Space-Efficient Graph Coarsening with Applications to Succinct Planar Encodings

Frank Kammer
THM, Technische Hochschule Mittelhessen, Giessen, Germany

Johannes Meintrup
THM, Technische Hochschule Mittelhessen, Giessen, Germany

Abstract

We present a novel space-efficient graph coarsening technique for n-vertex planar graphs G, called *cloud partition*, which partitions the vertices $V(G)$ into disjoint sets C of size $O(\log n)$ such that each C induces a connected subgraph of G. Using this partition $\mathcal{P}$ we construct a so-called *structure-maintaining minor* F of G via specific contractions within the disjoint sets such that F has $O(n/\log n)$ vertices. The combination of $(F, \mathcal{P})$ is referred to as a *cloud decomposition*.

For planar graphs we show that a cloud decomposition can be constructed in $O(n)$ time and using $O(n)$ bits. Given a cloud decomposition $(F, \mathcal{P})$ constructed for a planar graph G we are able to find a balanced separator of G in $O(n/\log n)$ time. Contrary to related publications, we do not make use of an embedding of the planar input graph. We generalize our cloud decomposition from planar graphs to H-minor-free graphs for any fixed graph H. This allows us to construct the succinct encoding scheme for H-minor-free graphs due to Blelloch and Farzan (CPM 2010) in $O(n)$ time and $O(n)$ bits improving both runtime and space by a factor of $\Theta(\log n)$.

As an additional application of our cloud decomposition we show that, for H-minor-free graphs, a tree decomposition of width $O(n^{1/2+\epsilon})$ for any $\epsilon > 0$ can be constructed in $O(n)$ bits and a time linear in the size of the tree decomposition. A similar result by Izumi and Otachi (ICALP 2020) constructs a tree decomposition of width $O(k\sqrt{n}\log n)$ for graphs of treewidth $k \leq \sqrt{n}$ in sublinear space and polynomial time.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases planar graph, H-minor-free, space-efficient, separator, tree decomposition

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.62

Related Version *Full Version*: `https://arxiv.org/abs/2205.06128`

Funding *Johannes Meintrup*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 379157101.

1 Introduction

Graphs are used to model a multitude of systems that can be expressed via entities and relationships between these entities. Many real-world problems operate on very large graphs for which standard algorithms and data structures use too much space. This has spawned an area of research with the aim of reducing the required space. Examples include large road-networks [50] or social graphs arising from interactions between users of large internet communities [19]. Therefore, it is of high interest to find compact representations of such graphs, space-efficient algorithms and other space-efficient or succinct data structures. In the following we denote by n the number of vertices of a graph under consideration and m the number of edges. An algorithm is called space-efficient if it has (almost) the same asymptotic runtime as a standard algorithm for the same problem, but uses asymptotically fewer bits. Examples include space-efficient graph searching algorithms such as depth-first search and breadth-first search, which run in linear time, but use $O(n \log n)$ bits with standard methods. Space-efficient solutions lower the space requirement to $O(n)$ bits and keep the runtime asymptotically (almost) the same. For a data structure or algorithm to be called *succinct* the space used must be $Z + o(Z)$ bits with Z being the information theoretic minimum to store the data. One of the most researched topics regarding space-efficiency are graph-traversal algorithms such as depth-first search (DFS). It is still an open research question if a linear-time DFS exists that uses $O(n)$ bits, with the current best bound of $O(n + m + \min\{n, m\} \log^* n)$ due to Hagerup [24], with $\log^* n$ being the iterated logarithm. This result is the work of gradual improvements over the span of seven years spanning multiple publications by multiple research groups, with some of the first results providing a runtime $O(m \log n)$ time due to Asano et al. [6] and a runtime of $O((n + m) \log \log n)$ due to Elmasry et al. [17]. Other typical problems that can be easily solved in settings where space is of no concern, such as storing spanning trees or simple mappings between vertices, require new problem-specific strategies when considered in the space-efficient setting.

Another interesting setting is aiming for arbitrary polynomial time runtime, but using only $o(n)$, or even $O(\log n)$ bits. In such settings information can be recomputed in polynomial time, and for problems such as mappings this is often trivial, while other problems such as deciding if two vertices are connected by a path in a graph requires the very involved algorithm due to Reingold [48], but is quite easy in a space-efficient setting. Thus, our space-efficient setting differentiates itself quite strongly from both, the well-researched sublinear-space settings and the common settings that do not regard space as limited.

Many large graphs that arise in practice have some known structural property that allows the design of specialized techniques that make use of these properties. A common such structural property is the existence of small separators. Such a *separator* is a small subset of vertices whose removal disconnects the graph if it was connected previously, or increases the number of connected components if it was not connected. For the graphs of interest to this work the separators are so-called balanced separators. For now the intuition suffices that such balanced separators split the graph in somewhat equally parts. Section 2 contains precise definitions of these terms and also defines what we consider small in regard to separators. Arguably the most well-known graphs that contain such small balanced separators are planar graphs, which are graphs that can be drawn in the plane without overlapping edges. All planar graphs contain a balanced separator of size $O(\sqrt{n})$ [42]. Similar separator theorems exist for almost-planar graphs [20] (which includes road networks) and well-formed meshes such as nearest neighbor graphs [44] and H-minor-free graphs for some fixed graph H [36]. For arbitrary separable graphs there exists a polylogarithmic approximation algorithm for finding balanced separators due to Leighton and Rao [40].

We present a novel partitioning scheme called *cloud partition* for planar graphs that partitions the vertices of a connected input graph G into connected subsets called *clouds* that induce a connected subgraph and are of size $O(\log n)$. For a cloud partition $\mathcal{P}$ constructed for G we construct a so-called *structure-maintaining minor* F of G induced by $\mathcal{P}$. For easier reading comprehension we call vertices of such a minor *nodes*. Intuitively, a node v of a structure-maintaining minor F is mapped to one or more clouds $C \in \mathcal{P}$ such that $|V(F)| = O(n/\log n)$ – the exact mapping is outlined in Section 3. A solution for some problems such as finding small separators can be found in F and then translated to an approximate solution for G. As F contains $O(n/\log n)$ nodes, the time and space bounds of linear or superlinear algorithms can be decreased by a factor of $\Omega(\log n)$ when being executed on F instead of G. We show how this speedup is especially helpful for recursive algorithms such as the recursive separator search used during the computation of the succinct representation of separable graphs due to Blelloch and Farzan [11]. Additionally, we show that small modifications of this recursive separator search can be used to find a tree decomposition of width $O(n^{1/2+\epsilon})$ for any $\epsilon > 0$ for planar graphs in $O(n)$ bits and a time linear in the size of the tree decomposition. Finally, we generalize our partitioning scheme from planar graphs to H-minor-free graphs for any fixed graph H.

One of the key points of our novel scheme is that we do not make use of a (planar) embedding of the input graph, as there is no known way to construct such an embedding with $O(n)$ bits and in linear time. It is often implied that an embedding is given alongside a planar graph in many of the publications regarding planar graphs, as it is computable in linear time [28] when space is of no concern. Additionally, it is often required that the planar graph is maximal. Both of these properties are, for example, required for the major result of the $O(\sqrt{n})$-separator theorem [42] or the so-called r-partition of Klein et al. [39], which similarly to our result partitions the input graph into regions of size $O(n/r)$. Again, if the graph is not maximal, it can be easily made maximal in linear time with the help of an embedding. In a space-efficient context we can not make use of either of these properties and are thus quite limited. For sublinear space settings there exist algorithms that produce an embedding in polynomial time with a rather large polynomial degree [5, 16]. Thus, the $O(n)$-bit setting provides a unique challenge due to the additional goal of matching the runtime of non space-efficient algorithms.

Succinct and space-efficient representations of planar graphs is a highly researched topic partially due to the practical applications. For compressing planar graphs without regards to providing fast access operations refer to the work of Keeler et al. [37] for an $O(n)$ bits representation and for a compression within the information theoretic lower bound refer to He et al. [26]. Due to Munro and Raman [45] there exists an encoding using $O(n)$ bits that allows constant-time queries which has subsequently been improved by Chiang et al. [14] to use a constant factor less space. We use the succinct representation due to Blelloch and Farzan, which allows encoding arbitrary separable graphs and subsequently allows constant-time adjacency-, neighborhood- and degree-queries [11], which builds on the work of Blanford et al. [10]. For H-minor-free graphs for a fixed graph H, their encoding takes $\Theta(n \log n)$ time and $\Theta(n \log n)$ bits using their described technique of recursive separator searches. We are able to improve it to $O(n)$ time and $O(n)$ bits. For planar graphs we only improve the space-requirement from $\Theta(n \log n)$ to $O(n)$ due to the algorithm of Goodrich [22]. Note that all mentioned publications above regarding succinct representations of planar graphs use $\Theta(n \log n)$ bits during the construction. Additionally, they assume a planar embedding is given explicitly or implicitly by making direct use of it, or referring to other publications as sub-routines that require them. For maximal planar graphs there exist special encodings. Some major research in this field is due to Aleardi et al. [1, 2, 3, 4].

The general technique of graph coarsening has been heavily used in the past for practical and theoretical algorithms [13, 18, 27, 34, 49]. A typical approach is to find a matching $E' \subset E(G)$ for a graph G and contract the edges in E' to find a minor F of G. The problem at hand is then solved on F and the solution is translated to an approximate solution for G [34]. Highly specialized practical algorithms using sophisticated data structures are implemented in the METIS [35] and SCOTCH [46] library. Other approaches such as graph coarsening on the GPU [7, 8] or graph coarsening via neural networks [12] have been developed.

For finding tree decompositions in a space-efficient manner refer to the work of Kammer et al. [32] and the work of Izumi and Otachi [30]. Izumi and Otachi showed that for a given graph G with treewidth $k \leq \sqrt{n}$ there exists an algorithm that obtains a tree decomposition of width $O(k\sqrt{n} \log n)$ in polynomial time and $O(k\sqrt{n} \log^2 n)$ bits. Note that the polynomial degree in the runtime is rather large due to the use of the well known s-t reachability algorithm of Reingold [48]. Izumi and Otachi additionally mention that for planar graphs there exists a polynomial-time and sublinear-space algorithm for finding a tree decomposition of a planar graph due to a simple recursive separator search with the result of Imai et al. [29], which present an algorithm for finding balanced separators of size $O(\sqrt{n})$ in planar graphs with sublinear space and polynomial runtime. Note that the algorithm of Imai et al. makes use of an embedding of the input graph and also uses the algorithm of Reingold. In contrast, we present an algorithm that computes a tree decomposition of width $O(n^{1/2+\epsilon})$ in time linear in the size of the tree decomposition using $O(n)$ bits. As the tree decomposition has $O(n)$ bags of size $O(n^{1/2+\epsilon})$ this results in a runtime of $O(n^{3/2+\epsilon})$.

In Section 2 we present concepts needed to understand the subsequent sections. In Section 3 we present our space-efficient graph-coarsening framework. Following that, in Section 4 and 5 we show how this scheme is used for finding separators, tree decompositions and constructing succinct encodings of planar graphs. In Section 6 we generalize our work to H-minor-free graphs. For space reasons most proofs can be found in the full version.

2 Background

We operate in the standard word RAM model of computation with word size $w = \Omega(\log n)$. This assumes the existence of read-only input memory, read/write working memory and write-only output memory. When we talk about space-usage we focus on bits used in the read/write working memory. This is a common setting for space-efficient algorithms.

We make use of common graph theoretic notations and terminology. Refer to Diestel [15] for more information. When using space-efficient graph algorithms the exact representation of the input graph is important. We assume that any input graph is given via *adjacency arrays* or an equivalent interface. Given a vertex u and index $i \leq \texttt{degree}(u)$ this allows us to determine in constant time the ith edge $\{u, v\}$ out of u. All our input graphs are assumed to be undirected with the vertices labeled from $1, \dots, n$. Such a labeling is given implicitly by the order of the adjacency arrays. Given the use of adjacency arrays we store each undirected edge $\{u, v\}$ as two directed *arcs* uv and vu. For each arc uv we call vu the *co-arc of* uv. We assume the existence of so-called *crosspointers* that allow us to find the co-arc of an arc uv in constant time. Typically, this is realized by storing an index i in addition to the vertex v in u's adjacency array. The index i then indicates the position of u in v's adjacency array. We assume w.l.o.g. that every given graph G is connected as otherwise all our techniques can be done iteratively for each connected component of G. We assume that a given input graph is in read-only memory.

A *separator* S of a graph $G = (V, E)$ is a subset of V such that its removal from V divides V into non-empty sets $A \subset V$ and $B \subset V$ so that $\{A, S, B\}$ is a partition of V with the constraint that all paths from a vertex $u \in A$ to a vertex $v \in B$ contain at least one vertex of S. If $|A| < \alpha n$ and $|B| < \alpha n$ for some $\alpha < 1$, then S is called a *α-balanced separator* or simply *balanced separator*.

A family of graphs $\mathcal{G}$ that is closed under taking vertex-induced subgraphs satisfies the *$f(\cdot)$-separator theorem* exactly if, for constants $\alpha < 1$ and $\beta > 0$, each member $G \in \mathcal{G}$ has an α-separator S of size $|S| < \beta f(n)$. We say a family of graphs $\mathcal{G}$ is *separable* exactly if it satisfies the *n^c-separator theorem* for some $c < 1$. We say a graph is separable if it belongs to a separable family of graphs. For planar graphs there exist an $O(\sqrt{n})$-separator theorem with $\alpha = 2/3$ that runs in linear time [42], later extended to graphs of bound genus with the same runtime [21]. For minor-closed graph classes excluding the complete graph K_t on t vertices for some constant t there exists an $O(n^{(2-\epsilon)/3})$-separator theorem with runtime $O(n^{1+\epsilon} + m)$ [36]. Note that this includes H-minor-free graphs for any fixed graph H.

The definition of the separable graph families are not closed under taking minors, but vertex-induced subgraphs [11]. As we specifically construct minors F of a separable graph G at various points in this publication, we require the more restrictive property that F also belongs to the same family of separable graphs as G, which holds for all graph classes explicitly mentioned in the previous paragraph.

Due to Lipton et al. [41] is it known that a separable graph class $\mathcal{G}$ has *bounded density* d for some constant d. This means that each $G \in \mathcal{G}$ contains at most dn edges. In particular, we use the well-known fact, by Euler's Formula, that for a planar graph $G = (V, E)$ it holds that $|E| \leq 3|V| - 6$. For planar bipartite graphs a stronger bound $|E| \leq 2|V| - 4$ holds.

A *tree decomposition* of a graph $G = (V, E)$ consists of a tree T and a family $\mathcal{X}$ of subsets X_w (called *bags*) of V, one for each $w \in V(T)$, such that:

1. $\bigcup_{w \in V(T)} X_w = V$
2. for all $\{u, v\} \in E$ there exists $w \in V(T)$ such that $u, v \in X_w$
3. for all $w_1, w_2, w_3 \in V(T)$, if there is a path from w_1 to w_3 that contains w_2, then $X_{w_1} \cap X_{w_3} \subseteq X_{w_2}$.

The *width* of a tree decomposition is the size of the largest bag minus one. The treewidth of a graph G is the smallest width amongst all possible tree decompositions of G.

We make use of *indexable dictionaries*, which is a structure that supports constant-time rank-select queries over a bitvector. The $\texttt{rank}(i)$ operation counts the number of occurrences of bits set to 1 before the ith index and the $\texttt{select}(i)$ operation returns the index of the *ith* bit set to 1.

▶ **Lemma 1** ([47])**.** *Given a bitvector S of length ℓ there is an indexable dictionary on S that requires $o(\ell)$ additional bits, supports rank-select queries in constant time and can be constructed in $O(\ell)$ time.*

What can be thought of a dynamic alternative to indexable dictionaries is the so-called *choice dictionary* [23]. A choice dictionary is initialized for a universe $1, \ldots, \ell$ and supports constant time `insert`, `delete`, and `contains` operations, with the latter returning `true` exactly if an element of the universe is contained in the choice dictionary. Additionally, the choice dictionary offers the operation `choice` that returns an arbitrary member and iteration over its members. The iteration outputs all members of a choice dictionary in constant time per member. All operations run in constant time and the iteration over its members is linear in the number of members. The following lemma has been adapted from [33], with some minor rephrasing to make it more clear in the context of this paper.

▶ **Lemma 2.** *There is a succinct choice dictionary initialized for the universe $1, \ldots, \ell$ that occupies $\ell + o(\ell)$ bits and provides constant-time `insert`, `delete`, `contains` and `choice` operations and constant-time (per member) iteration. The choice dictionary can be initialized in $O(\ell)$ time.*

We make use of a folklore technique called *static space allocation* allowing us to store ℓ items of varying size compactly. The following description is adapted from [31]. Each item B_k occupies d_k bits for $k \in \{1, \ldots, \ell\}$. Denote by L the amount of bits all these items totally occupy. We want to store all these items with $L + o(L)$ bits such that we can access each item B_k in $O(1)$ time for $k \in \{1, \ldots, \ell\}$. In $O(\ell + L)$ time compute the sums $s_k = k + \sum_{j=1}^{k-1} d_j$ and an indexable dictionary over a bitvector S of size $\ell + L$ with $S[i] = 1$ exactly if $i = s_k$ for $k \in \{1, \ldots, \ell\}$. The location of B_k is then equivalent to $\texttt{select}_B(k) - k$. Refer to [9, 25, 31] for more information and detailed descriptions.

For traversing graphs we use standard breadth-first search (BFS), which puts a start vertex in a first queue, then the unprocessed neighbors of the first queue in a second queue, swaps the queues and repeats. It can easily be seen that space requirement is $O(n \log n)$ bits. More precise, the BFS uses $O(\ell \log n)$ bits where ℓ is the maximum number of vertices in a queue at any point of the BFS.

In this work we need to store subgraphs G' of a given graph $G = (V, E)$. Similar to the techniques used by Hagerup et al. [25] we are able to store G' using $O(n + m)$ bits. We store a subgraph G' of G via n choice dictionaries D_v for each vertex $v \in V(G)$. Each D_v has length d_v with d_v being the degree of v in G. The choice dictionaries are stored using static space allocation. A member i in D_v then indicates the existence of the ith arc out of v's adjacency array. Another choice dictionary D' of length n can be used to mark vertices of degree > 0. Iterating over the neighborhood of a vertex $v \in V(G')$ can be done in linear time of the degree of v in G'. This additionally allows dynamic insertion and deletion of edges as long as G' remains a subgraph of G, i.e., no new edges can be added. Note that we do not directly allow the deletion of vertices. Instead, when a vertex has degree 0 in G' (due to the deletion of all incident edges) it is marked in D'. This allows to iterate over all vertices $V' = \{v \in V(G') | d_v > 0\}$ in $O(|V'|)$ time with d_v being the degree of v in G'. Additionally, an arbitrary vertex $v \in V(G')$ with $d_v > 0$ can be obtained via $D'.\texttt{choice}()$. We refer to such a structure as a *dynamic subgraph G' of G*. Note that a dynamic subgraph G' can be used to direct an edge $\{u, v\} \in E(G')$ by deleting only the arc uv or vu in G'. Using this we are able to implement the next lemma in $O(n)$ time and $O(n)$ bits using the same algorithm as described in [10].

▶ **Lemma 3.** *Let G be a separable graph. We can obtain the directed graph G' of G such that each vertex of G' has bounded in-degree (out-degree) in $O(n)$ time and $O(n)$ bits.*

3 Graph Coarsening Framework for Planar Graphs

In this section we outline a strategy for coarsening a planar graph $G = (V, E)$. The idea is to create a specific type of partition $\mathcal{P}$ of V, which we call *cloud partition* that induces a unique minor F of G, defined later. We refer to each $C \in \mathcal{P}$ as a *cloud*. Thus, a cloud is a set of vertices. In the following we describe the exact specifications that define such a cloud partition. For each $C \in \mathcal{P}$, C induces a connected subgraph of G and $|C| \leq \lceil c \log n \rceil$ for an arbitrary, but fixed constant c. We differentiate between different types of clouds $C \in \mathcal{P}$, which we define after introducing some terminology. Let $C_1, C_2 \in \mathcal{P}$ with $C_1 \neq C_2$. We call an edge $\{u, v\} \in E$ a *border edge* exactly if $u \in C_1$ and $v \in C_2$. We then refer to C_1 and C_2 as *adjacent* or *neighbors* and as *incident to* $\{u, v\}$. Furthermore, we call C a *big cloud* if $|C| = \lceil c \log n \rceil$ and a *small cloud* otherwise. We call a small cloud C a *leaf cloud* if C is adjacent to one cloud, a *bridge cloud* if it is adjacent to two clouds and a *critical cloud* if it is adjacent to at least three clouds. We call two clouds C_1, C_2 adjacent to a bridge cloud B *connected by B*. A cloud partition is created by starting with the following scheme: Initially, mark all vertices as unvisited. Run a BFS from an arbitrary unvisited vertex v. The BFS only traverses vertices marked unvisited. Each time an unvisited vertex is traversed, it is marked visited. The BFS runs until either $\lceil c \log n \rceil$ vertices are marked visited or no unvisited vertices can be reached by the BFS. In the first case a big cloud is found and in the second case a small cloud is found. Repeat this until no unvisited vertices remain, always starting a new BFS at an arbitrary unvisited vertex. Only a partition created in this way is referred to as a cloud partition. By fixing the search for unvisited vertices and the BFS algorithm, each graph has a fixed cloud partition. The next two observations are derived directly from the process of creating a cloud partition.

▶ **Observation 4.** *Let $\mathcal{P}$ be an arbitrary cloud partition created for a graph G. Then no two small clouds $C_1, C_2 \in \mathcal{P}$ are adjacent.*

▶ **Observation 5.** *Let $\mathcal{P}$ be an arbitrary cloud partition created for a graph G. Since each big cloud has size $\lceil c \log n \rceil$ for a constant c, $\mathcal{P}$ contains at most $n/\lceil c \log n \rceil$ big clouds.*

For planar graphs, the number of critical clouds can be bounded similarly. The proof is based on the fact that a planar graph has $O(n)$ edges.

▶ **Lemma 6.** *Let $\mathcal{P}$ be a cloud partition created for a planar graph G containing k big clouds. Then $\mathcal{P}$ contains $O(k)$ critical clouds.*

Proof. Consider the graph $F = (V', E')$ constructed by contracting the vertices of each cloud $C \in \mathcal{P}$ to a single vertex $v \in C$. The graph F is a minor of G with $|V'| = |\mathcal{P}|$ and each vertex $v \in V'$ represents a single cloud $C_v \in \mathcal{P}$. Two vertices $v, u \in V'$ are adjacent in F exactly if C_v and C_u are adjacent. For brevity's sake we refer to vertices of F introduced for critical clouds as critical vertices, and vertices introduced for big clouds as big vertices. Assume that all vertices that are not big vertices or critical vertices are removed in F and all edges between big vertices are removed as well. This turns F into a bipartite planar graph due to Observation 4, with big vertices on one side of the bipartition, and small and critical on the other. It is well-known that $|E'| \leq 2|V'| - 4$ for bipartite planar graphs. Denote by ℓ the number of critical vertices and k the number of big vertices in F. Note that for each critical vertex $v \in V'$ it holds $\texttt{degree}(v) \geq 3$. As such, $|E'| \geq 3\ell$. Using this the number of critical vertices in F is bounded via $3\ell \leq 2|V'| - 4 = 2k + 2\ell - 4$ and thus $\ell \leq 2k - 4$. (Note that the addition of any edges between big vertices in F can only tighten this bound.) As the number of critical vertices in F is equal to the number of critical clouds in $\mathcal{P}$ we have shown that there are at most $2k - 4$ critical clouds in $\mathcal{P}$. ◀

The next corollary follows together with Obs. 5.

▶ **Corollary 7.** *Let $\mathcal{P}$ be a cloud partition created for a planar graph G. Then $\mathcal{P}$ contains $O(n/\log n)$ critical clouds.*

Note that for leaf or bridge clouds no such bound exists as a cloud partition can contain $O(n)$ leaf and bridge clouds. Extreme examples include a cloud partition $\mathcal{P}$ with one big cloud C and $n - |C|$ leaf clouds, all adjacent to C and containing only one vertex. For bridge clouds a similar example partition $\mathcal{P}$ can be constructed, with two big clouds $C_1, C_2 \in \mathcal{P}$ and $n - (|C_1| + |C_2|)$ bridge clouds, all adjacent to both C_1 and C_2.

Next we focus on the construction of a specific weighted minor F of G with weights $w(v)$ assigned to each node $v \in V(F)$. Intuitively, F represents a minor of G that is constructed by repeatedly contracting the vertices in one or more clouds, and the weights keep track of the number of vertices that have been contracted. We call such an F a *structure-maintaining minor* of G, with the formal definition outlined shortly. As each such minor is constructed specifically for a cloud partition P we say that F is *induced by P*. We now define the properties of such a minor and follow with a sketch of a construction. Let F be a structure-maintaining minor induced by a cloud partition $\mathcal{P}$. Denote by $\mathcal{C}_u$ the set of clouds contracted to a node $u \in V(F)$. Each node $u \in V(F)$ is assigned a weight $w(u) = \sum_{C \in \mathcal{C}_u} |C|$. For each u one of the following two properties holds: (1) $|\mathcal{C}_u| = 1$, $G[C]$ is connected for $C \in \mathcal{C}_u$ and $w(u) = \lceil c \log n \rceil$ for some fixed constant c or (2) each $C \in \mathcal{C}_u$ is adjacent to exactly the same clouds, $G[\bigcup_{C \in \mathcal{C}_u} C]$ contains $|\mathcal{C}_u|$ connected components. Additionally, $\{v, w\} \in E(F)$ exactly if there exist adjacent clouds C_v and C_w with $C_v \in \mathcal{C}_v$ and $C_w \in \mathcal{C}_w$.

We now outline the construction of F. Initially, $F = (V' = \emptyset, E' = \emptyset)$. For each cloud C that is big or critical, add a node v to V' with $w(v) = |C|$. Add edges between $u, v \in V'$ to E' exactly if the clouds u and v represent are adjacent. We call v a *big node* if it was added to V' for a big cloud and *critical node* if it was added for a critical cloud. For each pair of big clouds $C_1, C_2 \in \mathcal{P}$ connected via one or more bridge clouds, let $\mathcal{B}_{C_1,C_2} \subset \mathcal{P}$ be the set of all such bridge clouds. Add a single node v to V' adjacent to the nodes u, w added to V' for C_1, C_2 and set $w(v) = \sum_{B \in \mathcal{B}_{C_1,C_2}} |B|$. We refer to such a node v as a *meta-bridge node.* For each big cloud $C \in \mathcal{P}$ adjacent to one or more leaf clouds, denote by $\mathcal{L}_C \subset \mathcal{P}$ the set of all leaf clouds adjacent to C. For each such C add a single node v to V' adjacent to the node u added to V' for C with $w(v) = \sum_{L \in \mathcal{L}_C} |L|$. We call such a node v a *meta-leaf node.*

Since we have only $O(n/\log n)$ big nodes, we also have only $O(n/\log n)$ critical, meta-bridge and meta-leaf nodes, expressed explicitly in the following lemma. Note that the weight of a meta-bridge or meta-leaf node can be bounded only by n. We next bound the number of nodes and edges of F.

▶ **Lemma 8.** *Let $\mathcal{P}$ be a cloud partition constructed for a planar graph G and $F = (V', E')$ the structure-maintaining minor induced by $\mathcal{P}$. Then $|V'| = O(n/\log n)$ and $O(|E'|) = O(|V'|)$.*

Proof. As F is a minor of G it is planar as well and so it holds that $O(|E'|) = O(|V|')$. From Obs. 5 we know that the number of big nodes in F is $k = O(n/\log n)$. From the construction of meta-leaf nodes it directly follows that each meta-leaf node is adjacent to exactly one big node and vice-versa. Therefore there are at most k meta-leaf nodes in V'. For meta-bridge nodes the proof follows analogous, consider two big nodes u, v that are adjacent to the same meta-bridge node w. Then there is no meta-bridge node $w \neq w$ adjacent to both u and v, as per the definition of meta-bridge nodes, there exist at most one meta-bridge node between two big nodes. By the fact that F is planar it follows that there are at most $O(k) = O(n/\log n)$ meta-bridge nodes in V'. Putting it all together we arrive at $|V'| = O(n/\log n)$. ◀

In the following we describe a data structure for a cloud partition and show how it can be constructed and stored in $O(n)$ time and $O(n)$ bits. We refer to this data structure as *cp-structure.* A cp-structure constructed for a planar connected graph G admits the following operations:

- `type`(v): Given a vertex v outputs the type of cloud v belongs to (big, small, critical, bridge or leaf) in $O(1)$ time.
- `cloud`(v): Given a vertex v returns all vertices of the cloud C in which v is contained in $O(|C|)$ time and $O(|C| \log n)$ space.
- `border`(v, k): Outputs if the k-th arc out of v's adjacency array is part of a border edge in $O(1)$ time, with $v \in V(G)$.

Additionally, the structure allows access to the subgraph G' of G induced by all non-border edges of $\mathcal{P}$. The graph G' admits adjacency array access to its vertices and edges. The following lemma describes the runtime and space usage of constructing a cloud partition and a cp-structure.

▶ **Lemma 9.** *Let G be a planar connected graph. We can compute a cloud partition $\mathcal{P}$ of G and a cp-structure of $\mathcal{P}$ in $O(n)$ time and $O(n)$ bits.*

Next we show how to construct a structure-maintaining minor F induced by a cloud partition $\mathcal{P}$ of a graph G in $O(n)$ time and $O(n)$ bits of space. By Lemma 8, F can be stored in $O(n)$ bits by adjacency lists. We additionally store a bi-directional mapping from each big/critical cloud C to the node $v \in V(F)$ added to $V(F)$ to represent C. This mapping is stored by a pointer at v to a single vertex $v' \in C$ and a pointer at v' to v. As there are $O(n/\log n)$ pointers to store for this bi-directional mapping we use standard pointers of size $\Theta(\log n)$. To store the pointers from the direction of a vertex $v \in V(G)$ we use static-space allocation. Note that the choice of the vertex to store these pointers (per cloud) is arbitrary, but should be fixed. We choose the vertex with the lowest label. For meta-bridge and meta-leaf nodes $v \in V(F)$ we store a pointer from v to the lowest labeled vertex v' amongst all clouds represented by the meta-bridge or meta-leaf node. The details are described in the proof of the next lemma. Next, we define a special operation on F.

- `expand`(v): Given a node $v \in V(F)$ with weight $w(v)$, first determine the cloud $\mathcal{C}$ mapped to v and then return iteratively all vertices part of the clouds $C \in \mathcal{C}$.

Later we make use of the `expand` operation when translating solutions for problems such as finding separators from F to a solution to G. The key difficulty is implementing the `expand` operation for a meta bridge v. As the clouds represented by v may induce up to $n - O(\log n)$ connected components in G, and we store only a pointer to the lowest labeled vertex $u \in V(G)$ amongst all such clouds represented by v, we are unable to simply output each cloud without additional information, or we must traverse (in the worst case) the entire graph G. The idea is to store a spanning tree that spans all vertices in clouds represented by v. As mentioned, since these clouds each induces a connected component, a spanning tree with these clouds does not exist. Therefore, we create a spanning tree by additionally using the vertices of one adjacent big cloud. Traversing all vertices represented by v can now be done by traversing the respective spanning tree. The key observation for this is that we can direct the edges of F in such a way that each vertex has up to $c = O(1)$ outgoing edges (Lemma 3). By this we are able to store all spanning trees in $O(n)$ bits due to the fact each big cloud is used only by c spanning trees. Concerning the problem of having c spanning trees that use the vertices of one big cloud, i.e., that overlap, our approach is to assign each spanning tree a color from a set of c colors. These spanning trees are then stored in c dynamic subgraphs, with each subgraph storing all spanning trees of one color.

▶ **Lemma 10.** *Let G be a connected planar graph and $\mathcal{P}$ a cloud partition of G given as a cp-structure. We can construct the structure-maintaining minor F induced by $\mathcal{P}$ in $O(n)$ time and bits such that the* `expand` *operation on $v \in V(F)$ runs in $O(w(v) + \log n)$ time.*

We refer to the combined data structure of Lemma 9 and Lemma 10 as a *cloud decomposition* $(F, \mathcal{P})$ *of* G.

4 Applications: Succinctly Encoded Planar Graphs

In this section we show how a cloud decomposition $(F, \mathcal{P})$ constructed for a planar graph G can be used to find a balanced separator of size $O(\sqrt{n \log n})$ in $O(n/\log n)$ time and $O(n)$ bits, which in turn can be used to construct a succinct encoding of planar graphs, described later. Afterwards we show how such a cloud decomposition can be used together with the search for balanced separators to compute a tree decomposition with width $O(n^{1/2+\epsilon})$ for any $\epsilon > 0$ of a planar graph G. We make use of an $O(n)$-time $O(n \log n)$-bits algorithm for finding $2/3$-balanced separators of size $O(\sqrt{n})$ in weighted planar graphs as long as no vertex has a weight more than $1/3$ times the total weight [38, 42, 43]. The next theorem shows how a balanced separator can be constructed for a planar graph with the help of a cloud decomposition. The idea is to run a slightly modified standard algorithm for finding balanced separators S on the structure-maintaining minor F and translating S to a separator S' of G.

▶ **Theorem 11.** *Let $G = (V, E)$ be a planar graph and $(F = (V', E'), \mathcal{P})$ a cloud decomposition of G. We can construct a balanced separator S of G with size $O(\sqrt{n \log n})$ in $O(n/\log n)$ time using $O(n)$ bits.*

Due to Blelloch and Farzan [11] there exists a succinct encoding of an arbitrary separable graph $G = (V, E)$ that provides constant-time adjacency, degree and neighborhood queries. The encoding is constructed via recursive application of a separator algorithm such that V is split into two sets A and B via a separator S. This recursion is continued for $G_a = G[A \cup S]$ and $G_b = G[B \cup S]$ until the remaining graphs are of size at most $\log^\delta n$ for a graph class specific constant δ, which they refer to as *mini graphs*. In several papers, what is constructed here is referred to as a *separator hierarchy*. For technical reasons the edges between vertices of S are only included in G_a and not G_b. The graph G is then encoded by a combination of these mini graphs, which in turn are further decomposed by the same recursive separator search into *micro graphs*, which are then small enough to be handled via lookup tables encoding every possible micro graph and in turn are used to encode the mini graphs. Constructing these mini and micro graphs can be done in $O(n)$ time for a planar graph G using the techniques described by Blelloch and Farzan combined with the algorithm of Goodrich for constructing a separator hierarchy [22], but we want to use only $O(n)$ bits and maintain the runtime. The main result of [11] is summed up by the next theorem.

▶ **Theorem 12** ([11])**.** *Any family of separable graphs with entropy $\mathcal{H}(n)$ can be succinctly encoded in $\mathcal{H}(n) + o(n)$ bits such that adjacency, neighborhood, and degree queries are supported in constant time.*

One important point of the construction is the fact that the number of *duplicate vertices* in each mini graph is bounded. A duplicate is a vertex that is contained in more than one mini graph. A vertex becomes a duplicate any time it is part of a separator S, as it is then contained in both G_a and G_b. The exact bound is defined by the next lemma.

▶ **Lemma 13** ([11])**.** *The number of mini graphs is $\Theta(n/\log^\delta n)$. The total number of duplicates among mini graphs is $O(n/\log^2 n)$. The sum of number of vertices of mini graph together is $n + O(n/\log^2 n)$.*

Our goal is to execute the recursive decomposition of the graph G via a cloud decomposition $(F, \mathcal{P})$ by recursively searching for separators in F until the entire weight of the remaining graphs is less than $\log^\delta n$. These small subgraphs of F then are expanded to be exactly the mini graphs of G we aim to find. For each mini graph a new cloud decomposition is constructed, and the recursive separator search is repeated for each mini graph to construct the micro graphs.

▶ **Lemma 14.** *Let G be a planar graph and $(F, \mathcal{P})$ a cloud decomposition of G. We can output all mini graphs of G in $O(n)$ time.*

The bound on the number of duplicates due to Lemma 13 can be upheld with some care.

▶ **Lemma 15.** *Let G be a planar graph and $(F, \mathcal{P})$ a cloud decomposition of G. The total number of duplicate vertices among the mini graphs of the cloud decomposition is $O(n/\log n)$.*

The full encoding in $O(n)$ time can be done by using the algorithm of Lemma 14 to output all mini graphs. For every mini graph G_m we again construct a cloud decomposition in linear time and use the algorithm of Lemma 14 to find the micro graphs of G_m. All micro graphs are encoded using the table lookup scheme outlined in [11]. All other structures needed for the encoding can be constructed in $O(n)$ time and $O(n)$ bits easily, as they are simple indexable dictionaries over vertices of the micro graphs and mini graphs or small lookup tables which can be initialized in time linear in their size, which is $O(n)$. From this description the next theorem follows.

▶ **Theorem 16.** *A planar graph G with entropy $\mathcal{H}(n)$ can be succinctly encoded in $\mathcal{H}(n)+o(n)$ bits such that adjacency, neighborhood, and degree queries are supported in constant time. The encoding can be constructed in $O(n)$ time using $O(n)$ bits.*

5 Applications: Planar Tree Decompositions

We next present a simple modification of the recursive separator search of Lemma 14 using standard techniques to output a tree decomposition of G. Let $(F, \mathcal{P})$ be a cloud decomposition of G. Each balanced separator S of F induces a bag in a tree decomposition of F via the following recursive relation. Let F' be the input graph used in the recursive calls, initially $F' = F$. Additionally, maintain a set X containing vertices contained in separators found in previous recursive calls, initially $X = \emptyset$. When a separator S is found for F that partitions $V(F)$ into three sets $\{A, S, B\}$ output the next bag of the tree decomposition of F as $S \cup X$. Continue the recursion for the input $F'[S \cup A]$ and $X = (X \cup S) \cap A$ and the input $F'[S \cup B]$ and $X = (X \cup S) \cap B$. Note that this tree decomposition of F has width $O(\sqrt{|V(F)|})$. This tree decomposition can easily be expanded to a tree decomposition of G by expanding all vertices of a bag B to their respective clouds. The width of this expanded tree decomposition is $O(\sqrt{|V(F)|}\log n) = O(n^{1/2+\epsilon})$ for any $\epsilon > 0$. Each bag can be output in $O(n^{1/2+\epsilon})$ time and there are $O(n)$ bags. This description allows the next corollary.

▶ **Corollary 17.** *Let G be a planar graph. We can compute and output the bags of a tree decomposition with width $O(n^{1/2+\epsilon})$ for any $\epsilon > 0$ in time linear in the size of the tree decomposition $O(n^{3/2+\epsilon})$ using $O(n)$ bits.*

6 Generalizing to H-Minor-Free Graphs

To generalize the previous results to H-minor-free graphs from planar graphs we must generalize cloud partitions and their induced minors. Recall that a cloud partition $\mathcal{P}$ for a planar graph G contains $O(n/\log n)$ critical clouds. Critical clouds are exactly those of size $< c \log n$ and with ≥ 3 adjacent big clouds. We define a ϕ-critical cloud as a small cloud adjacent to $\geq \phi$ big clouds. The idea of the proof is the same as for the proof of Lemma 6, but we now have a bound ϕ that depends on the graph class. The existence of the bound ϕ is due to the face that a separable graph $G = (V, E)$ has bound density d, i.e., $|E| \leq d|V|$ for some fixed constant d. In particular we show that setting ϕ as $d + 1$ gives us the desired properties outlined in the following.

▶ **Lemma 18.** *Let $\mathcal{G}$ be a H-minor-free family of graphs for some fixed graph H and let d be the maximal density of a graph in $\mathcal{G}$. There exists a cloud partition $\mathcal{P}$ for each graph $G \in \mathcal{G}$ such that $\mathcal{P}$ contains $O(n/\log n)$ ϕ-critical clouds for $\phi = d + 1$.*

It remains to show how small clouds adjacent to $< \phi$ big clouds are to be handled. For planar graphs such clouds are exactly the bridge and leaf clouds. To generalize from planar graphs to H-minor-free graphs we must in turn generalize bridge clouds similarly to the generalization from critical to ϕ-critical clouds. We call such generalized bridge clouds *ϕ-bridge clouds*, which we define as small clouds adjacent to $< \phi$, but > 2 big clouds.

Analogous to regular bridge clouds, there can be $O(n)$ such clouds in a cloud partition for an H-minor-free graph. We refer to cloud partitions with the additional labeling of clouds as ϕ-critical and ϕ-bridge as a *generalized cloud partition* $\mathcal{P}$.

To construct the structure-maintaining minor F induced by a generalized cloud partition $\mathcal{P}$ we can use the same strategy for bridge clouds and leaf clouds. For ϕ-critical clouds the strategy also remains the same as for critical clouds in regular cloud partitions. For ϕ-bridge clouds we introduce a generalized version of the meta-bridge node to F, which we call ϕ-meta-bridge node. Note that each ϕ-meta-bridge node has degree $< \phi$ in F. For adding all ϕ-meta bridge nodes we iterate over $i \in (3, \dots, \phi - 1)$. For each i we add all ϕ-meta bridges with degree i to $V(F)$. Adding all degree-i ϕ-meta bridge nodes v takes $O(in)$ time as for each neighbor of v the respective clouds must be explored. Thus, an overall time of $O(\phi^2 n)$ is needed during the construction. As described above Lemma 10, we store spanning trees for meta-bridge nodes and meta-leaf nodes. We store and construct the exact same spanning trees for the ϕ-meta-bridge nodes. These spanning trees are stored in c different dynamic subgraphs, with c being some constant, as described in Lemma 3. Storing these dynamic subgraphs takes $O(cn) = O(\phi n)$ bits of space.

▶ **Lemma 19.** *Let G be a H-minor-free graph for some fixed graph H with a generalized cloud partition $\mathcal{P}$. We can construct the minor induced by $\mathcal{P}$ in $O(\phi^2 n)$ time and $O(\phi n)$ bits with ϕ being a graph class dependent constant.*

Analogous for planar graphs, we call the combination of a generalized cloud partition $\mathcal{P}$ and a minor induced by $\mathcal{P}$ a *ϕ-cloud decomposition*. This allows us to generalize Theorem 11, Theorem 12 and Corollary 17 to H-minor-free graphs using such generalized cloud partitions. For generalizing the proofs of these theorems and the corollary we only must handle the new case of ϕ-meta-bridge nodes exceeding the weight threshold during the separator search, as all other cases work analogous. Recall that we must handle the case when a node $v \in V(F)$ has too large of a weight, and thus no balanced separator can be found. A new additional case now arises when a ϕ-meta-bridge node v exceeds the weight threshold which is handled exactly the same as meta-bridge and meta-leaf nodes. In detail, the neighborhood S of v in F is a separator that separates v from $V(F) \setminus S$. Expanding S to a separator S' of G then separates all vertices in clouds represented by v from the rest of the graph. The balanced separator S' then contains $O(\phi \log n)$ vertices. For the following theorems we make use of the linear time $O(n^{2/3})$-separator theorem for H-minor-free graphs [36].

▶ **Theorem 20.** *Let G be a H-minor-free graph for some fixed graph H, let ϕ be a graph class dependent constant and $(F, \mathcal{P})$ a generalized cloud partition of G. We can compute a balanced separator S of G with size $O(n^{2/3+\epsilon})$ for any $\epsilon > 0$ in $O(n/\log n)$ time using $O(n)$ bits.*

▶ **Theorem 21.** *A H-minor-free graph for some fixed graph H with entropy $\mathcal{H}(n)$ can be succinctly encoded in $\mathcal{H}(n) + o(n)$ bits such that adjacency, neighborhood, and degree queries are supported in constant time. The construction of the encoding of takes $O(\phi^2 n)$ time and uses $O(\phi n)$ bits with ϕ a graph class dependent constant.*

▶ **Corollary 22.** *Let G be a H-minor-free graph for some fixed graph H, ϕ a graph class dependent constant and $(F, \mathcal{P})$ a generalized cloud partition of G. We can compute a tree decomposition of G with width $O(n^{2/3+\epsilon})$ for any $\epsilon > 0$ in $O(n^{5/3+\epsilon})$ time using $O(n)$ bits.*

References

1 Luca Castelli Aleardi and Olivier Devillers. Array-based compact data structures for triangulations: Practical solutions with theoretical guarantees. *J. Comput. Geom.*, 9(1):247–289, 2018. doi:10.20382/jocg.v9i1a8.

2 Luca Castelli Aleardi, Olivier Devillers, and Gilles Schaeffer. Dynamic updates of succinct triangulations. In *Proceedings of the 17th Canadian Conference on Computational Geometry, CCCG'05, University of Windsor, Ontario, Canada, August 10-12, 2005*, pages 134–137, 2005. URL: http://www.cccg.ca/proceedings/2005/45.pdf.

3 Luca Castelli Aleardi, Olivier Devillers, and Gilles Schaeffer. Succinct representation of triangulations with a boundary. In *Algorithms and Data Structures*, pages 134–145. Springer, 2005.

4 Luca Castelli Aleardi, Olivier Devillers, and Gilles Schaeffer. Succinct representations of planar maps. *Theor. Comput. Sci.*, 408(2-3):174–187, 2008. doi:10.1016/j.tcs.2008.08.016.

5 Eric Allender and Meena Mahajan. The complexity of planarity testing. *Inf. Comput.*, 189(1):117–134, 2004. doi:10.1016/j.ic.2003.09.002.

6 Tetsuo Asano, Taisuke Izumi, Masashi Kiyomi, Matsuo Konagaya, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, Jun Tarui, and Ryuhei Uehara. Depth-first search using $o(n)$ bits. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation*, pages 553–564, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-13075-0_44.

7 Bas Fagginger Auer and Rob H Bisseling. Graph coarsening and clustering on the GPU. *Graph Partitioning and Graph Clustering*, 588:223, 2012.

8 Bas O Fagginger Auer and Rob H Bisseling. A GPU algorithm for greedy graph matching. In *Facing the Multicore-Challenge II*, pages 108–119. Springer, 2012.

9 Niranka Banerjee, Sankardeep Chakraborty, and Venkatesh Raman. Improved space efficient algorithms for BFS, DFS and applications. In *Computing and Combinatorics*, pages 119–130. Springer International Publishing, 2016.

10 Daniel K. Blandford, Guy E. Blelloch, and Ian A. Kash. Compact representations of separable graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 679–688. Society for Industrial and Applied Mathematics, 2003.

11 Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In *Proceedings of the 21st Annual Conference on Combinatorial Pattern Matching*, CPM 2010, pages 138–150, 2010.

12 Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks, 2021. arXiv:2102.01350.

13 Cédric Chevalier and Ilya Safro. Comparison of coarsening schemes for multilevel graph partitioning. In Thomas Stützle, editor, *Learning and Intelligent Optimization, Third International Conference, LION 3, Trento, Italy, January 14-18, 2009. Selected Papers*, volume 5851 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2009. doi:10.1007/978-3-642-11169-3_14.

14 Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications to graph encoding and graph drawing. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 506–515. Society for Industrial and Applied Mathematics, 2001.

15 Reinhard Diestel, Alexander Schrijver, and Paul D. Seymour. Graph theory, 2007.

16 Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 383–392, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591865.

17 Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient Basic Graph Algorithms. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 288–301, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2015.288.

18 Matthew Fahrbach, Gramoz Goranci, Richard Peng, Sushant Sachdeva, and Chi Wang. Faster graph embeddings via coarsening. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2953–2963. PMLR, 13–18 July 2020. URL: `https://proceedings.mlr.press/v119/fahrbach20a.html`.

19 Volodymyr Floreskul, Konstantin Tretyakov, and Marlon Dumas. Memory-efficient fast shortest path estimation in large social networks. *Proceedings of the International AAAI Conference on Web and Social Media*, 8:91–100, 2014. URL: `https://ojs.aaai.org/index.php/ICWSM/article/view/14532`.

20 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987. `doi:10.1137/0216064`.

21 John R Gilbert, Joan P Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs of bounded genus. *Journal of Algorithms*, 5(3):391–407, 1984. `doi:10.1016/0196-6774(84)90019-1`.

22 Michael T. Goodrich. Planar separators and parallel polygon triangulation. *J. Comput. Syst. Sci.*, 51(3):374–389, December 1995. `doi:10.1006/jcss.1995.1076`.

23 Torben Hagerup. A constant-time colored choice dictionary with almost robust iteration. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 64:1–64:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.MFCS.2019.64`.

24 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`.

25 Torben Hagerup, Frank Kammer, and Moritz Laudahn. Space-efficient Euler partition and bipartite edge coloring. *Theoretical Computer Science*, 754:16–34, 2019. Algorithms and Complexity. `doi:10.1016/j.tcs.2018.01.008`.

26 Xin He, Ming-Yang Kao, and Hsueh-I Lu. A fast general methodology for information-theoretically optimal encodings of graphs. *SIAM J. Comput.*, 30(3):838–846, 2000. `doi:10.1137/S0097539799359117`.

27 Emilie Hogan, John R. Johnson, and Mahantesh Halappanavar. Graph coarsening for path finding in cybersecurity graphs. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, CSIIRW '13. Association for Computing Machinery, 2013. `doi:10.1145/2459976.2459984`.

28 John Hopcroft and Robert Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, October 1974. `doi:10.1145/321850.321852`.

29 Tatsuya Imai, Kotaro Nakagawa, Aduri Pavan, N Variyam Vinodchandran, and Osamu Watanabe. An $O(n^{1/2+\epsilon})$-space and polynomial-time algorithm for directed planar reachability. In *2013 IEEE Conference on Computational Complexity*, pages 277–286. IEEE, 2013. `doi:10.1109/CCC.2013.35`.

30 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 *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 67:1–67:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ICALP.2020.67`.

31 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`.

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

33 Frank Kammer and Andrej Sajenko. Simple 2^f-Color Choice Dictionaries. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 66:1–66:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ISAAC.2018.66`.

34 G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. In *Supercomputing '95:Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, pages 29–29, 1995. doi:10.1109/SUPERC.1995.242800.

35 George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM JOURNAL ON SCIENTIFIC COMPUTING*, 20(1):359–392, 1998.

36 Ken-ichi Kawarabayashi and Bruce Reed. A separator theorem in minor-closed classes. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 153–162, 2010. doi:10.1109/FOCS.2010.22.

37 Kenneth Keeler and Jeffery Westbrook. Short encodings of planar graphs and maps. *Discrete Appl. Math.*, 58(3):239–252, 1995. doi:10.1016/0166-218X(93)E0150-W.

38 Philip Klein and Shay Mozes. Separators in planar graphs, 2021. URL: http://planarity.org/Klein_separators_in_planar_graphs.pdf.

39 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 505–514, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2488608.2488672.

40 T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 422–431, 1988. doi:10.1109/SFCS.1988.21958.

41 Richard J. Lipton, Donald J. Rose, and Robert E. Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16:346–358, 1977.

42 Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. doi:10.1137/0136016.

43 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986. doi:10.1016/0022-0000(86)90030-9.

44 Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, January 1997. doi:10.1145/256292.256294.

45 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

46 François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking*, pages 493–498. Springer Berlin Heidelberg, 1996.

47 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4):43–es, November 2007. doi:10.1145/1290672.1290680.

48 Omer Reingold. Undirected st-connectivity in log-space. *Electronic Colloquium on Computational Complexity (ECCC)*, January 2004. doi:10.1145/1060590.1060647.

49 Peter Sanders and Christian Schulz. Advanced multilevel node separator algorithms. In *Experimental Algorithms*, pages 294–309. Springer International Publishing, 2016.

50 Ben Strasser, Dorothea Wagner, and Tim Zeitz. Space-Efficient, Fast and Exact Routing in Time-Dependent Road Networks. In *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 81:1–81:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.81.