DFG Project Report: Space-efficient Algorithms

Frank Kammer

Frank.Kammer@mmi.thm.de

February 11, 2025

1 General Information

DFG reference number: KA 4663/1-2
Project number: 379157101
Project title: Space-efficient Algorithms
Name of the applicant: Frank Kammer
Official address: Technische Hochschule Mittelhessen, Wiesenstraße 14, 35390 Gießen
Name(s) of the co-applicants: None
Name(s) of the cooperation partners: None
Reporting period (entire funding period): 01.01.2018–31.07.2024 (with 4 months break)

2 Summary

2.1 English

Graphs arise in many applications, from social networks to transportation and logistics. Often, these graphs are constructed from massive data sets, commonly referred to as *Big Data*. As applications are faced with ever-increasing amounts of data, researchers tackled the problem from various directions. One such direction is the field of so-called *space-efficient algorithms*, which for a given problem aim to maintain the runtime of standard solutions while significantly reducing the memory requirements, which typically means using a linear number of bits. This is motivated by the observation that algorithms using a sublinear amount of space tend to have impractical runtimes, together with a lower bound of Barnes et al. [SIAM J. Comput., 1998], who showed that directed s-t-connectivity can not be solved in polynomial time with $o(n/\sqrt{\log n})$ bits in certain models.

Prior to the research project, only a handful of space-efficient algorithms were known such as those for standard graph traversals like depth-first search and breadth-first search. These algorithms use a linear number of bits while (almost) maintaining an optimal linear runtime. We have extended the toolbox of space-efficient algorithms with powerful frameworks such as algorithms for constructing so-called *tree decompositions*, a structure commonly used in the design of parameterized algorithms for NP-hard problems. Other results include algorithms for special classes of graphs such as planar graphs and graphs that can be categorized by so-called *forbidden minors*. One such result is a so-called *graph-coarsening* framework that allows us to execute various algorithms space-efficiently with a trade-off in solution quality. We have obtained results not only for static graphs, but also in the dynamic settings. First, for the construction of efficient and succinct data structures that provides so-called *minor operations* (delete vertices as well as delete/contract edges) in planar graphs. Minor operations are useful in numerous applications. Using the aforementioned graph-coarsening framework we are able to construct this data structure space-efficiently. Second, space-efficient results for so-called *exploration* and *multi-stage problems* on *temporal graphs*, i.e.,

graphs where the set of edges changes over discrete time steps. For practical applications, we applied our knowledge of space-efficient techniques to design a winning solver for the PACE challenge 2024 on upper bounding the parameter of so-called *twinwidth*, in addition to implementing various space-efficient algorithms and data structures in a library.

2.2 German

Graphen kommen in vielen Anwendungen vor, von sozialen Netzwerken bis hin zu Transport und Logistik. Häufig werden diese Graphen aus riesigen Datensätzen erstellt, die als *Big Data* bezeichnet werden. Da die Anwendungen mit immer größeren Datenmengen konfrontiert werden, wurden Lösungen aus verschiedensten Richtungen entwickelt. Eine dieser Forschungsrichtungen sind die so genannten *platzeffizienten Algorithmen*, die für ein bestimmtes Problem darauf abzielen, die Laufzeit von Standardlösungen beizubehalten und gleichzeitig den Speicherbedarf erheblich zu reduzieren, was in der Regel bedeutet, dass eine lineare Anzahl von Bits verwendet wird. Dies wird durch die Beobachtung motiviert, dass Algorithmen, die eine sublineare Menge an Speicherplatz verwenden, zu enormen Laufzeiten tendieren, zusammen mit einer unteren Schranke von Barnes et al. [SIAM J. Comput., 1998], dass gerichtete s-t-Konnektivität in bestimmten Modellen nicht in polynomieller Zeit mit $o(n/\sqrt{\log n})$ Bits gelöst werden kann.

Vor dem Forschungsprojekt waren nur wenige platzeeffiziente Algorithmen bekannt, zum Beispiel für platzeffiziente Varianten von Tiefen- und Breitensuche. Diese genannten platzeffizienten Algorithmen verwenden eine lineare Anzahl von Bits, während sie die optimale lineare Laufzeit (fast) beibehalten. Das Projekt hat die Toolbox für platzeffiziente Algorithmen um mächtige Frameworks erweitert, wie zum Beispiel Algorithmen für die Konstruktion so genannter *Baumzerlegungen*, eine Struktur, die häufig beim Entwurf parametrisierter Algorithmen für NP-schwere Probleme verwendet wird. Andere Ergebnisse umfassen Algorithmen für spezielle Klassen von Graphen wie planare Graphen und Graphen, die durch sogenannte *verbotene Minoren* kategorisiert werden können. Ein solches Ergebnis ist ein sogenanntes *Graph-Coarsening*-Framework, das es ermöglicht, verschiedene Algorithmen platzeffizient auszuführen, mit einem leichten Verlust in der Lösungsqualität. Die Ergebnisse des Projekts umfassen nicht nur statische Graphen, sondern auch dynamische. Erstens für die Konstruktion effizienter und succincter Datenstrukturen, die sogenannte *Minor-Operationen* (Knotenlöschung sowie das Löschen/Zusammenziehen von Kanten) in planaren Graphen bereitstellen. Minor-Operationen sind in zahlreichen Anwendungen nützlich. Mit Hilfe des oben erwähnten Graph-Coarsening Frameworks kann die genannte Datenstruktur platzeffizient konstruiert werden. Zweitens, platzeffiziente Ergebnisse für sogenannte *Explorations-* und *Multi-Stage-Probleme* auf *temporalen Graphen*, das heißt auf Graphen bei denen sich die Menge der Kanten über diskrete Zeitschritte ändert. Für praktische Anwendungen wurden mit Hilfe von platzeffizienten Strategien ein erfolgreicher Solver für die PACE-Challenge 2024 entworfen, der eine obere Schranke für den Parameter der so genannten Twinwidth berechnet, sowie verschiedene platzeffiziente Algorithmen und Datenstrukturen in einer Bibliothek implementiert.

3 Progress Report

In the following sections we give a summary of all results published as part of the DFG project "Space-Efficient Algorithms" (379157101). Throughout this report we use n to denote the number of vertices of a graph and m to denote the number of edges of a graph under consideration.

3.1 Space-efficient Algorithms (01.01.2018 – 31.03.2021)

In this section we give an overview of the goals we proposed for the first part of the project. For each goal we present our related results. The first part of this projected was proposed for two full time researchers (the applicant and one doctoral student), but was only approved for one doctoral student. We want to additionally note that during this first phase of project there were various restrictions due to the world-wide Corona pandemic. This mostly affected possible international collaborations and conference attendance, and included a 3-month extension of the first part of the

project. The subsubsections below are numbered analogously to the subsubsections of the objectives in the proposal.

3.1.1 New data structures for space-efficient algorithms

We proposed to research new succinct and compact data structures that can be used as utilities for space-efficient algorithms. In the past, most research in the field of succinct/compact data structures focussed only on the space-requirement of the final data structure [7, 48, 49]. Typically, an analysis of the runtime and space usage during the construction step is not analyzed, or only roughly, e.g., arbitrary polynomial time. In the rest of the subsection we present two related results. Note that most of the results we present in later sections are realized with the help of space-efficient data structures developed during the research process for these results, and thus are at least tangentially related to this subsection.

Kammer and Sajenko [40] present a general framework for constructing various succinct data structures while using less space than standard approaches. Roughly speaking, many succinct data structures use lookup tables to answer queries. If using such a data structure, one first has to wait for the tables to be initialized. The framework of Kammer and Sajenko allows using these data structures without waiting for the lookup table initialization, as it is constructed on-the-fly. To give an example, for succinct dictionary data structures such tables contain precomputed answers to queries for tiny bit strings, typically of size $\leq \frac{\log n}{2}$ [33]. The same technique can be used in other succinct data structures, such as succinctly encoding trees [49], planar graphs [48] or minor closed graphs [7]. Considering space requirements, storing these tables usually requires a sublinear amount of space, but the space that is required during the construction step can be (super)linear. The presented framework mitigates this problem and applies the results to the construction of well-known data structures such as a data structure for c-ary memory emulation [13] and a succinct encoding of arbitrary graphs [20].

A *choice dictionary* is a data structure that is initialized for a universe $U = \{1, \ldots, \ell\}$ of consecutive integers that dynamically manages a set $S \subset U$ while providing the following operations: insert and delete operations, containment queries, iteration of S, and an operation called *choice* that returns some element of S. The first succinct choice dictionary was introduced by Hagerup and Kammer [26]. Following that, Kammer and Sajenko [41] extend the choice dictionary to additionally store colors (integer values) while providing filtered operations in the form of iterating only over elements of a certain color, and an analogous colored choice operation. Their colored choice dictionary requires only one additional bit over the information theoretical lower bound. When the number of colors is constant, all operations run in optimal $O(1)$ time. The choice dictionary by Kammer and Sajenko requires the number of colors to be 2^c for some integer c. A following work by Hagerup [24] improved upon this by allowing the number of colors to be an arbitrary (constant) integer.

3.1.2 Space-efficient polynomial-time algorithms

One of the stated goals was to make well-known polynomial-time algorithms space-efficient while maintaining the optimal (known) runtime. The main idea we sketched in the proposal was to construct a succinct data structure for storing planar graphs such that the construction step is space-efficient. This was achieved by a more general result of Kammer and Meintrup [35] that constructs a succinct encoding of minor-closed graphs, which includes planar graphs. We present that work in Subsection 3.2.1. We have also obtained results for space-efficient isomorphism algorithms, which we present in Subsection 3.2.2.

In the following, we now present a result of Kammer and Sajenko [40] for an in-place depth-first search (DFS) and breadth-first (BFS). Research of space-efficient DFS and BFS began with a first result that enabled executing a DFS in time $O((n+m)\log n)$ while using $O(n)$ bits of space [17]. The current best DFS runs in $O((n+m)\log^* n)$ time and uses $O(n)$ bits [25], and the current best BFS runs in $O(n+m)$ time and uses $O(n)$ bits [11]. Other results include finding biconnected components [34] and st-numbering [12]. All these results assume that the graph is given in read-only

memory, and the space is measured in the working memory. In contrast to this setting, Kammer and Sajenko [40] present results for linear-time DFS and BFS that are executed in-place, while using only a constant number of words in the working memory. In other words, they assume that an input graph is given such that we are allowed to change the memory occupied by the input graph. To realize these results, they present several techniques that transform the graph representation in place. Prior to this result, no in-place BFS or DFS existed that runs in linear-time. In a different setting where one is only allowed to rotate the adjacency arrays of the input graph, there exists an $O(n^3 \log n)$-time in-place DFS [10].

3.1.3 Space-efficient parameterized algorithms for NP-hard problems

In our proposal we stipulated that we can implement various fixed parameter tractable algorithms (FPT algorithms) space-efficiently. One of the main goals was to design a space-efficient algorithm that finds a tree decomposition of width $O(k)$ with k the treewidth of the input graph. The techniques we sketched were based on a well-known recursive algorithm of Reed [50]. We present this result in more detail in the following.

Kammer, Meintrup and Sajenko [38] present a space-efficient approximation algorithm for graphs of treewidth k, based on a well-known $(f(k)n \log n)$-time $(8k + 6)$-approximation algorithm due to Reed [50]. The main procedure of this approximation algorithm is to recursively construct so-called *balanced separators* of roughly size k, which are vertex sets whose removal splits the graph into multiple, roughly equally sized, connected components. Their new algorithm approximates the treewidth by the same factor as Reed's original result, but improves the space usage to $O(n)$ bits (for $k < n^{1/2-\epsilon}$ for any $\epsilon < 0$) and additionally reduces the runtime to $f(k)n \log \log n \log^* n$ by combining the algorithm with a $(2^O(k)n)$-time algorithm for a 5-approximation of the treewidth [8]. There exist many different treewidth approximation algorithms, with the previously mentioned 5-approximation being the first results that approximates the treewidth within a constant factor while maintaining a runtime that is single-exponential in k, and linear in n. The current state-of-the art is a 2-approximation due to Korhonen [44] that runs single-exponential in k, and linear in n. Other space-efficient results include the polynomial time and sublinear-space tree-decomposition algorithm of Izumi and Otachi [32] for graphs of low treewidth, such as planar graphs, and the logspace result of Elberfeld, Jakoby and Tantau [16].

The key contribution of the algorithm of Kammer, Meintrup and Sajenko is a space-efficient balanced-separator algorithm based on network-flow techniques. They also show that one can implement classical dynamic programming techniques on tree decompositions using only $O(n)$ bits, while maintaining the standard runtime, ignoring poly-logarithmic terms. Additionally, Kammer and Sajenko [43] found various FPT algorithms in the form of kernelizations that run in polynomial time and use a sublinear amount of space, in particular, they require $f(k) \log n$ bits for some function f that depends only on the parameter k under consideration. Their work includes algorithms for so-called vertex cover, path contraction and cluster editing.

3.1.4 Space-efficient approximation algorithms and heuristics

We proposed to research approximation algorithms for, e.g., vertex cover or similar NP-hard problems under the aspect of space-efficiency. In particular, we planned to find a space-efficient implementation of the well-known polynomial-time approximation scheme (PTAS) of Baker [3]. Using this PTAS one can construct an $(1 + \epsilon)$-approximation for many of NP-hard problems on planar graphs for any $\epsilon > 0$. The PTAS has a runtime of $(f(\epsilon)n)$. The algorithm roughly works by partitioning the input planar graph G into k-outerplanar graphs G' via a simple BFS procedure, with $k = O(1/\epsilon)$. For each such graph G' one constructs a tree decomposition of width $O(k)$ followed by using standard dynamic programming techniques to solve the problem at hand exactly for G'. For each such G' the solutions are merged to form an approximate solution for G. Note that constructing a tree decomposition for a k-outerplanar graph of width $O(k)$ can easily be done in linear time.

We have as-of-writing not yet achieved a space-efficient implementation of Baker's PTAS. One

of the intermediate steps we have not yet been able to focus on is the requirement of an embedding of the planar graph. If such an embedding is not given, it has to be computed. In a standard setting this is unproblematic, as practical linear-time embedding algorithms exist that use $\Theta(n \log n)$ bits [46]. But in a space-efficient setting, one needs to compute this embedding space-efficiently, i.e., with $O(n)$ bits. We want to note that we have unpublished results for implementing Baker's PTAS if an embedding is given as the input, but consider it only of minor interest. One can easily implement it space-efficiently using standard techniques, e.g., a space-efficient BFS to construct the k-outerplanar graphs. For each k-outerplanar graph the standard linear-time tree-decomposition algorithm can be implemented space-efficiently with $O(kn)$ bits if an embedding is given, as it consists of a simple traversal of the faces. The dynamic-programming step of Baker's PTAS can be implemented by techniques of Kammer, Sajenko and Meintrup [38]. If no embedding is given, an alternative implementation is possible that adds an extra factor of $O(\log n)$ to the runtime by using the space-efficient variant of Reed's algorithm described in the previous section. To summarize, we can run Baker's PTAS space-efficiently in $\Theta(n \log n)$ time when no embedding is given, or $\Theta(n)$ time when an embedding is given. We consider this result only of interest if it would be published as an application of a space-efficient planar-embedding algorithm. Finding a space-efficient planar embedding algorithm is part of our current research.

3.1.5 Compare techniques to make graph and geometric algorithms space-efficient

The grant proposal briefly mentions research techniques that would enable the implementation of geometric algorithms space-efficiently. We did not focus on those algorithms.

3.1.6 Implementations of space-efficient algorithms

One goal of the project was to practically implement theoretical results for space-efficient algorithms. For this purpose, Kammer and Sajenko, together with bachelor and master students they supervised, implemented various algorithms and data-structures in a library for space-efficient algorithms [42]. The library is written in `C++` and contains, e.g., succinct-dictionary data structures, space-efficient BFS and DFS variants and data structures to efficiently store recursive sub-graphs. Practical experiments showed that under certain conditions, e.g., for very large graphs, the space-efficient solutions use both less space and provided faster execution times compared to standard algorithms [47].

3.2 Space-efficient Algorithms 2 (01.08.2021 – 31.07.2024)

We now discuss the goals we proposed in the proposal for the grant for a second part of the project. We want to note that the proposal was aimed to include two doctoral students, but was funded only for one student. Due to this, we were unable to research all proposed topics. Nonetheless, we will shortly mention the proposed research in these fields.

3.2.1 New data structures for space-efficient algorithms

Analogous to the proposed research for the first part of the project, we planned to continue research on novel data structures that are useful as utilities for space-efficient data algorithms. One of the main goals was to design a data structure that allows to maintain a minor of a given input graph. A prior space-efficient result exists for storing subgraphs of a given input graph [27]. The techniques we sketched in the proposal were intended for maintaining such a minor in the case that the input graph is a tree. Kammer and Meintrup [36] have shown a more general result that maintains such a minor for all planar graphs. All common access operations such as neighborhood and degree queries are provided in optimal runtime, in addition to providing (amortized) constant time dynamic operations in the form of vertex and edge deletions, and edge contractions. Before this result no such succinct (or compact) encodings existed for planar graphs that provide minor operations, and the best state-of-the art data structures for providing minor operations in (amortized) constant time required $\Theta(n \log n)$ bits of space [31].

Additionally, Kammer, Meintrup and Sajenko [39] presented a new succinct dictionary data structure that provides the common operations of rank and select for a string of self-delimiting numbers. Also, they provided a sorting algorithm that sorts an N-bit string S that consists of k self-delimiting numbers in $O(N/\log N+k)$ time and $O(N)$ bits. Rank and select data structures are well-researched for integer sequences [49], but prior to this work no result existed for providing these operations for sequences of self-delimiting numbers. Analogously, sorting integers and strings is well-researched, but no result existed that specifically works with self-delimiting numbers. Intuitively speaking, sorting self-delimiting numbers can be thought of as an "in-between" case, as each element represents an integer, but each element is stored as a binary string of variable length. To sort strings in the general case, a logarithmic factor is required, as it is well-known that the lower bound for comparison based sorting is $\Omega(n \log n)$, with n the number of elements. Kammer, Meintrup and Sajenko instead achieve a running time that is linear in the number of elements by splitting the sorting process in two steps: first, sorting the small self-delimiting numbers is reduced to integer sorting, and second, sorting the remaining large self-delimiting numbers is reduced to string sorting (via a comparison based sort). As there can only be a few large numbers, the logarithmic factor of the second step becomes asymptotically irrelevant.

3.2.2 Algorithms, recognition, and properties on/of special graph classes

For special graph classes, such as planar graphs, we proposed to research new space-efficient isomorphism algorithms and space-efficient planar-embedding algorithms. During the research project Kammer, Meintrup and Sajenko [39] have designed space-efficient isomorphism algorithms for trees and (colored) forests. An extended version of their work [37], currently under review for a special issue of IWOCA at *Theoretical Computer Science*, contains algorithms for the isomorphism of (maximal) outerplanar graphs. All aforementioned algorithms run in optimal linear time and use $O(n)$ bits. As far as planar-embedding algorithms are concerned, so far we were unable to realize a linear time and bit algorithm for computing a planar embedding. A main obstacle we currently focus on is the need for a special DFS tree that common embedding algorithms [46, 9] require to be realized in linear time. Roughly speaking, while we can execute a DFS on planar graphs in linear time and bits, the embedding algorithms require sorting the edges of each vertex according to a certain orientation of the DFS tree. Since we assume the input graph to be in read-only memory, a copy of the input graph is required. Thus, all standard embedding algorithms use $\Theta(n \log n)$ bits just for this initial sorting step. We want to note that Kammer and Meintrup [36] have shown that their data structure can be used to construct an embedding of outerplanar graphs. We believe that techniques used in their work can be extended to construct so-called *SPQR-trees*, which partitions a (planar) graph into 3-connected components. We consider this as the first step to find an embedding for planar graphs. As of writing we have not finalized these ideas.

Graph coarsening is a general technique to replace an input graph G by a graph G' that closely resembles G (e.g., is a minor of G), but contains far fewer vertices and/or edges. Related to this topic, Kammer and Meintrup [35] presented a coarsening algorithm for planar graphs called *cloud partition* that runs in time $O(n)$ time and uses $O(n)$ bits of space. The technique is based on partitioning the vertices V of a given graph $G = (V, E)$ into $\Theta(n/\log n)$ vertex sets called *clouds* of size $\Theta(\log n)$ using a greedy algorithm based on the iterative execution of a BFS. The technique they present is similar in spirit to the well known r-division for planar graphs [23], which partitions the edges of a planar graph $\Theta(n/r)$ such that each set of the partition induces a connected subgraph (called *piece*) containing at most r vertices while bounding the number of vertices that are incident to edges that belong to multiple pieces by $O(n/\sqrt{r})$. Using arguments based on planarity they show that clouds are either connected and of size $\Theta(\log n)$, or structurally simple and can be treated as special cases. As an application of this coarsening technique they show that they can construct a so-called *recursive separator decomposition*, which can then be used to construct a well-known succinct encoding for planar graphs of Blelloch and Farzan [7] in linear time and bits, improving both the runtime and the space requirement by a factor of $\Theta(\log n)$. They also show that their coarsening technique can be used to construct a so-called *tree decomposition* of width $O(\sqrt{n} \log n)$ in linear time and bits, almost matching the folklore lower bound on the width of $\Omega(\sqrt{n})$ (a planar grid graph).

Prior to this work, a comparable result by Izumi and Otachi [32] construct a tree decomposition of width $O(k\sqrt{n}\log n)$ for graphs of treewidth $k \leq \sqrt{n}$ (and planar graphs) in sublinear space and polynomial-time (of large polynomial degree). They further generalize their results to H-minor-free graphs. In an extended version Kammer and Meintrup, together with Hammer [28], verified the easy use of their cloud partitioning by practical experiments, showing that for many cases, such as for real-world road networks, a simplified version of their algorithm is sufficient, that is, the special cases mentioned above hardly occur.

3.2.3 Approximation algorithms

We proposed to further research Baker's PTAS [3] with the goal to implement it in optimal runtime while using only $O(n)$ bits and without relying on an embedding of the planar graph. We did not get any further results in this direction, not least because the project was only granted for one doctoral student. For further information on our preliminary results in this direction see Subsection 3.1.4 and our research on planar graph embeddings in Subsection 3.2.2.

3.2.4 Temporal Graphs

Temporal graphs are graphs where the edge set changes over discrete time while the vertex set remains the same. We proposed research of temporal graph algorithms under the aspect of space-efficiency. During the project we were able to publish multiple works on temporal graphs. While some of them contain results on space-efficiency, this was not yet the main focus of these publications. We give an overview of these results.

Prior to this project, Erlebach, Hoffmann and Kammer [18] presented results for the *temporal exploration problem* (TEXP), which asks for a time-respecting walk (called *exploration schedule*) that visits all vertices. For TEXP they presented results in the form of upper- and lower bounds, as well as complexity results. They showed that for connected temporal graphs, i.e., temporal graphs that are connected at each time step, an exploration schedule can always be found that spans $O(n^2)$ time steps, which matches their shown lower bound of $\Omega(n^2)$ time steps for connected temporal graphs. This has spawned research into finding ways to break the lower bound, e.g., by restricting the graph class.

Erlebach, Kammer, Luo and Sajenko [19] showed results that break this lower bound when the degree of each vertex is bounded by a constant in each time step, or when we are allowed to traverse two edges per time step. For these cases, they showed one can always find an exploration schedule that spans $O(n^{1.75})$ time steps.

A problem related to exploration is the *rendezvous problem*, which asks for two agents to meet in a graph. On static graphs, this problem has received a considerable amount of interest in the last decades [1, 2, 22]. Dogeas, Erlebach, Kammer, Meintrup and Moses [15] extended the rendezvous problem to the so-called *temporal rendezvous problem*, which asks for two agents to meet in a temporal graph. They show that two agents can always rendezvous within $O(n^{1+\epsilon})$ time steps in connected temporal graphs, with any fixed $\epsilon > 0$, almost matching their presented lower bound of $\Theta(n \log n)$ time steps. Along the way, they show various results for exploration. Their results for exploration make use of the symmetries in the temporal graph; in particular, they show that a set of vertices, a so-called *orbit*, can be explored in $O(n^{1+\epsilon})$ time steps, for any fixed $\epsilon > 0$. Intuitively, an orbit is a maximal set of vertices that are perfectly symmetric. Every (temporal) graph can be uniquely partitioned into its set of r orbits. Based on this, they show that every connected temporal graph can be explored in time $O(rn^{1+\epsilon})$. For small r this is notably faster than the previously presented lower bound of $\Omega(n^2)$ time steps for exploration in general connected temporal graphs. In an extended version [14], they also present randomized algorithms for exploring orbits that significantly reduces both the runtime and space requirements of their deterministic techniques for exploration. To give some details, the deterministic algorithm creates a matrix with one row for each automorphism of the temporal graph, thus temporal graphs exists where this table contains $\Omega(n!)$ rows. Intuitively, the randomized algorithm skips this step entirely by virtually choosing a row uniformly at random and computing it on-the-fly. Additionally, the automorphisms themselves

need not be stored or constructed using this randomized approach.

Heeger, Himmel, Kammer, Niedermeier, Renken and Sajenko [30] presented results for so-called *multistage optimization* problems on temporal graphs. Roughly speaking, they want to maintain a small solution, e.g., for a vertex cover or a cluster editing solution, t each time step, while minimizing the changes necessary to transform solutions between time steps. Each transformation, i.e., adding or removing a vertex from the solution, is associated with a cost that is bounded by a total global budget, and in addition the size of the solution at each time step is also associated with a cost. They show various complexity results for the multistage setting, e.g., FPT algorithms for vertex cover. As far as space-efficiency is concerned, they also present a space-efficient version of this FPT algorithm with a space bound sublinear in n and the lifetime ℓ of the given temporal graph. Comparable prior results [4, 5, 21] have focused on minimizing the changes between time steps; a setting where only a local budget is considered.

3.2.5 Bioinformatics

We proposed to research problems that commonly occur in bioinformatics, such as the so-called *global alignment problem*, which is a special string matching problem. We were unable to focus on finding space-efficient solutions to common problems in the field of bioinformatics.

3.2.6 Implementation of space-efficient algorithms

During the second part of this project, we implemented various space-efficient algorithms in a `rust` library such as common succinct dictionaries (rank/select and the previously described choice dictionary) and the space-efficient graph coarsening algorithm for planar graph [35]. The library is available on github [29].

Researchers from the University of Applied Sciences (Frank Kammer and Johannes Meintrup) and researchers from the Goethe University (Leonhardt, Dell, Haak, Meyer, and Penschuck) formed a team to participate in the *Parameterized Algorithms and Computational Experiments* (PACE) Challenge 2023 [6]. The PACE challenge is a competition between research groups to implement efficient software for problems that are interesting in terms of parameterized complexity, but often lack practical results in terms of practical implementations. In 2023, the problem to be solved was *twinwidth*, a parameter measuring how close a graph is to a co-graph. The aforementioned team achieved the first place in the category of designing the best heuristic solver, and the second place in the category of designing the best exact solver [45]. As the servers that were used for benchmarking the submitted solvers intentionally restricted the available memory to 8 GB, space-efficient techniques were necessary to design an efficient winning solver. Combined with large graph instances used to evaluate the (heuristic) solvers, this made it necessary to implement memory-conscious solutions. The winning heuristic solver is based on a greedy heuristic that uses a table structure to keep track of previously processes merges and edge data. A naive implementation of such a table has $O(n^2)$ entries, which is too large for the massive graphs of the challenge. The winning team devised a data structure based on union-find techniques that uses less space with a small cost in the runtime.

3.3 Summary of our results and deviations from the original concept

We now list the subsections for which we met our stated goals. We designed new data structures for space-efficient algorithms, including a general framework for constructing succinct data structures [40], in addition to new dictionary data structures [41] and sorting algorithms [39] (Subsection 3.1.1 and Subsection 3.2.1). Furthermore, we obtained new space-efficient polynomial-time algorithms, in particular we achieved our goal of constructing a space-efficient data structure for maintaining a minor of a given input graph [36]. Arguably, we superseded our goal, as we aimed to find such a data structure for trees, but designed a data structure for planar graphs (Subsection 3.1.2).

One of our main goals was to design a space-efficient tree-decomposition algorithm that constructs a tree decomposition of width at most a constant of the treewidth. To this end, we designed

a space-efficient vertex separator algorithm [38] that enabled us to implement Reed's tree decomposition algorithm [50]. In addition, we showed that we can implement classical dynamic programming techniques based on tree decompositions in a space-efficient way. We extended our work on parameterized algorithms with several sublinear-space kernelization algorithms [43] (Subsection 3.1.3).

We obtained algorithms for special graph classes, such as space-efficient coarsening of minor-closed graphs, and isomorphism algorithms for trees, forests, and (maximal) outerplanar graphs [39]. Not only that, but we have extended these results with applications such as embedding outerplanar graphs and constructing tree decompositions of planar graphs in linear time (Subsection 3.2.2).

As far as temporal graphs are concerned, we have obtained three key results as outlined in Subsection 3.2.4. The main challenge was to find any solutions for temporal exploration [15, 19] and multistage problems [30]. For both problems, we present space-efficient solutions [14, 30]. As for the practical results (Subsection 3.1.6 and Subsection 3.2.6), we have implemented various data structures and algorithms as open source libraries [29, 42]. In addition, we used our knowledge of (space-)efficient algorithms to contribute to winning the PACE 2023 challenge [45].

3.4 Outlook

We now provide a short outlook for future research. As described in Subsection 3.1.4, we have not yet finalized our ideas for embedding planar graphs space-efficiently. We consider an embedding algorithm to one of the challenges we have proposed for this project. We believe that the data structure of [36] can be used as a basis for implementing an embedding algorithm. First, we plan to use the techniques to design a space-efficient SPQR-tree algorithm. As noted above, both the SPQR-tree algorithm and the embedding algorithm require some ordering of the adjacency lists of the input graph. We believe that this step can be implemented virtually using techniques from [36].

Additionally, our focus for the future is the research of efficient temporal-graph algorithms. To this end, we have applied for a new grant for a project we call *Efficient Temporal-Graph Algorithms (ETGA)*. We want to extend our research beyond space efficiency by also considering classical runtime efficiency and so-called *sublinear-time algorithms* that are algorithms that consider only a tiny fraction of the input data. We believe that this is in line with our general goal of developing techniques that are optimized for large input sizes, for example in the context of Big Data.

4 References

[1] Steve Alpern. The rendezvous search problem. *SIAM Journal on Control and Optimization*, 33(3):673–683, 1995.

[2] Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55. Springer Science & Business Media, 2006. doi:10.1007/b100809.

[3] Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. doi:10.1145/174644.174650.

[4] Evripidis Bampis, Bruno Escoffier, Michael Lampis, and Vangelis Th. Paschos. Multistage matchings. In *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, volume 101 of *LIPIcs*, pages 7:1–7:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.SWAT.2018.7.

[5] Evripidis Bampis, Bruno Escoffier, and Alexandre Teiller. Multistage knapsack. *J. Comput. Syst. Sci.*, 126:106–118, 2022. doi:10.1016/J.JCSS.2022.01.002.

[6] Max Bannach and Sebastian Berndt. Parameterized algorithms and computational experiments (pace) 2023: Twinwidth, 2023. URL: https://pacechallenge.org/2023.

[7] Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In *Combinatorial Pattern Matching*, pages 138–150. Springer, 2010. doi:10.1007/978-3-642-13509-5_13.

[8] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A c^k n 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

[9] John M. Boyer and Wendy J. Myrvold. On the cutting edge: Simplified o(n) planarity by edge addition. *J. Graph Algorithms Appl.*, 8(3):241–273, 2004. doi:10.7155/JGAA.00091.

[10] Sankardeep Chakraborty, Anish Mukherjee, Venkatesh Raman, and Srinivasa Rao Satti. A Framework for In-place Graph Algorithms. In *Proc. 26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.13.

[11] Sankardeep Chakraborty, Anish Mukherjee, and Srinivasa Rao Satti. Space efficient algorithms for breadth-depth search. In *Fundamentals of Computation Theory - 22nd International Symposium, FCT 2019, Copenhagen, Denmark, August 12-14, 2019, Proceedings*, volume 11651 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2019. doi:10.1007/978-3-030-25027-0_14.

[12] Sankardeep Chakraborty, Venkatesh Raman, and Srinivasa Rao Satti. Biconnectivity, st-numbering and other applications of DFS using o(n) bits. *J. Comput. Syst. Sci.*, 90:63–79, 2017. doi:10.1016/J.JCSS.2017.06.006.

[13] Yevgeniy Dodis, Mihai Pătraşcu, and Mikkel Thorup. Changing base without losing space. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 593–602. ACM, 2010. doi:10.1145/1806689.1806770.

[14] Konstantinos Dogeas, Thomas Erlebach, Frank Kammer, Johannes Meintrup, and William K. Moses Jr. Exploiting automorphisms of temporal graphs for fast exploration and rendezvous. *CoRR*, abs/2312.07140, 2023. arXiv:2312.07140.

[15] Konstantinos Dogeas, Thomas Erlebach, Frank Kammer, Johannes Meintrup, and William K. Moses Jr. Exploiting automorphisms of temporal graphs for fast exploration and rendezvous. In *Proc. 51st International Colloquium on Automata, Languages, and Programming, (ICALP 2024)*, volume 297 of *LIPIcs*, pages 55:1–55:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.55.

[16] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of bodlaender and courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 143–152. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.21.

[17] Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient basic graph algorithms. In *Proc. 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *LIPIcs*, pages 288–301. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

[18] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. In *Proc. 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015), Part I*, volume 9134 of *LNCS*, pages 444–455. Springer, 2015. doi:10.1007/978-3-662-47672-7_36.

[19] 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.

[20] Arash Farzan and J. Ian Munro. Succinct encoding of arbitrary graphs. *Theor. Comput. Sci.*, 513:38–52, 2013. doi:10.1016/j.tcs.2013.09.031.

[21] Till Fluschnik, Rolf Niedermeier, Valentin Rohm, and Philipp Zschoche. Multistage vertex cover. *Theory Comput. Syst.*, 66(2):454–483, 2022. doi:10.1007/S00224-022-10069-W.

[22] Shmuel Gal. Rendezvous search on the line. *Oper. Res.*, 47(6):974–976, 1999. doi:10.1287/OPRE.47.6.974.

[23] Michael T. Goodrich. Planar separators and parallel polygon triangulation. *J. Comput. Syst. Sci.*, 51(3):374–389, 1995. doi:10.1006/JCSS.1995.1076.

[24] Torben Hagerup. A constant-time colored choice dictionary with almost robust iteration. In *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *LIPIcs*, pages 64:1–64:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.MFCS.2019.64.

[25] Torben Hagerup. Space-efficient DFS and applications to connectivity problems: Simpler, leaner, faster. *Algorithmica*, 82(4):1033–1056, 2020. doi:10.1007/s00453-019-00629-x.

[26] Torben Hagerup and Frank Kammer. Succinct choice dictionaries. *CoRR*, abs/1604.06058, 2016. arXiv:1604.06058.

[27] Torben Hagerup, Frank Kammer, and Moritz Laudahn. Space-efficient Euler partition and bipartite edge coloring. *Theor. Comput. Sci.*, 754:16–34, 2019. doi:10.1016/j.tcs.2018.01.008.

[28] Nina Hammer, Frank Kammer, and Johannes Meintrup. Space-efficient graph coarsening with applications to succinct planar encodings. *CoRR*, abs/2205.06128, 2022. arXiv:2205.06128.

[29] Nina Hammer, Frank Kammer, and Johannes Meintrup. Star: Library for space- and time-efficient algorithms. In *GitHub*, 2023. URL: https://github.com/thm-mni-ii/StaR.

[30] Klaus Heeger, Anne-Sophie Himmel, Frank Kammer, Rolf Niedermeier, Malte Renken, and Andrej Sajenko. Multistage graph problems on a global budget. *Theor. Comput. Sci.*, 868:46–64, 2021. doi:10.1016/J.TCS.2021.04.002.

[31] Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, Eva Rotenberg, and Piotr Sankowski. Contracting a planar graph efficiently. In *Proc. 25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *LIPIcs*, pages 50:1–50:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.50.

[32] Taisuke Izumi and Yota Otachi. Sublinear-Space Lexicographic Depth-First Search for Bounded Treewidth Graphs and Planar Graphs. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *LIPIcs*, pages 67:1–67:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.67.

[33] Guy Joseph Jacobson. *Succinct static data structures*. PhD thesis, Carnegie Mellon University, USA, 1988. AAI8918056. doi:10.5555/915547.

[34] Frank Kammer, Dieter Kratsch, and Moritz Laudahn. Space-efficient biconnected components and recognition of outerplanar graphs. *Algorithmica*, 81(3):1180–1204, 2019. doi:10.1007/S00453-018-0464-Z.

[35] Frank Kammer and Johannes Meintrup. Space-efficient graph coarsening with applications to succinct planar encodings. In *Proc. 33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248 of *LIPIcs*, pages 62:1–62:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ISAAC.2022.62.

[36] Frank Kammer and Johannes Meintrup. Succinct planar encoding with minor operations. In *Proc. 34th International Symposium on Algorithms and Computation (ISAAC 2023)*, volume 283 of *LIPIcs*, pages 44:1–44:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.44.

[37] Frank Kammer, Johannes Meintrup, and Andrej Sajenko. Sorting and ranking of self-delimiting numbers with applications to outerplanar graph isomorphism. *CoRR*, abs/2002.07287, 2020. Invited to special issue of IWOCA at Theoretical Computer Science. arXiv:2002.07287.

[38] Frank Kammer, Johannes Meintrup, and Andrej Sajenko. Space-efficient vertex separators for treewidth. *Algorithmica*, 84(9):2414–2461, 2022. doi:10.1007/S00453-022-00967-3.

[39] Frank Kammer, Johannes Meintrup, and Andrej Sajenko. Sorting and ranking of self-delimiting numbers with applications to tree isomorphism. In *Proc. 34th International Workshop on Combinatorial Algorithms (IWOCA 2023)*, volume 13889 of *LNCS*, pages 356–367. Springer, 2023. Invitation to Special Issue of TCS. doi:10.1007/978-3-031-34347-6_30.

[40] 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.

[41] 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.

[42] Frank Kammer and Andrej Sajenko. Space efficient (graph) algorithms. *GitHub repository*, 2018. URL: https://github.com/thm-mni-ii/sea.

[43] Frank Kammer and Andrej Sajenko. Space-efficient graph kernelizations. In *Proc. 18th Annual Conference on Theory and Applications of Models of Computation (TAMC 2024)*, volume 14637 of *LNCS*, pages 260–271. Springer, 2024. doi:10.1007/978-981-97-2340-9_22.

[44] Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS 2022)*, pages 184–192, 2022. doi:10.1109/FOCS52979.2021.00026.

[45] Alexander Leonhardt, Holger Dell, Anselm Haak, Frank Kammer, Johannes Meintrup, Ulrich Meyer, and Manuel Penschuck. Pace solver description: Exact (guthmi) and heuristic (guthm). In *Proc. 18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*, volume 285 of *LIPIcs*, pages 37:1–37:7. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.IPEC.2023.37.

[46] Kurt Mehlhorn and Petra Mutzel. On the embedding phase of the hopcroft and tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, 1996. doi:10.1007/BF01940648.

[47] Johannes Meintrup. Implementation and Evaluation of Space-Efficient Euler Tours and Subgraph Stacks. Master's thesis, University of Applied Sciences Mittelhessen, Germany, 2018.

[48] J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 118–126. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646100.

[49] Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, pages 233–242. ACM/SIAM, 2002.

[50] Bruce A. Reed. Finding approximate separators and computing tree width quickly. In *Proc. 24th Annual ACM Symposium on Theory of Computing (STOC 1992)*, pages 221–228. ACM, 1992. doi:10.1145/129712.129734.

5 Published Project Results

5.1 Publications with scientific quality assurance

The following publications were obtained as part of the project and have been published in peer-reviewed journals or conferences.

[1] Konstantinos Dogeas, Thomas Erlebach, Frank Kammer, Johannes Meintrup, and William K. Moses Jr. Exploiting automorphisms of temporal graphs for fast exploration and rendezvous. In *Proc. 51st International Colloquium on Automata, Languages, and Programming, (ICALP 2024)*, volume 297 of *LIPIcs*, pages 55:1–55:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.55.

[2] 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.

[3] Klaus Heeger, Anne-Sophie Himmel, Frank Kammer, Rolf Niedermeier, Malte Renken, and Andrej Sajenko. Multistage graph problems on a global budget. *Theor. Comput. Sci.*, 868:46–64, 2021. doi:10.1016/J.TCS.2021.04.002.

[4] Frank Kammer and Johannes Meintrup. Space-efficient graph coarsening with applications to succinct planar encodings. In *Proc. 33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248 of *LIPIcs*, pages 62:1–62:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ISAAC.2022.62.

[5] Frank Kammer and Johannes Meintrup. Succinct planar encoding with minor operations. In *Proc. 34th International Symposium on Algorithms and Computation (ISAAC 2023)*, volume 283 of *LIPIcs*, pages 44:1–44:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.44.

[6] Frank Kammer, Johannes Meintrup, and Andrej Sajenko. Space-efficient vertex separators for treewidth. *Algorithmica*, 84(9):2414–2461, 2022. doi:10.1007/S00453-022-00967-3.

[7] Frank Kammer, Johannes Meintrup, and Andrej Sajenko. Sorting and ranking of self-delimiting numbers with applications to tree isomorphism. In *Proc. 34th International Workshop on Combinatorial Algorithms (IWOCA 2023)*, volume 13889 of *LNCS*, pages 356–367. Springer, 2023. Invitation to Special Issue of TCS. doi:10.1007/978-3-031-34347-6_30.

[8] 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.

[9] 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.

[10] 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.

[11] Frank Kammer and Andrej Sajenko. Space-efficient graph kernelizations. In *Proc. 18th Annual Conference on Theory and Applications of Models of Computation (TAMC 2024)*, volume 14637 of *LNCS*, pages 260–271. Springer, 2024. doi:10.1007/978-981-97-2340-9_22.

[12] Alexander Leonhardt, Holger Dell, Anselm Haak, Frank Kammer, Johannes Meintrup, Ulrich Meyer, and Manuel Penschuck. Pace solver description: Exact (guthmi) and heuristic (guthm). In *Proc. 18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*, volume 285 of *LIPIcs*, pages 37:1–37:7. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.IPEC.2023.37.

5.2 Other publications and published results

The following results have not undergone scientific quality assurance such as open-source software packages or contributions to workshops.

[1] Deepak Ajwani, Rob H. Bisseling, Katrin Casel, Ümit V. Çatalyürek, Cédric Chevalier, Florian Chudigiewitsch, Marcelo Fonseca Faraj, Michael Fellows, Lars Gottesbüren, Tobias Heuer, George Karypis, Kamer Kaya, Jakub Lacki, Johannes Langguth, Xiaoye Sherry Li, Ruben Mayer, Johannes Meintrup, Yosuke Mizutani, François Pellegrini, Fabrizio Petrini, Frances Rosamond, Ilya Safro, Sebastian Schlag, Christian Schulz, Roohani Sharma, Darren Strash, Blair D. Sullivan, Bora Uçar, and Albert-Jan Yzelman. Open problems in (hyper)graph decomposition, 2023. arXiv:2310.11812.

[2] Nina Hammer, Frank Kammer, and Johannes Meintrup. Star: Library for space- and time-efficient algorithms. In *GitHub*, 2023. URL: https://github.com/thm-mni-ii/StaR.

[3] Frank Kammer and Andrej Sajenko. Space efficient (graph) algorithms. *GitHub repository*, 2018. URL: https://github.com/thm-mni-ii/sea.

[4] George Karypis, Christian Schulz, Darren Strash, Deepak Ajwani, Rob H. Bisseling, Katrin Casel, Ümit V. Çatalyürek, Cédric Chevalier, Florian Chudigiewitsch, Marcelo Fonseca Faraj, Michael R. Fellows, Lars Gottesbüren, Tobias Heuer, Kamer Kaya, Jakub Lacki, Johannes Langguth, Xiaoye Sherry Li, Ruben Mayer, Johannes Meintrup, Yosuke Mizutani, François Pellegrini, Fabrizio Petrini, Frances A. Rosamond, Ilya Safro, Sebastian Schlag, Roohani Sharma, Blair D. Sullivan, Bora Uçar, and Albert-Jan Yzelman. Recent trends in graph decomposition (dagstuhl seminar 23331). *Dagstuhl Reports*, 13(8):1–45, 2023. doi:10.4230/DAGREP.13.8.1.