local budget view studied so far. More specifically, we allow for few disruptive changes between subsequent layers but request that overall, that is, summing over all layers, the degree of change is moderate. Studying several classical graph problems (both NP-hard and polynomial-time solvable ones) from a parameterized complexity angle, we encounter both fixed-parameter tractability and parameterized hardness results. Somewhat surprisingly, we find that sometimes the global multistage versions of NP-hard problems such as Vertex Cover turn out to be computationally more tractable than the ones of polynomial-time solvable problems such as Matching.

References

[1] Thomas Erlebach, Frank Kammer, Kelin Luo, Andrej Sajenko, and Jakob T. Spooner. Two moves per time step make a difference. In *Proc. 46th International Colloquium on Automata, Languages, and Programming, (ICALP 2019)*, volume 132 of *LIPIcs*, pages 141:1–141:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.141.

[2] Klaus Heeger, Anne-Sophie Himmel, Frank Kammer, Rolf Niedermeier, Malte Renken, and Andrej Sajenko. Multistage problems on a global budget. *CoRR*, abs/1912.04392, 2019. arXiv:1912.04392.

[3] Frank Kammer, Johannes Meintrup, and Andrej Sajenko. Space-efficient vertex separators for treewidth. *CoRR*, abs/1907.00676, 2019. arXiv:1907.00676.

[4] Frank Kammer and Andrej Sajenko. Extra space during initialization of succinct data structures and dynamical initializable arrays. In *Proc. 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume

117 of *LIPIcs*, pages 65:1–65:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.65.

[5] Frank Kammer and Andrej Sajenko. Simple 2^f-color choice dictionaries. In *Proc. 29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *LIPIcs*, pages 66:1–66:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.66.

[6] Frank Kammer and Andrej Sajenko. Linear-time in-place DFS and BFS on the word RAM. In *Proc. 11th International Conference on Algorithms and Complexity (CIAC 2019)*, volume 11485 of *LNCS*, pages 286–298. Springer, 2019. doi:10.1007/978-3-030-17402-6_24.

[7] Frank Kammer and Andrej Sajenko. FPT-space graph kernalizations. *CoRR*, 2020. arXiv:2007.11643.

[8] Frank Kammer and Andrej Sajenko. Sorting and ranking of self-delimiting numbers with applications to tree isomorphism. *CoRR*, 2020. arXiv:2002.07287.