Succinct Representations of Separable Graphs

Guy E. Blelloch[1] and Arash Farzan[2]

[1] Computer Science Department, Carnegie Mellon University
[2] Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany
blelloch@cs.cmu.edu, afarzan@mpi-inf.mpg.de

Abstract. We consider the problem of highly space-efficient representation of separable graphs while supporting queries in constant time in the RAM with logarithmic word size. In particular, we show constant-time support for adjacency, degree and neighborhood queries. For any monotone class of separable graphs, the storage requirement of the representation is optimal to within lower order terms.

Separable graphs are those that admit a $O(n^c)$-separator theorem where $c < 1$. Many graphs that arise in practice are indeed separable. For instance, graphs with a bounded genus are separable. In particular, planar graphs (genus 0) are separable and our scheme gives the first succinct representation of planar graphs with a storage requirement that matches the information-theory minimum to within lower order terms with constant time support for the queries.

We, furthers, show that we can also modify the scheme to succinctly represent the combinatorial planar embedding of planar graphs (and hence encode planar maps).

1 Introduction

Many applications use graphs to model connectivity information and relationship between different objects. As the size of these graphs grow, the space efficiency becomes increasingly important. The structural connectivity of the Web modeled as the Web graph is an example which presently contains billions of vertices and the number is growing [1]. As a result, compact representation of such graphs for use in various algorithms has been in interest [2,3,4,5]. Planar (and almost planar) graphs which capture various structural artifacts such as road networks, form another example of graphs whose space-efficient representation is crucial due to their massive size. For all these applications, it is desirable to represent the graph compactly and be able to answer dynamic queries on the graph quickly.

A succinct representation of a combinatorial object is a compact representation of that object such that its storage requirement matches the information-theoretic space lower bound to within lower order terms, and it supports a reasonable set of queries in constant time. Succinct data structures perform under the uniform-cost word RAM-model with $\Theta(\lg n)$ word size [6][1]. Hence, the main distinction between succinct and compact representations of an object

[1] $\lg n$ denotes $\log_2 n$.

A. Amir and L. Parida (Eds.): CPM 2010, LNCS 6129, pp. 138–150, 2010.

is that unlike compact representations, the storage requirement of a succinct representation cannot be a constant factor away from the optimal and moreover, queries must perform in constant time.

Unstructured graphs are highly incompressible (see section 1.1). Fortunately however, most types of graph that arise in practice have some structural properties. A most common structural property that graphs in practice have is that they have small separators. A graph has small separators if its induced subgraphs can be partitioned into two parts of roughly the same size by removing a small number of vertices (to be defined precisely in section 2). Planar graphs (such as 2-dimensional meshes), almost planar graphs (such as road networks, distribution networks) [7,8], and most 3-dimensional meshes [9] have indeed small separators.

In this paper, we study the problem of succinct representations of separable undirected and unlabeled graphs (as defined precisely in definition 1). We present a succinct representation with a storage requirement which achieves the information-theoretic bound to within lower order terms and show constant time support for the following set of queries: adjacency queries, neighborhood queries, and degree queries. Adjacency queries on a pair of vertices x, y determines whether (x, y) is an edge. Neighborhood queries iterate through the neighbors of a given vertex x. Finally, the degree query outputs the number of incident edges to a given vertex x. A representation that supports these queries in constant time has the functionality of both an adjacency list and an adjacency matrix at the same time.

Analogous to Fredrickson's partitioning scheme for planar graphs [8], our succinct representation is based on recursive decomposition of graphs into smaller graphs. We repeatedly separate the given graph into smaller graphs to obtain small graphs of poly-logarithmic size which we refer to as by *mini-graphs*. These mini-graphs are further separated into yet smaller graphs of sub-logarithmic size which we refer to as by *micro-graphs*. Micro-graphs have small enough sizes to be catalogued and listed in a look-up table. Micro-graphs are encoded by a reference to within the look-up table. At each step that a graph is repeatedly split into two smaller subgraphs, the vertices in the separator are copied into both subgraphs. Therefore there are duplicate vertices in mini-graphs and micro-graphs. The main difficulty is to be able to represent the correspondence between duplicate vertices and the original graph vertices.

The time to construct the representation is dominated by the time needed to recursively decompose the graph into mini-graphs and micro-graphs and also by the time needed to assemble the look-up table for micro-graphs. The time for finding the separators and decomposing the graph recursively varies significantly from a family of graphs to another. For instance, there are linear time algorithms for finding separators in planar graphs and well-shaped meshes in arbitrary dimensions [7,9]. For our purposes a poly-logarithmic approximation of the separator size suffices and therefore we use Leighton-Rao's polynomial time construction [10]. The time required to assemble the look-up table depends on the maximum size of micro-trees, since we need to exhaustively list all separable

graphs modulo their isomorphism up to that size. We have a large degree of freedom on the choice of maximum size of a micro-graph, choice of $\sqrt{\frac{\lg n}{\lg \lg n}}$ as the maximum micro-graph size ensures a sub-linear look-up table construction time. Albeit, for simplicity of presentation of this paper, we will use $\frac{\lg n}{\lg \lg n}$ as the maximum micro-graph size.

1.1 Related Work

As mentioned previously, unstructured graphs are highly incompressible. A simple counting argument shows that a random graph with n vertices and m edges requires $\left\lceil \lg \binom{\binom{n}{2}}{m} \right\rceil$ bits. Blandford *et al.* [11] achieves this bound within a constant multiplicative factor for sparse graphs. Raman *et al.* [12] give a representation with a storage requirement which is roughly twice the information theory minimum and supports adjacency and neighborhood queries in constant time. Farzan and Munro [13] prove the infeasibility of achieving the information-theoretic space lower bound to within lower order terms and constant-time query support. and give a representation with a storage requirement that is a factor of $1 + \epsilon$ away from the minimum (for any constant $\epsilon > 0$).

Hence, space efficient representations of graphs with a certain combinatorial structure has been of interest: *e.g.* bounded-genus graphs [14], graphs with limited arboricity, and c-decomposable graphs [15]. A strong line of research has been on compressing planar graphs. Given a planar graph with n vertices, Turán [16] gives a $O(n)$-bit representation. Keeler and Westbrook [17] improve the space by a constant factor. He *et al.* [18] improve the first order term of space to the information-theory minimum. However, none of these consider fast support for queries.

Jacobson [19] gives a linear-space representation for planar graphs which supports adjacency queries in logarithmic time. Munro and Raman [20] gives a linear-space encoding for planar graphs in which supports queries in constant time. Chuang *et al.* [21] and subsequently Chiang *et al.* [22] improve the constant on the high order term for space. There is a vast literature on encoding subfamilies of planar graphs. Two important subfamilies are tri-connected planar graphs and triangulated planar graphs for which in a culminating work Castelli Aleardi *et al.* [23] show a succinct representation. This representation, used for general planar graphs, has a storage requirement which is a constant factor away from the optimal (and therefore is not succinct).

One important aspect in representing planar graphs has been to also represent the associated planar embedding together with the graph (*i.e.* to represent planar maps). We demonstrate our scheme yields a succinct representation for both general planar graphs and planar maps.

Blandford *et al.* [11] study space-efficient representations of separable graphs with constant time support for adjacency, degree, and neighborhood queries. However their representation is not succinct and can have a storage which is a multiplicative factor away from the optimal. We present a succinct representation for separable graphs that supports the same set of queries in constant time.

2 Preliminaries

A *separator* S in a graph $G = (V, E)$ with n vertices is a set of vertices that divides V into non-empty parts $A \subset V$ and $B \subset V$ such that $\{A, S, B\}$ is a partition of V, and no edge in G joins a vertex in A to a vertex in B.

Definition 1. *A family of graphs $\mathcal{G}$ that is closed under taking the vertex-induced subgraphs satisfies the $f(.)$-*separator theorem *[7] if there are constants $\alpha < 1$ and $\beta > 0$ such that each member graph $G \in \mathcal{G}$ with n vertices has a separator S of size $|S| < \beta f(n)$ which divides the vertices into parts A, B each of which contains at most αn vertices ($|A| \leq \alpha n, |B| \leq \alpha n$). We define a family of graphs as* separable *if it satisfies the n^c-separator theorem for some constant $c < 1$. A graph is separable if it belongs to a separable family of graphs.*

Lipton, Rose, and Tarjan [24] prove that a in family of graphs satisfying a $\left(n/(\log n)^{1+\epsilon}\right)$-separator theorem for some $\epsilon > 0$, the number of edges of a graph is linear in the number of vertices. Since separable graphs satisfy a stronger separator theorem, a separable graph has linear number of edges.

We use the dictionary data structures heavily in this work. The first data structure we need in our tool set is an indexable dictionary (ID) to represent a subset of a universe supporting membership, rank, and select queries on member elements in constant time. A membership query on a given element X determines whether x is present in the subset. A rank query on an element x reports the number of present elements less than x in the subset. Finally, a select query (which are reverse to rank queries) for a given number i reports element at rank i in the increasing order in the subset.

Lemma 1 ([12]). *Given a set S of size s which is a subset of a universe $U = \{1, \ldots, u\}$, there is an indexable dictionary (ID) on S that requires $\lg \binom{u}{s} + o(s) + O(\log \log u)$ bits and supports rank/select on elements of S in constant time (rank/select on non-members is not supported).*

Unlike IDs, *fully indexable dictionaries* (FID) support membership, rank, and select queries on both members and non-members. These are very powerful structures, as they can support predecessor queries in constant time. As a result, they are not as space-efficient as IDs.

Lemma 2 ([12]). *Given a subset S of a universe U, there is a* fully indexable dictionary **(FID)** *structure which requires $\lg \binom{|U|}{|S|} + O(|U| \log \log |U| / \log |U|)$ bits and supports rank and select queries both on members and nonmembers of S in constant time.*

3 Succinct Representation

Analogous to the compact representation of separable graphs [11], we find and remove separators recursively to decompose the graph. Given a graph G with n vertices, we find a small separator S ($|S| < \beta n^c$) whose removal divides G

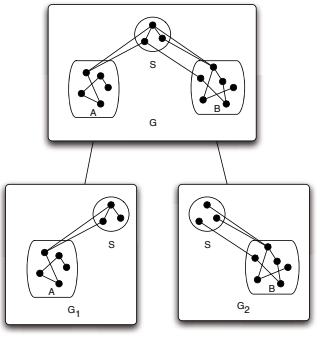

Fig. 1. Decomposition of a separable graph G into G_1, G_2

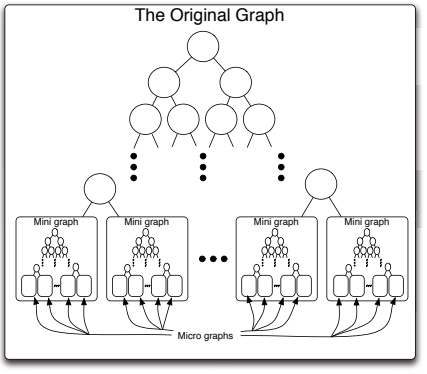

Fig. 2. A schematic view of the decomposition of a separable graph to mini-graphs and then to micro-graphs

into two parts A, B with at most αn vertices each. We obtain two induced subgraphs $G_1 = A \cup S$ and $G_2 = B \cup S$. We remove internal edges of S from G_1 (and retain them in G_2). Therefore, we obtain two subgraphs G_1, G_2 from G. Figure 1 illustrates the decomposition of an example graph G into G_1 and G_2.

We decompose G_1 and G_2 to obtain smaller subgraphs. Smaller subgraphs are in turn decomposed similarly into yet smaller subgraphs. We define a constant $\delta = 2/(1-c)$ where there are n^c-separators (definition 1). We repeat the separator-based decomposition till the subgraphs have at most $(\lg n)^\delta$ vertices where n is the number of vertices in the initial graph. We refer to these subgraphs with at most $(\lg n)^\delta$ vertices as *mini-graphs*.

Mini-graphs are further decomposed in the same fashion. Each mini-graph is decomposed repeatedly until the number of vertices in subgraphs is at most $\lg n / \lg \lg n$. We refer to these subgraphs with at most $\lg n / \lg \lg n$ vertices as *micro-graphs*. Figure 2 illustrates the decomposition into mini and micro graphs.

The graph representation consists of the representations of mini-graphs which in turn consist of the representations of micro-graphs. Micro-graphs are small enough to be catalogued by a look-up table. Vertices in separators are duplicated by each iteration of the decomposition and therefore there can be many occurrences of a single vertex of the original graph across different mini-graphs and/or micro-graphs.

Each occurrence of a vertex receives three labels: a label within the containing micro-graph which we refer to as by *micro-graph label*, a label within the containing mini-graph which we refer to as by *mini-graph label*, and finally a label in the entire graph which we refer to as by *graph label* and is visible from outside our abstract data type for the graph. Queries indicate vertices using their graph labels. Dictionary structures of lemmas 1 and 2 are used to maintain the relationship between duplicates of a single vertex.

Combining representations of mini-graphs. We assume mini-graphs are encoded (using a scheme to be discussed shortly), we explain here how these encodings can be combined to represent the entire graph. We start by bounding the number of vertices of an individual mini-graph and their accumulative size (proof omitted due to space constrainsts):

Lemma 3. *The number of mini-graphs is* $\Theta\left(n/(\log n)^{\delta}\right)$. *The total number of duplicates among mini-graphs (*i.e. *sum of multiplicities greater than one) is* $O\left(n/\log^2 n\right)$. *The sum of the number of vertices of mini-graphs together is* $n + O\left(n/\log^2 n\right)$. □

As discussed previously the given graph is unlabeled and we pick (graph) labels for vertices. Labels of vertices with no duplicates precede labels of vertices with duplicates. Mini-graphs are scanned in order and non-duplicate vertices are assigned graph labels consecutively. Duplicate vertices in the original graph are assigned graph labels arbitrarily using the remaining labels.

To translate graph labels to/from mini-graph labels, we build a bit vector `Translate` with length equal to the sum of the number of vertices in mini-graphs. This vector spans across all mini-graphs in order containing an entry for each vertex of a mini-graph. The entry is set to zero if the corresponding vertex has no duplicates and is set to one if it has a duplicate. The fully indexable dictionary (FID) of lemma 2 is used to represent one entries over the universe of all entries in `Translate`. Support for rank and select on both zeros and ones allows us to translate between locations in `Translate` and graph labels. The space of this structure by lemmas 2,3 is $o(n)$. Figure 3 depicts an overview of these structures.

`Boundaries` is another bit vector which is encoded also using a FID. It marks the boundaries of mini-graphs in `Translate`. `Translate` and `Boundaries` together enable us to translate labels of non-duplicate vertices. Given the graph label of such a vertex, we find the corresponding location in `Translate` by a select query and then perform rank on `Boundary` to obtain the mini-graph number and the offset from the predecessor one which is the mini-graph label of that vertex. Conversely, given the mini-graph label of a non-duplicate vertex, we perform select on `boundaries` to find the start location of the mini-graph in `Translate` and add to it the mini-graph label to find the corresponding location in there. Now a rank over non-duplicates gives us the graph label.

For translating labels of duplicate vertices, we maintain other structures. `Duplicates` has a list for each duplicate vertex which contains all duplicates of the vertex as positions in `Translate`. `Duplicates` empowers us to iterate through duplicates of a vertex. `Real-names` is an array with length equal to the sum of multiplicities of duplicates vertices. Its entries contain in order the graph label of each occurrence of a duplicate vertex in `Translate`. `Real-names` allows us to determine the graph label of an occurrence of a duplicate vertex. Using these structures we can translate between graph labels and mini-graph labels of duplicate vertices. To account for the space of these structures, we note that $\Theta(\log n)$ bits are used for any occurrence of duplicate vertices of which there are $\Theta\left(n/\log^2 n\right)$ by lemma 3, and therefore the space is $\Theta(n/\log n)$ bits.

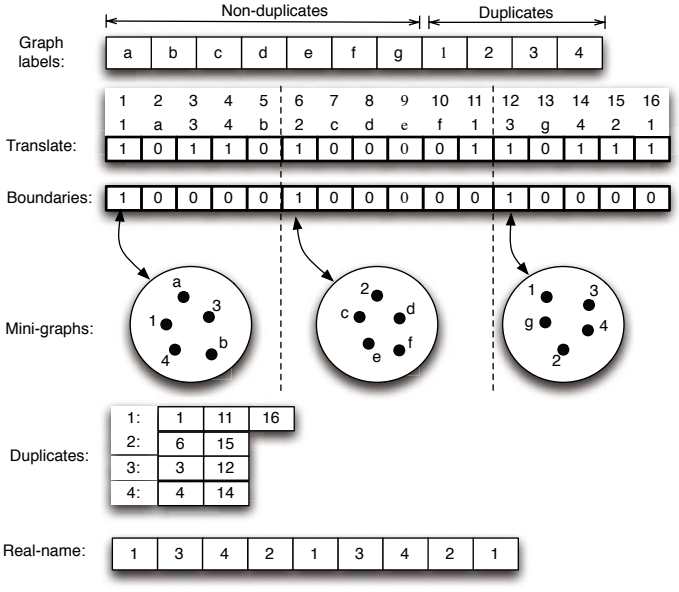

Fig. 3. Indexing structures used to translate between graph labels and mini-graph labels

Combining representations of micro-graphs. The representation of a mini-graph is composed of those of micro-graphs in the same manner as the representation of the entire graph is composed out of mini-graphs. The same set of structures are built and stored. The technical lemma in this construction is analogous to lemma 3. The details of this construction and the proof of lemma is omitted due to space constraints.

Lemma 4. *Within a particular mini-graph of m vertices, the number of micro-graphs is $\Theta((m \log\log n)/\log n)$. The total number of duplicates (*i.e. *sum of multiplicities) is $O\left((m \log\log^{1-c} n)/\log^{1-c} n\right)$. Sum of the number of vertices of micro-graphs together is $m + O\left((m \log\log^{1-c} n)/\log^{1-c} n\right)$.* □

Representations of micro-graphs. Micro-graphs have $\Theta(\log n/\log\log n)$ vertices and are encoded by an `Index` to a look-up table. The look-up table lists all possible micro-graphs with $\Theta(\log n/\log\log n)$ vertices ordered according to their numbers of vertices. The table also stores pre-computed answers to all queries of interest.

`Index` fields account for the dominant space term. Since we enumerate micro-graphs to list them in the look-up table, the length of the `Index` field matches the entropy bound for each micro-tree. Since a family of separable graphs has a linear entropy ($\mathcal{H}() = O(n)$) [11], the sum of the lengths of `Index` fields over all micro-graphs is $\mathcal{H}(\Sigma) + o(n)$ where Σ is the sum of the number of vertices of micro-graphs ($o(n)$ comes from the round-up for individual indices). Lemmas 3 and 4

show that $\Sigma = n + o(n)$ and thus the length of the encoding is $\mathcal{H}(n) + o(n)$. Since all indexes built to combine micro-graphs into mini-graphs and combine mini-graphs into an entire graph is $o(n)$ as shown, and the storage requirement of the look-up table is $o(n)$, the entire representation requires $\mathcal{H}(n) + o(n)$ bits.

We now turn to showing support for queries in constant time. The two main queries of interest are neighborhood and adjacency queries and support for degree queries is straightforward.

3.1 Neighborhood Queries

We now explain how neighbors of a vertex can be reported in constant time per neighbor. Given a vertex v by its graph label, we first determine if it has duplicates by a simple comparison. If there is no duplicates then the corresponding mini-graph and the mini-graph label are determined. If there are duplicates, we use `Duplicates` array to look-up each occurrence of the vertex one by one. Each occurrence leads us to a particular vertex in a mini-graph.

Once confined to a mini-graph and a particular vertex u therein, we determine analogously if u has duplicates across micro-graphs. If no duplicate exists, then we find the micro-graph and the micro-graph label therein and the query is answered using the pre-computed neighbors in the look-up table. In case duplicates exist, array `Duplicates` is used and each occurrence is handled analogously.

Each neighbor vertex name is a micro-graph label and should be translated to a graph label which is performed by a conversion to mini-graph label and subsequently to a graph label using `Translate, Boundaries` structures.

3.2 Adjacency Queries

We use the same approach as in [11] and direct the edges such that in the resulting graph each vertex has a bounded out-degree:

Lemma 5 ([11]). *The edges of a separable graph can be directed in linear time such that each vertex has out-degree at most b for some constant $b > 0$.*

In order to answer the adjacency query $q(u, v)$, it suffices to show how outgoing edges of a vertex can be looked-up in constant time as the (possible) edge between u, v is either directed from u to v or vice versa.

We cannot store the directed graph as the space requirement would exceed our desirable bound. We only store the direction of a sub-linear number of edges. The look-up table remains undirected and intact, and thus it does not reflect the direction of any edge.

We add the following structures to enable constant time look-up for out-going edges. In a mini-graph, for each vertex v with duplicates, we store b vertices that are endpoints of edges going out of v ($\Theta(\log \log n)$ bits each). Similarly, in the entire graph, for each vertex u with duplicates we explicitly store b endpoints of edges going out of u ($\Theta(\log n)$ bits each).

More importantly, for each vertex with duplicates across different mini-graphs, we store, in Structure `Duplicate-components`, the mini-graph numbers in which

it has a duplicate . We cannot simply list mini-graphs in `Duplicate-components` as we must support membership queries. We use the indexable dictionary structure (lemma 1) over the universe of mini-graphs. Internal to each mini-graph, we build the same structure as `Duplicate components` which captures the micro-graph numbers of duplicates of the same vertex across different micro-graphs. The extra space added by using these structures can be proved to be $o(n)$.

Given a query $q(u, v)$ on two vertices u, v. We state the procedure for vertex u, however the same procedure must be repeated for vertex v afterwards. We first determine if u has duplicates in different mini-graphs or micro-graphs. If it does so, then endpoints of its outgoing edges are explicitly listed which we compare against v by translating mini-graph and/or micro-graph labels of the listed vertices. In case u has duplicates neither across micro-graphs within the mini-graph nor across different mini-graphs, u appears in only one mini-graph and one micro-graph therein. For v to have and edge to u, it must appear in the same micro and mini-graph. We use structure `Duplicate-components` to determine if v has a duplicate in the same micro-graph as u. As otherwise, there cannot be an edge uv. We now use a rank query in `Duplicate-components` to index to `Duplicates` and retrieve the micro-graph label of the proper duplicate of v. Within a micro-graph, we use the look-up table, to determine if they are adjacent in constant time.

Theorem 1. *Any family of separable graphs (definition 1) with entropy $\mathcal{H}(n)$ where n is the number of vertices, can be succinctly encoded in $\mathcal{H}(n) + o(n)$ bits such that adjacency, neighborhood, and degree queries are supported in constant time.* □

4 Representing Planar Maps: Encoding the Combinatorial Embedding

A planar drawing of a planar graph is a drawing of the graph in $\mathbb{R}^2$ with no edge crossings. There is infinitely many planar drawings for any fixed planar graphs G. Two such planar drawings are *equivalent* if for all vertices the clockwise cyclic ordering of neighbors is the same in both graphs. An equivalency class of planar drawings specifies a clockwise cyclic order of neighbors for all vertices which is known as the *combinatorial planar embedding*. A *planar map* is a planar graph together with a fixed combinatorial planar embedding.

In this section, we address the issue of representing (unlabeled) planar maps. The underlying planar graphs of a planar map is separable and therefore the representation of section 3 can encode them succinctly to support adjacency, degree, and neighborhood queries in constant time. In planar maps representations, we not only need to encode the planar graph, but also we need to store the combinatorial planar embedding. Hence, we enhance the definition of neighborhood queries to report neighbors of a fixed vertex according to the combinatorial planar embedding: *i.e.* neighbors should be reported in the clockwise cyclic order in constant time per neighbor.

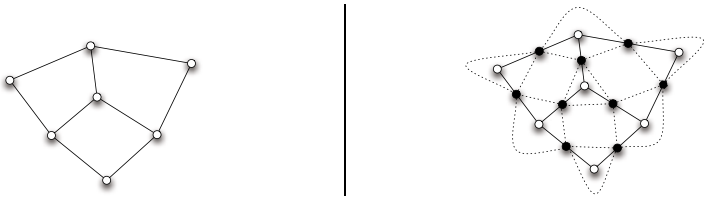

Fig. 4. A planar map (left) and the resulting graph where edges are subdivided and connected according to the combinatorial planar embedding (right)

We first note that we can easily achieve a planar map encoding by increasing the storage requirement by a constant factor. Given a planar map G, we subdivide all edges by introducing a dummy vertex of degree two on each edge and connect these dummy vertices circularly around each vertex (as depicted in figure 4. Since the number of edges of a planar graph is linear, the number of vertices is increased by a constant factor. It is easy to verify that the resulting graph is planar and therefore separable. We can encode this graph using any of the compact planar graph representations referred to in section 1.1 using $O(n)$ bits. Using the dummy vertices, we can produce neighbors of a vertex in the circular order according to the combinatorial embedding. Moreover, we explicitly store a bit for each dummy node which distinguishes the immediate clockwise and counter-clockwise neighbor (*e.g.* we set the bit to zero if the neighbor with a higher label is the clockwise one). Using these bits we can produce the neighbors in the actual clockwise circular order for any fixed node. This encoding proves that the entropy $\mathcal{H}_p(n)$ of planar maps is linear in the number of vertices n.

Although the simple encoding scheme achieves the entropy to within a constant factor, a succinct representation that achieves the entropy tightly to within lower order terms is desired and we will give such representation in this section.

Theorem 2. *A planar map G with n vertices can be encoded succinctly in $\mathcal{H}_p(n) + o(n)$ bits where n is the number of vertices of G. The encoding supports queries adjacency queries, degree queries, and neighborhood queries (according to combinatorial planar embedding of G) in constant time.*

We subdivide edges of G by introducing dummy vertices of degree two on each edge as described before to obtain graph G'. Since G' is planar and separable, we use the succinct separable graph representation of section 3 to represent it. This representation in its current form requires a space which is a constant factor away from entropy $\mathcal{H}_p(n)$. We will make modifications to lessen the space to $\mathcal{H}_p(n) + o(n)$. We will also show constant-time support for queries.

The succinct separable representation of section 3 divides G into mini-graphs and micro-graphs and creates duplicate vertices which are repeated in more than one mini/micro-graphs. Among dummy vertices, we retain all that are duplicates and discard all that are not. A non-duplicate dummy vertex d is discarded by a contraction which deletes the vertex and connects the endpoints of the edge d stood for. We refer to by the resulting graph as $\hat{G}$. By lemmas 3, and, 4 the

total number of dummy vertices that are retained is $o(n)$ and therefore the total number of vertices in the graph is $n + o(n)$. Using a bit vector which is stored as in lemma 2, we explicitly store whether a vertex is a dummy vertex.

Micro-graphs are stored by references into a look-up table as before. The micro-graph is a subgraph of G' and therefore is planar. Furthermore, the combinatorial planar embedding of G' induces a combinatorial planar embedding for micro-graphs. The table stores the combinatorial planar embedding of micro-graphs together with the structure of the graphs. Theorem 1 implies that the storage requirement of the representation is $\mathcal{H}_p(n) + o(n)$ bits.

It only remains to show constant-time support for queries. As the degrees of original vertices in G remain unchanged supporting *degree queries* is trivial. Support for *adjacency queries* is more complicated since we have introduced dummy vertices on edges of G. Nevertheless, the adjacency queries in G are handled in the same manner as adjacency queries in the representation (section 3.2). To show support for adjacency queries in section 3.2, we first oriented the edges of the graph such that each vertex has a bounded out-degree (lemma 5). To orient G', we orient the underlying graph G according to lemma 5 and if edge uv in G has a dummy vertex d in G', we orient edges of G' as $u \to d$ and $d \to v$. We orient edges between dummy vertices according to the clockwise cyclic order. It is easy to verify that all vertices have a constant out-degree in $\hat{G}$, and therefore we can repeat the same procedure as in section 3.2. However, the procedure guarantees that we can discover edges between immediate neighbors and in $\hat{G}$ there could be a dummy vertex on an edge. We first note that this is not an issue within a micro-graph as using the look-up table we can easily test if two vertices are connected through a degree-two vertex (which we must also verify to be a dummy vertex). For vertices that have a duplicate across mini/micro-graphs, we explicitly listed out-neighbors in section 3.2; here we list explicitly out-neighbors through dummy vertices as well (*i.e.* if node u is a duplicate and there are edges $u \to d \to v$ where d is a dummy vertex, we explicitly store v in the list). Response to adjacency queries can be computed as in section 3.2.

We now demonstrate how *neighborhood queries* are supported. Given an edge uv between two vertices u and v of graph G, the neighborhood query is to report the next neighbor of v in the circular order according to the combinatorial planar embedding. Let us denote by d the dummy vertex in G' that resides on edge uv of G. Also we denote by w the next neighbor of u in the circular order in G, and d' the dummy vertex that resides on edge uw in G'. Either of dummy vertices d, d' in G' may or may not be present in $\hat{G}$. Refer to figure 5.

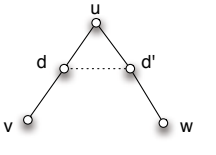

Fig. 5. Supporting neighborhood queries on vertex u: vertex w is reported after vertex v regardless of existence of d or d'

We distinguish two cases according to whether the edge dd' is present or absent in $\hat{G}$. If $dd' \in G'$ (both d and d' are present in $\hat{G}$), then clearly we can discover the next neighbor of u by taking the edge dd' and arriving at vertex d' which leads us to vertex w.

Therefore, the more interesting case is where edge dd' is absent. In this case, neither of d or d' could be a duplicate vertex (as otherwise, since we retain duplicate dummy vertices and their immediate neighbors, they both would be present and therefore edge dd' would exist). Since d and d' do not have duplicates and they are immediately connected (by edge dd'), they belong to the same micro-graph of G'. Moreover, since u, v, w are immediate neighbors of vertices d, d', these vertices or a duplicate of them must also belong to the same micro-graph. Since edges of G are not repeated in more than one micro-graph, the micro-graph is the one containing the (possibly subdivided) edge uv in $\hat{G}$. Hence, the edge uw can be read from the look-up table as the next neighbor of uv in the circular order. □

5 Conclusion and Discussion

We studied the problem of succinctly encoding separable graphs while supporting degree, adjacency, and neighborhood queries in constant time. For each family of separable graphs (*e.g.* planar graphs). The storage is the information-theoretic minimum to within lower order terms. We achieve the entropy bound for any monotone family of separable graphs with no knowledge of the actual entropy for that family of graphs since we use look-up tables for tiny graphs. Namely, when used for planar graphs, our representation requires a space which is the entropy of the planar graphs to within lower order terms while supporting queries in constant time. This is when the actual entropy (or equivalently the number of unlabeled planar graphs) is still unknown [25]. This is an improvement in the heavily-studied compact encoding of planar graphs. Moreover, we showed that our approach yields a succinct representation for planar maps (*i.e.* planar graphs together with a given embedding).

One interesting direction for future work is to extend the idea of this paper to represent dynamic separable graphs. These are graphs under updates in form of insertion and deletion of vertices and edges while the graphs remains separable.

References

1. Gulli, A., Signorini, A.: The indexable web is more than 11.5 billion pages. In: WWW 2005: Special interest tracks and posters of the 14th international conference on World Wide Web, pp. 902–903. ACM, New York (2005)
2. Claude, F., Navarro, G.: A fast and compact web graph representation. In: Ziviani, N., Baeza-Yates, R. (eds.) SPIRE 2007. LNCS, vol. 4726, pp. 118–129. Springer, Heidelberg (2007)
3. Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.: Graph structure in the web. Comput. Netw. 33(1-6), 309–320 (2000)
4. Adler, M., Mitzenmacher, M.: Towards compressing web graphs. In: DCC 2001: Proceedings of the Data Compression Conference, Washington, DC, USA, p. 203. IEEE Computer Society, Los Alamitos (2001)
5. Suel, T., Yuan, J.: Compressing the graph structure of the web. In: DCC 2001: Data Compression Conference, p. 213. IEEE, Los Alamitos (2001)

6. Munro, J.I.: Succinct data structures. Electronic Notes in Theoretical Computer Science 91, 3 (2004)
7. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Mathematics 36(2), 177–189 (1979)
8. Frederickson, G.N.: Fast algorithms for shortest paths in planar graphs, with applications. SIAM J. Comput. 16(6), 1004–1022 (1987)
9. Miller, G.L., Teng, S.H., Thurston, W., Vavasis, S.A.: Separators for sphere-packings and nearest neighbor graphs. J. ACM 44(1), 1–29 (1997)
10. Leighton, T., Rao, S.: An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In: FOCS 1988: Foundations of Computer Science, pp. 422–431. IEEE, Los Alamitos (1988)
11. Blandford, D.K., Blelloch, G.E., Kash, I.A.: Compact representations of separable graphs. In: SODA: ACM-SIAM Symposium on Discrete Algorithms (2003)
12. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Trans. Algorithms 3(4), 43 (2007)
13. Farzan, A., Munro, J.I.: Succinct representations of arbitrary graphs. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 393–404. Springer, Heidelberg (2008)
14. Lu, H.I.: Linear-time compression of bounded-genus graphs into information-theoretically optimal number of bits. In: SODA 2002: Proceedings of ACM-SIAM symposium on Discrete algorithms, pp. 223–224 (2002)
15. Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. SIAM J. Discrete Math. 5(4), 596–603 (1992)
16. Turán, G.: On the succinct representation of graphs. Discrete Applied Mathematics 8, 289–294 (1984)
17. Keeler, W.: Short encodings of planar graphs and maps. DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science 58 (1995)
18. He, X., Kao, M.Y., Lu, H.I.: A fast general methodology for information-theoretically optimal encodings of graphs. SIAM Journal on Computing 30(3), 838–846 (2000)
19. Jacobson, G.: Space-efficient static trees and graphs. In: 30th Annual Symposium on Foundations of Computer Science, 1989, October 30 – November 1, pp. 549–554 (1989)
20. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses, static trees and planar graphs. In: IEEE Symposium on Foundations of Computer Science, pp. 118–126 (1997)
21. Chuang, R.C.N., Garg, A., He, X., Kao, M.Y., Lu, H.I.: Compact encodings of planar graphs via canonical orderings and multiple parentheses. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 118–129. Springer, Heidelberg (1998)
22. Chiang, Y.T., Lin, C.C., Lu, H.I.: Orderly spanning trees with applications to graph encoding and graph drawing. In: SODA 2001: ACM-SIAM symposium on Discrete algorithms, pp. 506–515 (2001)
23. Devillers, L.C.A.O., Schaeffer, G.: Succinct representations of planar maps. Theor. Comput. Sci. 408(2-3), 174–187 (2008)
24. Lipton, R.J., Rose, D.J., Tarjan, R.E.: Generalized nested dissection. SIAM Journal on Numerical Analysis 16, 346–358 (1979)
25. Liskovets, V.A., Walsh, T.R.: Ten steps to counting planar graphs. Congressus Numerantium 60, 269–277 (1987)