Characterizing Planarity by the Splittable Deque

Christopher Auer, Franz J. Brandenburg,
Andreas Gleißner, and Kathrin Hanauer

University of Passau, Germany
{auerc,brandenb,gleissner,hanauer}@fim.uni-passau.de

Abstract. A *graph layout* describes the processing of a graph G by a data structure $\mathcal{D}$, and the graph is called a $\mathcal{D}$-graph. The vertices of G are totally ordered in a *linear layout* and the edges are stored and organized in $\mathcal{D}$. At each vertex, all edges to predecessors in the linear layout are removed and all edges to successors are inserted. There are intriguing relationships between well-known data structures and classes of planar graphs: The stack graphs are the outerplanar graphs [4], the queue graphs are the arched leveled-planar graphs [12], the 2-stack graphs are the subgraphs of planar graphs with a Hamilton cycle [4], and the deque graphs are the subgraphs of planar graphs with a Hamilton path [2]. All of these are proper subclasses of the planar graphs, even for maximal planar graphs.

We introduce *splittable deques* as a data structure to capture planarity. A splittable deque is a deque which can be split into sub-deques. The splittable deque provides a new insight into planarity testing by a game on switching trains. Here, we use it for a linear-time planarity test of a given rotation system.

1 Introduction

In a graph layout, the vertices are processed according to a total order, which is called *linear layout*. The edges correspond to data items that are inserted to and removed from a data structure: Each edge is inserted at the end vertex that occurs first according to the linear layout and is removed at its other end vertex. These operations obey the principles of the underlying data structure, such as "last-in, first-out" for a stack or "first-in, first-out" for a queue. Stack layouts (also known as book embeddings) and queue layouts have been studied extensively, e. g., in [4, 5, 7, 8, 10–12, 16, 18], and are used for 3D drawings of graphs [16], in VLSI design [5] and in other application scenarios [12]. Moreover, Gauss codes and permutation networks of two parallel stacks are characterized by two-stack graphs [14].

Graph layouts are a powerful tool to study planar graphs. A graph G is a $\mathcal{D}$-graph if it has a layout in $\mathcal{D}$. The stack graphs are the outerplanar graphs, and the 2-stack graphs are the subgraphs of planar graphs with a Hamiltonian cycle [4]. Heath et al. [8, 12] have characterized queue graphs as the arched leveled-planar graphs. Such graphs have a planar drawing with vertices placed

S. Wismath and A. Wolff (Eds.): GD 2013, LNCS 8242, pp. 25–36, 2013.

on levels. Inter-level edges connect vertices between two adjacent levels and intra-level edges (the *arches*) connect the left-most vertex to accessible vertices on the right side. In [2], we have characterized the proper leveled-planar graphs, i. e., the arched leveled-planar graphs without arches, as the bipartite queue graphs. Graph layouts can be extended to subdivisions where edges of the graph are replaced by paths. A graph is planar if and only if it has a subdivision that has a layout in two stacks [8].

In [1, 2], we have studied double-ended queue (*deque*) layouts: A deque has two ends, a head and a tail, and items can be inserted and removed at both sides. It can emulate two stacks and additionally allows for queue edges, i. e., edges inserted and removed at opposite sides. In [2], we have shown that the surplus power of a deque in comparison to two stacks captures the difference between Hamiltonian paths and cycles: A graph is a deque (2-stack) graph if and only if it is the subgraph of a planar graph with a Hamiltonian path (cycle). In fact, a planar embedding of a graph with a Hamiltonian path reflects the way the edges are processed in the deque: Fig. 1(c) shows an embedded graph with the Hamiltonian path $1, 2, 3, 4, 11, 12, 13$, which is the linear layout. An edge to the right of the path, e. g., edges e_6 and e_9, is inserted and removed at the tail of the deque whereas the queue edges change sides, e. g., e_3 and e_4 are inserted at the tail and removed at the head. Although more powerful than two stacks, not all planar graphs are deque graphs since there are maximal planar graphs with no Hamiltonian path and the respective decision problem is $\mathcal{NP}$-hard [2]. Yannakakis has shown that four stacks are sufficient and necessary for all planar graphs [17]. However, there are non-planar graphs that have a layout in four and even three stacks. This raises the following question: What is the additional operation a deque must perform to layout exactly the planar graphs? It turns out that the ability to split the deque into pieces is the adequate operation. We show that a graph is planar if and only if it is a *splittable deque (SD)* graph.

Our proof takes an algorithmic viewpoint: We give a linear-time algorithm to test whether or not a rotation system is planar, which uses the SD to process all edges. A rotation system defines the counterclockwise order of edges around each vertex and it is planar if it admits a plane drawing of the graph. In a nutshell, the algorithm is a depth-first search (DFS) which tries to process all edges in the SD according to the rotation system. Planarity follows if this is possible. Otherwise, an edge that cannot be processed, e. g., removed from the deque, causes a crossing. The algorithm is a means to an end for our characterization of planarity and there are other algorithms especially designed for solving the same problem [6]. Nevertheless, our algorithm has the benefit that it operates on an elementary data structure, i. e., a deque, which is very simple in comparison to other ones used for general planarity tests [15]. Note that any two-cell embedding on a surface of genus k can be defined by a rotation system. In particular, it is not sufficient to only test locally at each face whether the rotation system causes crossings. Another challenge are crossings between edges incident to the same vertex, which are ignored in the general case. As the SD exploits the structure of a graph's DFS tree, our characterization is related to the characterization of planarity by de Fraysseix et al. [9].

The SD also provides a playful characterization of planarity: At GD 2012, we presented the poster "Testing Planarity by Switching Trains" [3]. There, the edges are modeled as cars which have to be appended at the head and tail of a train which models the deque. The vertices are train stations which are the sources and destinations of the cars and, at junctions, the train can be split. We also implemented a Java game[1], which uses the time-reversed variant of the SD, i. e., the mergeable deque. The player is asked to switch the cars such that all can be removed at their destination station. The graph underlying a game level is obtained from a GraphML file. If it is possible to bring all cars to their destination without an error, then the underlying graph is planar.

2 Preliminaries

We consider simple, undirected, and connected graphs $G = (V, E)$ with vertices V and edges E such that $|V| \geq 2$. A graph $G = (V, E)$ is planar if it has a plane drawing which maps the vertices to distinct points in the plane and edges $\{u, v\}$ to Jordan arcs from u to v such that Jordan arcs do not cross except at common end points. A *rotation system* $\mathcal{R}_v$ defines a cyclic order of edges around each vertex v. From a plane drawing, we obtain a *planar rotation system* which is the counterclockwise ordering of the edges around each vertex in the drawing. Given a rotation system $\mathcal{R}_v$, each edge e has a successor edge $\mathrm{Succ}_v(e)$ and a predecessor edge $\mathrm{Pre}_v(e)$ at vertex v.

A *DFS tree* $\mathcal{T} = (V, E_\mathcal{T})$ is a rooted, directed spanning tree of G obtained from a DFS traversal starting at a root vertex r. We assume that the *tree edges* $E_\mathcal{T}$ are directed from the parent to its children. We denote by $u \to v$ that $(u, v) \in E_\mathcal{T}$. By $u \xrightarrow{+} v$, we denote a path of tree edges (at least one) from u to v. Vertex u is an *ancestor* of v and v is a *descendant* of u. By $u \xrightarrow{*} v$, we denote $u = v$ or $u \xrightarrow{+} v$. $\mathcal{T}$ partitions E into tree edges $E_\mathcal{T}$ and *forward edges* F. For each forward edge $\{u, v\} \in F$, there is a path $u \xrightarrow{+} v$ where u is an ancestor of v.

A *linear layout* if a total ordering $\prec$ of the vertices. If $u \prec v$, then u is called *predecessor* of v and v *successor* of u. In a graph layout, a vertex v can be seen as a processing unit that receives as *input* a data structure from which v's edges to predecessors are removed and edges to successors are inserted. Insertions and removals obey the modus operandi of the data structure. The resulting data structure is the *output* of the vertex and the input to the immediate successor. The input to the first and output of the last vertex is empty.

A *deque* is a doubly linked list whose content is denoted by $\mathcal{C} = (e_1, \ldots, e_k)$, where e_1 is at the *head* `h` and e_k is at the *tail* `t`. The empty deque is denoted by $()$. We denote by $e_i \in \mathcal{C}$ that e_i is in $\mathcal{C}$ and by $e_i \ll_\mathcal{C} e_j$ that $i < j$ in $\mathcal{C}$. We omit the subscript in $\ll_\mathcal{C}$ if it is clear from the context which configuration is meant.

[1] `http://www.infosun.fim.uni-passau.de/br/games/derail.jar`

An edge $e \in \mathcal{C}$ can be removed if it is situated at the head or the tail of $\mathcal{C}$. In the following, we denote by $\mathcal{C}_v$ the input of vertex v.

Let $G = (V, E)$ be a graph endowed with a rotation system and assume that G contains Hamiltonian path $p = (v_1, \ldots, v_n)$. Path p is a (degenerate) DFS tree of G and it induces a linear layout with $v_i \prec v_j$ if and only if $i < j$ for all $1 \leq i, j \leq n$. The edges on p are directed from each vertex to its immediate successor. A rotation system of G defines the order in which the edges are inserted to and removed from a deque and Algorithm 1 shows how: It takes as input a vertex v and its rotation system $\mathcal{R}_v$ along with a deque $\mathcal{C}_v$. e_p and e_s are the edges to the immediate predecessor and successor of v on p, respectively. The value $\perp$ indicates that v is the first or last vertex. Let v_i with $1 < i < n$ be an inner vertex of p. Its rotation system is sketched in Fig. 1(a). In Algorithm 1, all edges $e_1^{\mathtt{h}}, \ldots, e_k^{\mathtt{h}}$ between e_p and e_s are inserted and removed at the head in counterclockwise order of the rotation system (lines 1–4). An edge is removed if it points to a predecessor and it is inserted if it points to a successor. We say that these edges are to the *left* of p at v. Then, at v, all edges $e_1^{\mathtt{t}}, \ldots, e_l^{\mathtt{t}}$ to the *right* of p between e_p and e_s are processed at the deque's tail in reversed, i. e., clockwise, order of the rotation system (lines 5–10). Whereas an edge can always be inserted, removing might not be possible if the edge is not accessible at the corresponding side of the deque. In this case, Algorithm 1 aborts and returns $\perp$. At the endpoints v_1 and v_n of p, the rotation system can be divided at an arbitrary position. In this case, Algorithm 1 processes all edges at the head. Note that the edges of the Hamiltonian path p are not processed in the deque. The reason is that these edges can always be processed canonically without interfering with any other edges: First of all, the edge to the immediate predecessor is removed at the head and, last of all, the edge to the immediate successor is inserted at the head. Hence, we can safely ignore all edges on p. We say that a rotation system *admits* a deque layout if subsequently calling `ProcessDeque` for all vertices $v_1, \ldots, v_n$ in order never returns $\perp$ and the input to v_1 is the empty deque and so is the output of v_n. From [2], we obtain the following proposition.

Proposition 1. *The rotation system of a graph with a Hamiltonian path is planar if and only if it admits a deque layout.*

For an example, consider Fig. 1(c). The input of vertex 3 is the deque with content (e_4, e_3) and edge e_6 is inserted to the tail of the deque, which results in (e_4, e_3, e_6). Consider edge e' which crosses e_6. This is also reflected in the deque layout: At vertex 4, e_4 is removed at the deque's head and e' inserted to the tail resulting in (e_3, e_6, e'). At vertex 11, e_6 must be removed at the tail which is not possible since e' is in its way.

3 The Splittable Deque

The SD enhances the deque by allowing it to be split. This is the dual to the mergeable deque of Kosaraju [13]. In order to define SD layouts, we generalize

Algorithm 1. `ProcessDeque`

Input: vertex v with rotation system $\mathcal{R}_v = (e_1, \ldots, e_k)$, deque $\mathcal{C}_v$, edges e_p (e_s) to immediate predecessor (successor); or $\perp$ if v is first (last)
Output: new content of the deque or $\perp$ if deque layout is not possible

```
1  e ← Succ_v(e_p) if e_p ≠ ⊥, else Succ_v(e_s)
2  while e ≠ e_s ∧ e ≠ e_p do
3      if e is edge to successor then C_v.insertAtHead(e)
4      else C_v.removeAtHead(e) or return ⊥ if not possible e ← Succ_v(e)
5  if e_s ≠ ⊥ ∧ e_p ≠ ⊥ then v is an inner vertex
6      e ← Pre_v(e_p)
7      while e ≠ e_s do
8          if e is edge to successor then C_v.insertAtTail(e)
9          else C_v.removeAtTail(e) or return ⊥ if not possible
10         e ← Pre_v(e)
11 return C_v
```

linear layouts to tree layouts. A *tree layout* is an ordered DFS tree $\mathcal{T} = (V, E_\mathcal{T})$ of a graph G, i. e., a DFS tree in which the children of each inner vertex are totally ordered from left to right. Remember that the linear layout of a deque layout describes the processing pipeline, i. e., if u is the immediate predecessor of v, then the output of u is the input of v. With a tree layout $\mathcal{T}$, a vertex v can have multiple immediate successors, namely, its children $w_1, w_2, \ldots, w_l$ in order. For a graph G, let $\mathcal{T} = (V, E_\mathcal{T})$ be a tree layout with root r. The input $\mathcal{C}_r = ()$ of root r is empty. Let $\mathcal{C}_v = (e_1, \ldots, e_k)$ be the input of vertex $v \in V$ and let $w_1, \ldots, w_l$ be the children of v in $\mathcal{T}$ in order. Assume for now that v has at least one child. At first, $\mathcal{C}_v$ is split into $l \leq 1$ consecutive and disjoint pieces $c_{w_1}, \ldots, c_{w_l}$, where $\mathcal{C}_v$ is the concatenation of $c_{w_1}, \ldots, c_{w_l}$. These l pieces constitute l new SDs. Each forward edge from an ancestor of v in $\mathcal{T}$ has to be removed and each forward edge to a descendant has to be inserted at one of these SDs. The SD obtained from c_{w_i} after all removals and insertions is the input $\mathcal{C}_{w_i}$ of child w_i. If v is a leaf, its output must be empty. If $\mathcal{T}$ is a path, the SD is never split and behaves like a deque. A graph is an *SD graph* if it has a tree layout such that all edges can be processed in the SD.

For an example, consider the graph in Fig. 1(b) whose rotation system can be obtained from the drawing. The vertices are numbered according to a DFS run starting at root 1. The red, dashed edges are ignored for the moment. All tree edges are directed from parent to children and drawn bold. In Fig. 1(d), the tree layout as defined by the DFS tree is displayed where the children are ordered from left to right according to the rotation system. In fact, the rotation system as shown in Fig. 1(d) is equal to the one obtained from Fig. 1(b). In the example, the input SD of vertex 4 is $\mathcal{C}_4 = (e_5, e_2, e_1, e_4, e_3, e_6)$. At vertex 4, the SD is split into three pieces $c_5 = (e_5, e_2)$, $c_{10} = (e_1)$, and $c_{11} = (e_4, e_3, e_6)$. Afterwards, forward edge e_4 is removed at the head of c_{11}. We obtain $\mathcal{C}_5 = (e_5, e_2)$, $\mathcal{C}_{10} = (e_1)$, and $\mathcal{C}_{11} = (e_3, e_6)$ as inputs to vertices 5, 10, and 11, respectively. In principle, $\mathcal{C}_4$ can

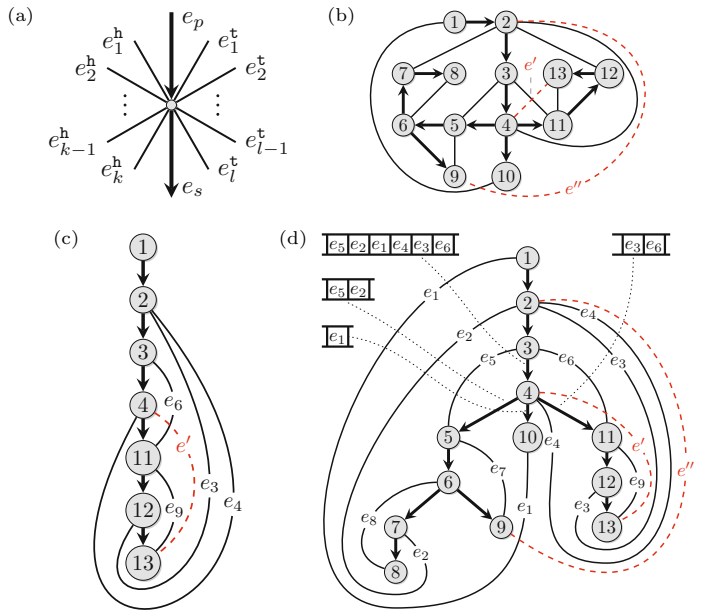

Fig. 1. (a) shows how a path splits the rotation system of a vertex and at which side of the deque the edges are processed. A planar graph with its DFS tree (directed and bold drawn edges) is shown in (b) and its tree layout in (d) along with the input SD for some vertices. (c) shows a path from the root to a leaf in the DFS tree, which corresponds to a deque layout.

also be split such that $c_{10} = (e_1, e_4)$ and $c_{11} = (e_3, e_6)$ and then e_4 is removed at the tail of c_{10}. This ambiguity occurs at vertices with more than one child and does not influence the property of a graph of being an SD graph.

4 Testing Planarity of a Rotation System by the SD

In this section, we prove our main result.

Theorem 1. *A graph G is an SD graph if and only if G is planar.*

To prove this theorem, we use Algorithm 2 which uses the SD to test whether a given rotation system is planar. By showing its correctness, Theorem 1 follows. Algorithm 2 defines the recursive routine `PlanarRS` which takes as input a vertex v endowed with a rotation system, a tree layout $\mathcal{T}$, the input SD $\mathcal{C}_v$ for v, and the tree edge e_a from its parent p in $\mathcal{T}$ or $\perp$ if v is the root. The tree layout $\mathcal{T}$ is obtained from a prior DFS run, where the children of each vertex v are ordered from left to right as given by the rotation system. In a nutshell, `PlanarRS` first splits $\mathcal{C}_v$ into pieces $c_{w_1}, \ldots, c_{w_l}$, one for each of v's children $w_1, \ldots, w_l$, and then removes and inserts all forward edges according to the rotation system of

Algorithm 2. `PlanarRS`

Input: vertex v with rotation system $\mathcal{R}_v = (e_1, \ldots, e_k)$, tree layout $\mathcal{T} = (V, E_\mathcal{T})$, SD $\mathcal{C}_v$, tree edge e_p from parent p or $\perp$ if v is the root

Output: `true` $\mathcal{R}$ is planar; `false` if one of $e_1, \ldots, e_k$ causes a crossing

1 **if** v *is leaf* **then return** *whether* `ProcessDeque`$(v, \mathcal{R}_v, \mathcal{C}_v, e_p, \perp) = ()$
2 $w_1, \ldots, w_l \leftarrow$ children of v in $E_\mathcal{T}$
3 Split $\mathcal{C}_v$ into pieces $c_{w_1}, \ldots, c_{w_l}$ such that $\forall e \in c_{w_i} : e$ is removed at v, w_i or one of w_i's descendents or return `false` if this is not possible
4 **foreach** $w_i \in \{w_1, \ldots, w_l\}$ **do** $\tilde{\mathcal{R}}_i \leftarrow []$; removed$[w_i] \leftarrow$ `false` $\mathcal{S} \leftarrow$ empty stack
5 $e_1, \ldots, e_k \leftarrow$ rotation system of v with $e_1 = e_p$ if v is not root; else $e_1 = (v, w_1)$
foreach $e = e_1, \ldots, e_k$ **do**
6 $\quad w_i \leftarrow$ child w_i with $e \in c_{w_i}$, e is removed at descendant of w_i, or $e = (v, w_i)$
7 $\quad$ **if** e *is forward edge* **then** $\tilde{\mathcal{R}}_i$.`append`(e) **if** $\mathcal{S}$.`top`$() = w_i$ **then continue** with next edge in line 5 **if** $w_i \in \mathcal{S}$ **then**
8 $\quad\quad$ **while** $\mathcal{S}$.`top`$() \neq w_i$ **do**
9 $\quad\quad\quad w_{i'} \leftarrow \mathcal{S}$.`pop`$()$; removed$[w_{i'}] \leftarrow$ `true`
10 $\quad\quad\quad \mathcal{C}_{w_{i'}} \leftarrow$ `ProcessDeque`$(v, \tilde{\mathcal{R}}_{i'}, c_{w_{i'}}, e_p, (v, w_{i'}))$
11 $\quad\quad\quad$ **if** $\mathcal{C}_{w_{i'}} = \perp$ **then return** `false` **else if** $\neg$`PlanarRS`$(w_{i'}, \mathcal{R}_{w_{i'}}, \mathcal{T}, \mathcal{C}_{w_{i'}}, (v, w_{i'}))$ **then return** `false`
12 $\quad$ **else if** $\neg$removed$[w_i]$ **then** $\mathcal{S}$.`push`(w_i) **else return** `false`
13 **while** $\neg\mathcal{S}$.`isEmpty`$()$ **do**
14 $\quad w_i \leftarrow \mathcal{S}$.`pop`$()$
15 $\quad \mathcal{C}_{w_i} \leftarrow$ `ProcessDeque`$(v, \tilde{\mathcal{R}}_i, c_{w_i}, e_p, (v, w_i))$
16 $\quad$ **if** $\mathcal{C}_{w_i} = \perp$ **then return** `false` **else if** $\neg$`PlanarRS`$(w_i, \mathcal{R}_{w_i}, \mathcal{T}, \mathcal{C}_{w_i}, (v, w_i))$ **then return** `false`
17 **return** `true`

v. Afterwards, it recursively calls `PlanarRS` for all its children. If at some point, the SD cannot be split adequately or an edge cannot be removed, `false` is returned and propagated back to the initial caller of `PlanarRS`. Otherwise, `true` is returned. In the following, we assume that in a drawing of G which respects the rotation system no pair of edges crosses more than once and no edge crosses any of the tree edges: As the tree layout $\mathcal{T}$ itself contains no cycles, it is always planar regardless of the rotation system. Also, all forward edges can be drawn such that they cause no crossing with any tree edge. This is also reflected in the SD where all tree edges can be processed canonically without interfering with any forward edge: After splitting the SD, the tree edge from the parent can be removed at the head of the first SD and, as the last step, each tree edge to child w_i can be inserted at the head of the SD for child w_i. As with the deque, we only consider forward edges in the SD layout.

To actually insert and remove forward edges to an SD, `PlanarRS` uses the routine `ProcessDeque`. The observation behind is that a deque is a special case of the SD, namely, whenever the tree layout is a path: Let G be a graph endowed

with a rotation system and $\mathcal{T}$ be a tree layout of G. Further, let p be a path from the root to a leaf of $\mathcal{T}$ and denote by G_p the subgraph of G induced by p which inherits G's rotation system. For instance, the subgraph G_p for the root-to-leaf path $p = 1 \xrightarrow{+} 13$ in Fig. 1(d) is shown in Fig. 1(c). G_p's rotation system is planar if and only if it admits a deque layout by Proposition 1. Hence, if G's rotation system is planar, then the rotation system on every root-to-leaf path admits a deque layout. In `PlanarRS`, all edges that lie on a common root-to-leaf path p are processed in the SD just like in a deque. If this is possible, then the rotation system of each root-to-leaf path is planar. This is already reflected in line 1: If v is a leaf, then the SD is not split and must be emptied by `ProcessDeque` and `true` is returned if and only if `ProcessDeque` returns an empty SD.

In line 3, the SD $\mathcal{C}_v$ is split into pieces $c_{w_1}, \ldots, c_{w_l}$ such that for each edge $e \in c_{w_i}$ edge e is removed at v, w_i or one of w_i's descendants. If this is not possible, `false` is returned. Consider edge e'' in Figs. 1(b) and (d), which crosses edge e_1. Edge e_1 is inserted at the head at vertex 1 and e'' at the tail at vertex 2, i. e., $e_1 \ll e''$. At vertex 4, the deque has to be split such that $e_1 \in c_{10}$ and $e'' \in c_5$. However, as $e_1 \ll e''$ and vertex 5 is left of vertex 10, this is not possible. In general, we obtain the following lemma:

Lemma 1. *Let e and e' be two forward edges in $\mathcal{C}_v$ such that e (e') is removed at w_i ($w_{i'}$) or one of its descendants. It is possible to split $\mathcal{C}_v$ such that $e \in c_{w_i}$ and $e' \in c_{w_{i'}}$ if and only if e and e' do not cross.*

Proof. We assume that e and e' are inserted at u and u' and removed at x and x', respectively. Since e and e' are forward edges, there are paths $p = u \xrightarrow{+} v \to w_i \xrightarrow{*} x$ and $p' = u' \xrightarrow{+} v \to w_{i'} \xrightarrow{*} x'$.

$\Leftarrow$: We prove the contrapositive and assume that $\mathcal{C}_v$ cannot be split such that $e \in c_{w_i}$ and $e' \in c_{w_{i'}}$. Either e and e' are inserted to the same side of the deque or to different sides. For the first case, assume that e and e' are both inserted at the head and, w. l. o. g., w_i is left of $w_{i'}$ in the total order of children at v. This implies that $e' \ll_{\mathcal{C}_v} e$. This situation is depicted in Fig. 2(a). Remember that `PlanarRS` inserts and removes edges to the SD as with a deque. Hence, both edges e and e' are to the left of p and p' at u and u', respectively. There is a cycle formed by path p and forward edge e. In a drawing of G which respects the rotation system, this circle encloses a region R (dark shaded in Fig. 2(a)) such that R does not contain $w_{i'}$. As $e' \ll_{\mathcal{C}_v} e$, e is inserted before e' and, thus, either u is an ancestor of u', or $u = u'$ and the rotation system at u is $\mathcal{R}_u = (\ldots, e, \ldots, e', \ldots, e_d, \ldots)$, where e_d is the tree edge from u to u's child on p. In either case, the edge curve of e' starts within region R and must end outside R which inevitably causes a crossing with e. The reasoning if e and e' are inserted at the tail is analogous.

In the second case, e and e' are inserted at different sides where, w. l. o. g., e is inserted at the tail and e' at the head (cf. Fig. 2(b)). Hence, $e' \ll_{\mathcal{C}_v} e$. Since $\mathcal{C}_v$ cannot be split adequately, this implies that w_i must be left of $w_{i'}$ at v. Again, let R be the region enclosed by p and e (dark shaded in Fig. 2(b)) such that R contains $w_{i'}$. As e is to the right of p at u and e' to the left of p' at u', the

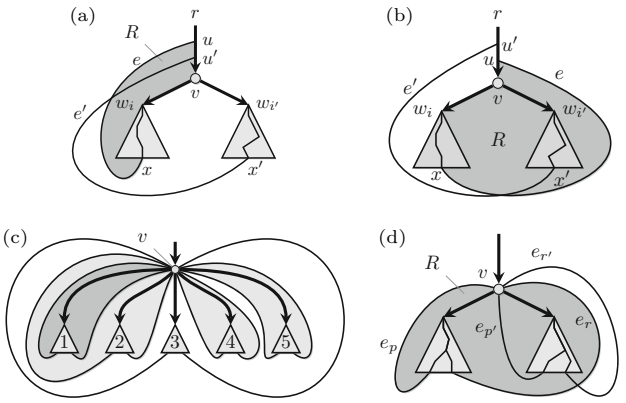

Fig. 2. The two cases where the SD cannot be split appropriately (Figs. (a) and (b)). Fig. (c) sketches nesting sectors and Fig. (d) clashing sectors.

edge curve of e' starts outside of R and must reach x' within R which leads to a crossing with e.

$\Rightarrow$: Again, we prove the contrapositive. We have two cases: Either e and e' lie on the same side of p and p' at u and u', respectively, or on different sides. For the first case, we assume w. l. o. g. that w_i is left of $w_{i'}$ at v. If a crossing between e and e' is unavoidable, then u is either an ancestor of u', or $u = u'$ and rotation system of u is $\mathcal{R}_u = (\ldots, e, \ldots, e', \ldots, e_d, \ldots)$, where e_d is the tree edge from u to u's child on p (Fig. 2(a)). In either case, e is inserted at the head before e' and $e' \ll_{\mathcal{C}_v} e$. Thus, $\mathcal{C}_v$ cannot be split such that $e \in c_{w_i}$ and $e' \in c_{w_{i'}}$. For the second case, e lies to the right of p and e' to the left of p' (w. l. o. g.). Thus, e is inserted at the tail and e' at the head and $e' \ll_{\mathcal{C}_v} e$. If a crossing between e and e' is unavoidable, then w_i is to the left of $w_{i'}$ at v (cf. Fig. 2(b)) and, again, $\mathcal{C}_v$ cannot be split such that $e \in c_{w_i}$ and $e' \in c_{w_{i'}}$. □

Let $\mathcal{R}_v = (e_1, \ldots, e_k)$ be the rotation system of v such that e_1 is the tree edge from v's parent if v is an inner vertex of $\mathcal{T}$. If v is the root, then e_1 is the tree edge to v's first child w_1. After $\mathcal{C}_v$ is successfully split into pieces $c_{w_1}, \ldots, c_{w_l}$, `PlanarRS` finds for each piece a subsequence of $\mathcal{R}_v$ which contains all forward edges that must be removed from and inserted to c_{w_i}. A *sector* $\tilde{\mathcal{R}}_i$ of child w_i is a subsequence of v's rotation system such that all edges in $\tilde{\mathcal{R}}_i$ are forward edges incident to an ancestor of v or a descendant of w_i. Further, for each forward edge e there is exactly one sector $\tilde{\mathcal{R}}_i$ with $e \in \tilde{\mathcal{R}}_i$. If the rotation system is planar, then all sectors must properly nest: Consider Fig. 2(c) in which v has five children corresponding to five subtrees. In the rotation system of v, the sector that corresponds to subtree 2 encloses the sector corresponding to subtree 1, and the sector belonging to 3 encloses both. Let $\tilde{\mathcal{R}}_i = (e_p, \ldots, e_q)$ and $\tilde{\mathcal{R}}_{i'} = (e_{p'}, \ldots, e_{q'})$ be two sectors of $\mathcal{R}_v = (e_1, \ldots, e_k)$. We say that $\tilde{\mathcal{R}}_i$ and $\tilde{\mathcal{R}}_{i'}$ *clash* if there exist edges $e_r \in \tilde{\mathcal{R}}_i$ and $e_{r'} \in \tilde{\mathcal{R}}_{i'}$ with $p < p' < r < r'$ (see Fig. 2(d)) or $r < r' < q < q'$.

Lemma 2. *In a planar rotation system, no pair of sectors clash.*

Proof. Assume for contradiction that G's rotation system is planar but the sectors $\tilde{\mathcal{R}}_i = (e_p, \dots, e_q)$ and $\tilde{\mathcal{R}}_{i'} = (e_{p'}, \dots, e_{q'})$ clash. Let $e_r \in \tilde{\mathcal{R}}_i$ and $e_{r'} \in \tilde{\mathcal{R}}_{i'}$ be edges with $p < p' < r < r'$. The reasoning for the case $r < r' < q < q'$ is similar. The situation is shown in Fig. 2(d). There is a circle formed by v, e_p, e_r and a simple path between the endpoints of e_p and e_r distinct from v. In a plane drawing of G respecting the rotation system, this circle encloses a region R (shaded in Fig. 2(d)) such that it contains the endpoints of $e_{p'}$ and $e_{r'}$ other than v. As $p < p' < r < r'$, the edge curve of $e_{r'}$ starts outside of R and ends inside which leads to a crossing; a contradiction. □

Corollary 1. *In a planar rotation system, all pairs of sectors $\tilde{\mathcal{R}}_i = (e_p, \dots, e_q)$, $\tilde{\mathcal{R}}_{i'} = (e_{p'}, \dots, e_{q'})$ with $p < p'$, are either disjoint, i. e., $p \leq q < p' \leq q'$, or nesting, i. e., $p < p' < q' < q$.*

If all sectors are disjoint or nesting, we can construct a plane drawing at vertex v given plane drawings of all subgraphs that belong to the subtrees of v's children. To test if the sectors are nesting, `PlanarRS` uses a stack in which v's children $w_1, \dots, w_l$ are inserted and removed. Further, for each child w_i, it maintains the boolean variable removed$[w_i]$ which stores if w_i has been removed from the stack. `PlanarRS` subsequently processes all edges e of v in order of the rotation system (line 5). In line 6, the child w_i "responsible" for e, is determined, i. e., either e is the tree edge from v to w_i, or e is a forward edge and must be removed from c_{w_i} or e must be inserted at v and is removed at a descendant of w_i. If e is a forward edge, it is appended to sector $\tilde{\mathcal{R}}_i$. If w_i is currently on top of the stack, no further action is needed (line 7). If $w_i \in \mathcal{S}$, all children $w_{i'}$ on the stack are removed until w_i is on top. For all removed children $w_{i'}$, removed$[w_{i'}]$ is set to `true`. If $w_i \notin \mathcal{S}$ and removed$[w_i]$ = `false`, w_i is pushed onto the stack (line 12).

If w_i is not in the stack and has previously been removed, `false` is returned (line 12) as the rotation system is not planar for the following reasons: Child w_i has been removed in a previous iteration when another child $w_{i'}$ further below in the stack needed to be on top. Let $\tilde{\mathcal{R}}_i = (e_p, \dots, e_q)$ be the sector of w_i and $\tilde{\mathcal{R}}_{i'} = (e_{p'}, \dots, e_{q'})$ the sector of $w_{i'}$. $w_{i'}$ has been inserted to $\mathcal{S}$ before w_i and, thus, $p' < p$. In the iteration when w_i is removed from $\mathcal{S}$, the edge of the iteration is $e_{r'} \in \mathcal{R}_{w_{i'}}$. Further, there is an edge $e_r \in \mathcal{R}_{w_i}$ when w_i would be reinserted with $r' < r$. Altogether we get $p' < p < r' < r$ and, hence, $\tilde{\mathcal{R}}_i$ and $\tilde{\mathcal{R}}_{i'}$ clash. Thus, the rotation system of G is not planar by Lemma 2.

Whenever a child $w_{i'}$ is removed from $\mathcal{S}$, it must never be inserted again and, hence, $\tilde{\mathcal{R}}_{i'}$ must contain all edges to be processed in $c_{w_{i'}}$. In line 10, `ProcessDeque` is called with sector $\tilde{\mathcal{R}}_{i'}$ and $c_{w_{i'}}$ as parameters. Remember that `ProcessDeque` needs two edges on a path which divide the rotation system into a left and right half. Here, these two edges are the tree edge from the parent of v, if existent, and the tree edge from v to $w_{i'}$. If the return value $\mathcal{C}_{w_{i'}}$ of `ProcessDeque` is $\perp$, not all edges could be processed in the deque and the rotation system is not planar by Proposition 1. Thus, `false` is returned in line 11.

Otherwise, `PlanarRS` is called recursively on $w_{i'}$ with input deque $\mathcal{C}_{w_{i'}}$ (line 11). After all forward edges of the rotation system are processed, all children remaining in $\mathcal{S}$ are removed (lines 13–16) and `ProcessDeque` and `PlanarRS` are called. If all calls of `PlanarRS` return `true`, the rotation system of each root-to-leaf path is planar and so are the rotation systems at each vertex with more than one child.

Lemma 3. `PlanarRS` *in Algorithm 2 returns* `true` *if and only if the rotation system of its input graph is planar.*

Since `PlanarRS` obeys the SD's modus operandi, Theorem 1 follows. Each edge is inserted and removed exactly once. Further, in the loop from lines 5–12 each forward edge is processed at most twice during the algorithm and, by using the DFS numbers, it is possible to decide in time $\mathcal{O}(1)$ in which subtree the end vertex of a forward edge lies. Also, each vertex is inserted at most twice to the stack for each edge. The operation that needs more arguing is splitting the deque (line 3) for which also a linear running time can be achieved.

Whenever `PlanarRS` returns `false`, the edges that cause a crossing can be determined: If one of the calls of `ProcessDeque` returns $\perp$, then an edge could not be removed from the SD since at least one other edge is blocking its way. Hence, these edges must cross. If the SD cannot be split adequately (line 3), then we obtain one of the situations as in Figs. 2(a) or (b) and the edges which prevent the SD from being split adequately are those that cause a crossing. Last, if some of v's children w_i were reinserted into the stack (lines 12 and 16), then the corresponding sectors would be classhing and, hence, there exist two edges that cross according to the proof of Lemma 2 (see Fig. 2(d)).

5 Conclusion

We characterized planarity by graph layouts in the splittable deque (SD): Although a stack, two stacks, or the deque characterize large classes of planar graphs, they do not capture all. We enhanced the deque by a split-operation and showed that it characterizes planarity. For our proof, we devised a linear-time algorithm operating on the SD to test the planarity of a rotation system. If it is not planar, the operations on the SD indicate crossing edges. Our test also works for graphs with multi-edges. Given a rotation system, it defines the order of order of insertions and removals to the SD. Conversely, given the order of insertions and removals to the SD, we can find a (planar) rotation system. So a planarity testing algorithm can use the SD to find an embedding of a graph.

References

1. Auer, C., Bachmaier, C., Brandenburg, F.J., Brunner, W., Gleißner, A.: Plane drawings of queue and deque graphs. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 68–79. Springer, Heidelberg (2011)

2. Auer, C., Gleißner, A.: Characterizations of deque and queue graphs. In: Kolman, P., Kratochvíl, J. (eds.) WG 2011. LNCS, vol. 6986, pp. 35–46. Springer, Heidelberg (2011)
3. Auer, C., Gleißner, A., Hanauer, K., Vetter, S.: Testing planarity by switching trains. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 557–558. Springer, Heidelberg (2013)
4. Bernhart, F., Kainen, P.: The book thickness of a graph. J. Combin. Theory, Ser. B 27(3), 320–331 (1979)
5. Chung, F.R.K., Leighton, F.T., Rosenberg, A.L.: Embedding graphs in books: A layout problem with applications to VLSI design. SIAM J. Algebra. Discr. Meth. 8(1), 33–58 (1987)
6. Donafee, A., Maple, C.: Planarity testing for graphs represented by a rotation scheme. In: Banissi, E., Börner, K., Chen, C., Clapworthy, G., Maple, C., Lobben, A., Moore, C.J., Roberts, J.C., Ursyn, A., Zhang, J. (eds.) Proc. Seventh International Conference on Information Visualization, IV 2003, pp. 491–497. IEEE Computer Society, Washington, DC (2003)
7. Dujmović, V., Wood, D.R.: On linear layouts of graphs. Discrete Math. Theor. Comput. Sci. 6(2), 339–358 (2004)
8. Dujmović, V., Wood, D.R.: Stacks, queues and tracks: Layouts of graph subdivisions. Discrete Math. Theor. Comput. Sci. 7(1), 155–202 (2005)
9. de Fraysseix, H., Rosenstiehl, P.: A depth-first-search characterization of planarity. In: Graph Theory, Cambridge (1981); Ann. Discrete Math., vol. 13, pp. 75–80. North-Holland, Amsterdam (1982)
10. Heath, L.S., Leighton, F.T., Rosenberg, A.L.: Comparing queues and stacks as mechanisms for laying out graphs. SIAM J. Discret. Math. 5(3), 398–412 (1992)
11. Heath, L.S., Pemmaraju, S.V., Trenk, A.N.: Stack and queue layouts of directed acyclic graphs: Part I. SIAM J. Comput. 28(4), 1510–1539 (1999)
12. Heath, L.S., Rosenberg, A.L.: Laying out graphs using queues. SIAM J. Comput. 21(5), 927–958 (1992)
13. Kosaraju, S.R.: Real-time simulation of concatenable double-ended queues by double-ended queues (preliminary version). In: Proc. 11th Annual ACM Symposium on Theory of Computing, STOC 1979, pp. 346–351. ACM, New York (1979)
14. Rosenstiehl, P., Tarjan, R.E.: Gauss codes, planar hamiltonian graphs, and stack-sortable permutations. J. of Algorithms 5, 375–390 (1984)
15. Shih, W.K., Hsu, W.L.: A new planarity test. Theor. Comput. Sci. 223(1-2), 179–191 (1999)
16. Wood, D.R.: Queue layouts, tree-width, and three-dimensional graph drawing. In: Agrawal, M., Seth, A.K. (eds.) FSTTCS 2002. LNCS, vol. 2556, pp. 348–359. Springer, Heidelberg (2002)
17. Yannakakis, M.: Four pages are necessary and sufficient for planar graphs. In: Proc. of the 18th Annual ACM Symposium on Theory of Computing, STOC 1986, pp. 104–108. ACM, New York (1986)
18. Yannakakis, M.: Embedding planar graphs in four pages. J. Comput. Syst. Sci. 38(1), 36–67 (1989)