Leftist Canonical Ordering

Melanie Badent[1], Michael Baur[2], Ulrik Brandes[1], and Sabine Cornelsen[1]

[1] Department of Computer & Information Science, University of Konstanz
{Melanie.Badent,Ulrik.Brandes,Sabine.Cornelsen}@uni-konstanz.de
[2] Department of Computer Science, Universität Karlsruhe (TH)
baur@informatik.uni-karlsruhe.de

Abstract. Canonical ordering is an important tool in planar graph drawing and other applications. Although a linear-time algorithm to determine canonical orderings has been known for a while, it is rather complicated to understand and implement, and the output is not uniquely determined. We present a new approach that is simpler and more intuitive, and that computes a newly defined leftist canonical ordering of a triconnected graph which is a uniquely determined leftmost canonical ordering.

1 Introduction

Canonical vertex orderings were introduced by de Fraysseix, Pach, and Pollack [13, 14] and are the backbone of several algorithms for planar graphs, including graph drawing algorithms [28, 27, 29, 30, 2–4, 8–10, 16–20, 22, 23], graph encoding [11, 26, 1], construction of realizers, spanners, or orderly spanning trees [5–7, 15, 31–33], and more [34, 12, 25].

Kant [28] generalized canonical orderings to triconnected graphs. While several implementations of the linear-time algorithm of Kant are available, this algorithm is neither trivial to code, nor is its correctness easily understood. Based on a simple and intuitive criterion, we present a new algorithm that might further broaden the scope of adoption and ease teaching.

Since a triconnected graph can have many canonical orderings, we introduce the leftist (and rightist) canonical ordering that is uniquely determined. The leftist canonical ordering is in particular a leftmost canonical ordering.

The main advantage of our algorithm compared to the algorithm in [28] is that we do not use the dual graph nor any face labels. Further, we compute the unique leftist canonical ordering from scratch, i. e., without any reordering, and we compute it from the low numbers to the high numbers contrary to the previous algorithm that builds the canonical ordering from the end by shelling off paths from the outer face. A similar approach for biconnected canonical orderings can be found in [24]. We also give a detailed pseudocode such that it can be easily implemented. Finally, our proof of correctness includes a new proof of the existence of a canonical ordering for triconnected graphs.

The paper is organized as follows. Canonical orderings are defined in Sect. 2. The new algorithm and its linear-time implementation are described in Sects. 3 and 4, respectively.

2 Preliminaries

Throughout this paper, let $G = (V, E)$ be a simple undirected graph with n vertices, $n \geq 3$, and m edges. We assume that G is planar and triconnected, hence it has a unique planar embedding up to the choice of the outer face. For a subset $V' \subseteq V$ we denote by $G[V']$ the subgraph of G induced by V'. By $\deg_G(v)$ we denote the number of edges of G that contain v. A path is a sequence $P = \langle v_0, \ldots, v_\ell \rangle$ of distinct adjacent vertices, i. e., $\{v_i, v_{i+1}\} \in E$. We also denote the set $\{v_0, \ldots, v_\ell\}$ by P.

Canonical orderings were introduced originally for triangulated graphs by de Fraysseix et al. [13, 14]. The following rephrases Kant's generalization to triconnected graphs [28].

Definition 1 (canonical ordering). *Let $\Pi = (P_0, \ldots, P_r)$ be a partition of V into paths and let $P_0 = \langle v_1, v_2 \rangle$, $P_r = \langle v_n \rangle$ such that v_2, v_1, v_n is a path on the outer face of G in clockwise direction. For $k = 0, \ldots, r$ let $G_k = G[V_k] = (V_k, E_k)$ be the subgraph induced by $V_k = P_0 \cup \cdots \cup P_k$, let C_k be the outer face of G_k. Partition Π is a* canonical ordering *of (G, v_1) if for each $k = 1, \ldots, r-1$:*

1. *C_k is a simple cycle.*
2. *Each vertex z_i in P_k has a neighbor in $V \setminus V_k$.*
3. *$|P_k| = 1$ or $\deg_{G_k}(z_i) = 2$ for each vertex z_i in P_k.*

P_k is called a singleton *if $|P_k| = 1$ and a* chain *otherwise.*

A canonical ordering Π is refined to a *canonical vertex ordering* $v_1, \ldots, v_n$ by ordering the vertices in each $P_k, k > 0$, according to their clockwise appearance on C_k (see Figs. 1(a)-1(c)).

The following observations help build an intuitive understanding of canonical orderings. Note that Propositions 4 and 5 of Lemma 1 are part of Kant's original definition.

Lemma 1. *For $k = 1, \ldots, r-1$:*

1. *P_k has no chord.*
2. *For each vertex v in P_k there is a v-v_n-path $v = v_{k_0}, \ldots, v_{k_\ell} = v_n$ where each v_{k_i} is in P_{k_i} and $k_i < k_j$ for $0 \leq i < j \leq \ell$. Especially:*
   (a) *$G[V \setminus V_k]$ is connected.*
   (b) *If $\deg_{G_k}(v) = 2$, then v* is in *C_k.*
   (c) *P_k is on C_k.*
3. (a) *A singleton P_{k+1} and a path of C_k bound some faces or*
   (b) *a chain P_{k+1} and a path of C_k bound one face.*
4. *G_k is biconnected.*
5. *If v, w is a separation pair of G_k, then both are on C_k.*

Proof. The properties are directly implied by the fact that G is triconnected and by the definition of a canonical ordering. □

Remark 1. Two incident faces of a triconnected planar graph share one vertex or one edge. Especially, no face has a chord.

2.1 Leftmost Canonical Ordering

Kant [28] introduced a leftmost and rightmost canonical ordering of G. Let $P_0, \ldots, P_k$ be a sequence of paths that can be extended to a canonical ordering of G. A path P of G is a *feasible candidate* for the step $k+1$ if also $P_0, \ldots, P_k, P$ can be extended to a canonical ordering. Let $v_1 = c_1, c_2, \ldots, c_q = v_2$ be the vertices from left to right on C_k. Let c_ℓ be the neighbor of P on C_k such that ℓ is as small as possible. We call c_ℓ the *left neighbor* of P.

Definition 2 (leftmost canonical ordering). *A canonical ordering $P_0, \ldots, P_r$ is called* leftmost (rightmost) *if for $k = 0, \ldots, r-1$ the following is true. Let c_ℓ be the left neighbor of P_{k+1} and let $P_{k'}, k+1 \leq k' \leq r$, be a feasible candidate for the step $k+1$ with left neighbor $c_{\ell'}$. Then either (1) $\ell \leq \ell' (\ell \geq \ell')$ or (2) there is an edge between P_{k+1} and $P_{k'}$ (see Fig. 1(b)).*

Note that once a canonical ordering is known a simple linear-time algorithm can be used to rearrange its paths so that it becomes leftmost [28]. Also note that Kant did not use Condition 2 of a leftmost canonical ordering in his definition, however, he used it in his reordering algorithm. While leftmost canonical orderings are particularly useful for many applications, we stress that the rearrangement is applicable to any canonical ordering and that a leftmost canonical ordering is only unique with respect to a given partiton.

2.2 Leftist Canonical Ordering

In the leftist canonical ordering we add in each step the leftmost possible path where the choice is not only within an already given partition.

Definition 3 (leftist canonical ordering). *A canonical ordering $P_0, \ldots, P_r$ is called* leftist (rightist) *if for $k = 0, \ldots, r-1$ the following is true. Let c_ℓ be the left neighbor of P_{k+1} and let P be a feasible candidate for the step $k+1$ with left neighbor $c_{\ell'}$. Then $\ell \leq \ell' (\ell \geq \ell')$ (see Figs. 1(c) and 1(a)).*

Note that a feasible candidate for the step $k+1$ needs not to be a feasible candidate for the step $k+2$ anymore. Also note that the leftist canonical ordering is unique irrespective of a given partition and it is a leftmost canonical ordering. A leftist canonical ordering can also be found by choosing always the rightmost face or singleton in the algorithm of Kant [28]. A similar concept related to Schnyder realizers without clockwise cycles was defined for triangulated graphs in [6].

3 New Algorithm

Starting from $P_0 = \langle v_1, v_2 \rangle$, we build the canonical ordering by adding $P_1, \ldots, P_r$ in this order. In step $k+1$, the "belt" around G_k, i. e., the sequence of vertices not in G_k that lie on faces incident to G_k is considered. Then, a candidate not causing any "self-intersection" within the belt is chosen. Before we give the details, we start with a recursive definition of which paths will be considered in the step $k+1$.

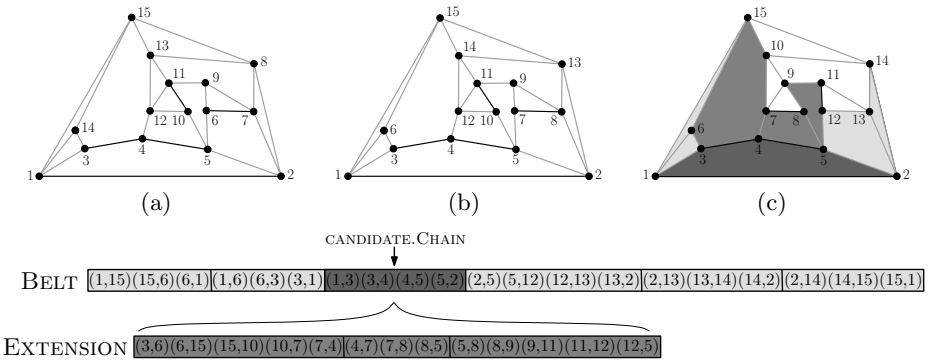

Fig. 1. Different canonical orderings (black paths are chains). (a) Rightist canonical ordering. (b) Leftmost canonical ordering respecting the ordering in (a). (c) Leftist canonical ordering and its construction. The light and dark grey faces are the belt of G_0. The next candidate in Algorithm 3 is $P_1 = \langle 3, 4, 5 \rangle$. Algorithm 5 substitutes the dark grey face by the middle grey faces, i. e., by the Extension found by Algorithm 4.

Definition 4 (cut faces and locally feasible candidates). *$P_0 = \langle v_1, v_2 \rangle$ is a locally feasible candidate. Let $P_0, \ldots, P_k$ be a sequence of locally feasible candidates and V_k, G_k, and C_k as in Definition 1. A* cut face *f of G_k is an inner face of G that is incident to some vertex on C_k but is not a face of G_k. Let P_f be the clockwise sequence of vertices incident to f that are not in V_k. If f is incident to an edge on C_k, then f is called a* candidate face *and P_f is called a* candidate *for the step $k+1$. A candidate face f and the candidate P_f are* locally feasible *for the step $k+1$ if*

1. *v_n is not in P_f or $P_0, \ldots, P_k, P_f$ is a partition of V,*
2. *$G[V \setminus (V_k \cup P_f)]$ is connected, and*
3. *P_f is a singleton or the degree of each vertex of P_f in $G[V_k \cup P_f]$ is two.*

In the remainder of this section, we will see that the locally feasible candidates are exactly the feasible candidates. We start with the following lemma which is a direct consequence of Definitions 1 and 4 and the triconnectivity of G.

Lemma 2. *1. A canonical ordering is a sequence of locally feasible candidates.*
2. If a sequence of locally feasible candidates partitions the whole vertex set of a triconnected graph, it is a canonical ordering.

In what follows, we consider the vertices on C_k to be from left to right between v_1 and v_2. Accordingly, we also consider the cut faces from left to right: A *cut edge* of G_k is an edge of G that is incident to one vertex in V_k and one vertex in $V \setminus V_k$. Let f and f' be two cut faces. Let c and c', respectively, be the leftmost vertices on C_k that are incident to f and f', respectively. We say that f is to the left of f' if c is to the left of c' on C_k or if $c = c'$, then the cut edges of f are to the left of the cut edges of f' in the incidence list of c.

Algorithm 1: Leftist Canonical Ordering

```
begin
    Let v_2, v_1, v_3, ..., v_ℓ be the bound of the inner face incident to {v_1, v_2}
    P_0 ← ⟨v_1, v_2⟩, P_1 ← ⟨v_3, ..., v_ℓ⟩, k ← 1
    while |V_k| < n − 1 do
        Let f be the leftmost locally feasible candidate face
        P_{k+1} ← P_f
        k ← k + 1
    P_{k+1} ← ⟨v_n⟩
```

Corollary 1. *If Algorithm 1 terminates, it computes the leftist canonical ordering of a triconnected planar graph.*

Before we prove that in each step there exists a locally feasible candidate face, we describe locally feasible candidates in terms of "self-intersection" of the belt. Let $P_0, \ldots, P_k$ be a sequence of locally feasible candidates. The *belt* of G_k is the sequence of vertices not in G_k that are incident to the cut faces of G_k from left to right. I. e., let $f_1, \ldots, f_s$ be the cut faces of G_k ordered from left to right. Let P_{f_0} be the vertices in $V \setminus V_k$ that are incident to the outer face in counterclockwise order. Then, the concatenation of $P_{f_1}, \ldots, P_{f_s}$ and P_{f_0} is the belt of G_k. Consider Fig. 1(a). Then, $P_2 = \langle 6, 7\rangle$, $P_3 = \langle 8\rangle$, and the belt of G_3 is $15, 14|14|14, 15, 13, 12|12, 10|10, 11, 9|9|9, 11, 13|13, 15|15$.

Definition 5 (forbidden, singular, stopper). *A vertex v of the belt of G_k is*

- forbidden *if v does not occur consecutively in the belt of G_k,*
- singular *if v occurs more than twice in the belt of G_k and its occurrence is consecutive, and*
- *a* stopper *if it is forbidden or singular.*

In the above example, 15, 13, and 11 are forbidden and 14 and 9 are singular vertices. Note that v_n is always the first and last vertex of the belt. Hence, it remains forbidden until the end. It will turn out that the locally feasible candidates are those that do not contain a stopper or that are singular singletons.

Lemma 3. *Let $P_0, \ldots, P_k$ be a sequence of locally feasible candidates. Let f be a candidate face for the step $k+1$ and let $P = P_f$.*

1. *If a vertex v of P is adjacent to more than two vertices in $V_k \cup P$, then v occurs more than twice in the belt.*
2. *If $G[V \setminus (V_k \cup P)]$ is not connected, then P contains a forbidden vertex.*
3. *If a vertex v of P is singular, then v is a locally feasible singleton.*
4. *If P contains a forbidden vertex v, then $G[V \setminus (V_k \cup P)]$ is not connected or P contains another vertex with more than two neighbors in $V_k \cup P$.*

Proof. 1. Let e be an edge incident to v and a vertex in $V_k \cup P$ that is not incident to f. By Remark 1, edge e is a cut edge and hence incident to two cut faces. Thus, v is incident to at least three cut faces.

2. Let W be the set of vertices in a connected component of the graph induced by $V \setminus (V_k \cup P)$ and not containing v_n. Since $V \setminus V_k$ was connected, W is adjacent to P and there is a path from P to v_n not intersecting W. By the triconnectivity of G, there is an edge between W and the part of C_k not contained in f. Further, there is at least a third vertex on $C_k \cup P$ adjacent to W. Let w be the rightmost vertex on $C_k \cup P$ that is adjacent to W and let v be the leftmost such vertex. Assume that w is on C_k. Then v is on P. Consider the face f' containing v and w. Then, the belt contains some vertices of W between the occurrences of v for the belt faces f and f' (see Fig. 2(a)).
3. If v is singular, then it is a candidate. By Proposition 2, $G[V \setminus (V_k \cup \{v\})]$ is connected.
4. Since v is forbidden, there is a cut face f' containing v and a cut face h between f and f' such that P_h contains a vertex $w \neq v$. If w is not incident to f, then w and v_n are in two connected components of $G[V \setminus (V_k \cup P)]$ (see Fig. 2(b)). So assume now that for all faces h' between f and f' the path $P_{h'}$ contains only vertices incident to f. Among these faces let h be the face that is next to f. By Remark 1, P_h consists of one vertex $w \neq v$ and w is singular (see Fig. 2(c)). □

Corollary 2. *1. A candidate that is a chain is locally feasible if and only if it does not contain any stopper.*
2. A vertex of the belt is a locally feasible singleton if and only if it is singular.

For example, the locally feasible candidates for the step $k + 1 = 4$ in Fig. 1(a) are $\langle 14 \rangle$, $\langle 12, 10 \rangle$, and $\langle 9 \rangle$.

Theorem 1. *Algorithm 1 computes the leftist canonical ordering of a triconnected planar graph.*

Proof. By Lemma 1, it remains to show that in each step of the algorithm there is a locally feasible candidate. By Corollary 2.2, if there are any singular vertices, we have a locally feasible candidate. So, assume now we do not have any singular vertices. By Corollary 2, we have to show that there is a candidate that does not contain any forbidden vertex.

Let f be a candidate face and let $P = P_f$. Assume that P contains a forbidden vertex v. Let f' be a cut face containing v such that the belt contains a vertex other than v between the occurrence of v in P_f and the occurrence of v in $P_{f'}$. Let $f, h_1, \ldots, h_\alpha, f'$ be the sequence of cut faces between f and f'. We show by induction on the number of forbidden vertices in $P_{h_1}, \ldots, P_{h_\alpha}$ that there is a locally feasible candidate among $P_{h_1}, \ldots, P_{h_\alpha}$.

By the choice of f' and by triconnectivity of G, there is at least one $i = 1, \ldots, \alpha$ such that P_{h_i} is a candidate that does not contain v. If v is the only forbidden vertex in $P_{h_1}, \ldots, P_{h_\alpha}$, then P_{h_i} is locally feasible.

If P_{h_i} contains a forbidden vertex w (recall that by our assumption there are no singular vertices), there is a cut face $h \neq h_i$ among $f, h_1, \ldots, h_\alpha, f'$ incident to w such that the belt contains a vertex other than w between the occurrence

of w in P_{h_i} and in P_h. The cut faces between h and h_i do not contain v. Hence, by the induction hypothesis, one of them is a locally feasible candidate. □

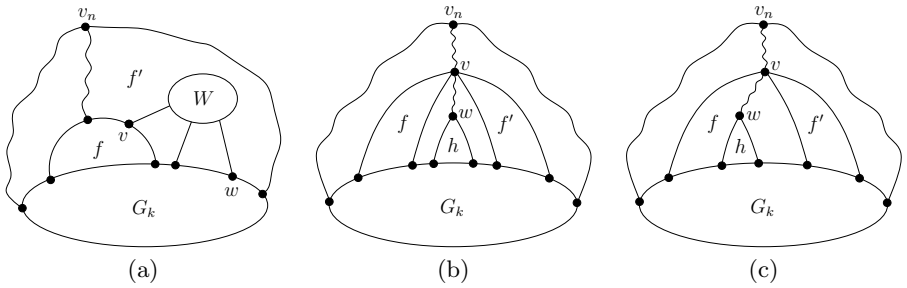

Fig. 2. Illustration of the proof of Lemma 3. (a) W is a connected component of $G[V \setminus (V_k \cup P_f)]$ not containing v_n. Faces f and f' are not consecutive in the belt of G_k. Thus, f contains a forbidden vertex v. (b, c) If v is forbidden, then (b) $G[V \setminus (V_k \cup P_f)]$ is not connected or (c) there is a singular vertex w.

4 Linear-Time Implementation

We will maintain a list BELT that represents the cut faces from left to right. For a simpler implementation, BELT contains lists of edges rather than one list of vertices and each cut face f is represented by a *belt item* which is a pair consisting of

- a list CHAIN of f's incident edges not in G_k in clockwise order and
- the rightmost stopper of P_f (if any).

We traverse the list BELT using a pointer CANDIDATE.

To decide whether a vertex is a stopper, we maintain two counters. Let CUTFACES(v) be the number of cut faces and CUTEDGES(v) the number of cut edges to which v is incident. In order to make the following lemma true also for v_n, we will count the outer face twice in CUTFACES(v_n).

Lemma 4. *A vertex v in the belt of G_k is*

- *forbidden if and only if* CUTFACES(v) > CUTEDGES(v) + 1 *and*
- *singular if and only if* 2 < CUTFACES(v) = CUTEDGES(v) + 1.

Proof. A vertex occurs once for each cut face it is incident to in the belt. Two occurrences of a vertex v in the belt are consecutive if and only if the corresponding cut faces share a cut edge incident to v. So, all occurrences of v in the belt are consecutive if and only if v is only incident to one more cut face than to cut edges. □

Algorithm 2: Leftist Canonical Ordering

Input : $G = (V, E)$ planar embedded triconnected undirected graph
$v_1 \in V$ on the outer face
Output : leftist canonical ordering $P_0, \ldots, P_k$ of (G, v_1)

canonicalOrdering

replace each $\{v, w\} \in E$ by (v, w) and (w, v)
$v_n \leftarrow$ clockwise neighbor of v_1 on outer face
$v_2 \leftarrow$ counterclockwise neighbor of v_1 on outer face

for $v \in V$ **do** cutFaces$(v) \leftarrow 0$; cutEdges$(v) \leftarrow 0$
cutFaces$(v_n) \leftarrow 1$

mark (v_1, v_2) and (v_2, v_1)
Belt $\leftarrow \langle(\langle(v_2, v_1), (v_1, v_2), (v_2, v_1)\rangle, \text{NIL})\rangle$

$k \leftarrow -1$
candidate $\leftarrow$ first item in Belt
while Belt $\neq \emptyset$ **do**
  $k \leftarrow k + 1$
  $P_k \leftarrow$ `leftmostFeasibleCandidate`
  `updateBelt`

The algorithm `canonicalOrdering` (see Algorithm 2) now works as follows. We start with a copy of G in which each undirected edge $\{v, w\}$ is replaced by the two directed edges (v, w) and (w, v). In the beginning, the belt is initialized by $(\langle(v_2, v_1), (v_1, v_2), (v_2, v_1)\rangle, \text{NIL})$. Thus, `leftmostFeasibleCandidate` (see Algorithm 3) chooses $P_0 = \langle v_1, v_2\rangle$ as the first path.

In general, each iteration in Algorithm 2 consists of three steps: (1) We choose the new leftmost locally feasible candidate P_k, (2) we find the new cut faces incident to P_k, and (3) we replace P_k by its incident cut faces in the belt and update its neighbors (see Fig. 1(c)). In detail:

`leftmostFeasibleCandidate` We traverse Belt from the current cut face candidate to the right doing the following: If candidate is a candidate face, traverse candidate.Chain from right to left until a stopper is found. If so, store it. If candidate.Chain contains no stopper or it is a singular singleton, it is the next locally feasible candidate. Otherwise, go to the next face. See Algorithm 3.

`beltExtension` To find the new cut faces, we traverse candidate.Chain from left to right. The outer repeat-loop iterates over all vertices incident to two edges of candidate.Chain. Each iteration finds the new cut faces incident to such a vertex and increments the counter cutEdges. In the inner repeat-loop, we traverse all new edges of a new cut face and store them in the list Chain. Here the counter cutFaces is incremented. Each list Chain is finally appended to the list Extension that stores all new belt items incident to candidate.Chain. See Algorithm 4.

`updateBelt` We replace CANDIDATE (and all its copies if it was a singleton) by the new cut faces found by `beltExtension`. The last edge of the predecessor and the first edge of the successor of CANDIDATE are removed since they are now contained in G_k. If the predecessor of CANDIDATE was not a candidate face before or it lost its stopper, then we go one step to the left in BELT and set CANDIDATE to its predecessor. See Algorithm 5.

Algorithm 3: Skip infeasible candidates

```
list leftmostFeasibleCandidate
    FOUND ← false
    repeat
        let ⟨(v_0, v_1), (v_1, v_2), ..., (v_ℓ, v_{ℓ+1})⟩ := CANDIDATE.CHAIN
        if v_0 ≠ v_{ℓ+1} then
            j ← ℓ
            while j > 0 and not(forbidden(v_j) or singular(v_j)) do j ← j − 1
            if j > 0 then CANDIDATE.STOPPER ← v_j
            if j = 0 or                                          then
               (singular(CANDIDATE.STOPPER) and ℓ = 1)
                FOUND ← true
                for (v, w) ∈ CANDIDATE.CHAIN do mark (w, v)
        if not FOUND then
            CANDIDATE ← successor(CANDIDATE)
            if CANDIDATE = NIL then HALT: illegal input graph
    until FOUND
    return ⟨v_1, ..., v_ℓ⟩
```

Theorem 2. *Algorithm 2 computes the leftist canonical ordering of a triconnected planar graph in linear time.*

Proof. **Linear running time:** In the algorithm `beltExtension` each edge is touched at most twice. In the algorithm `leftmostFeasibleCandidate` each candidate is scanned from right to left until the first stopper occurs. All the scanned edges will have been deleted from the list when the candidate will be scanned the next time. In total only $2m$ edges will be added to BELT.

Correctness: While scanning BELT from left to right, we always choose the leftmost locally feasible candidate: Assume that at step $k+1$ we scan a face f and there are no locally feasible candidates to the left of f. The face f is omitted because it is not a candidate or it contains a stopper. None of the two properties changes if no direct neighbor in BELT had been added to G_k. Hence, as long as f is not locally feasible, no face to the left of f has to be considered. Further, the number of incident cut faces or cut edges of a vertex never decreases. We show that a candidate can only become locally feasible after his rightmost stopper has become singular.

Algorithm 4: Construct list of new belt items incident to P_k

```
list beltExtension(list ⟨e_0, ..., e_ℓ⟩)
    EXTENSION ← ∅
    for j ← 1, ..., ℓ do /* check for new cut faces incident to source */
        v_start ← source(e_j)
        v_end ← target(e_j)
        FIRST ← e_j
        repeat
            FIRST = (v, w) ← clockwise next in N^+(v_start) after FIRST
            CUTEDGES(w) ← CUTEDGES(w) + 1
            if FIRST not marked then /* new cut face */
                CHAIN ← ∅
                e ← (v, w)
                repeat /* traverse clockwise */
                    mark e
                    append CHAIN ← e
                    CUTFACES(w) ← CUTFACES(w) + 1
                    e = (v, w) ← counterclockwise next in N^+(w) after (w, v)
                until w ∈ {v_start, v_end}
                mark e
                append CHAIN ← e
                append EXTENSION ← (CHAIN, NIL)
        until w = v_end
    return EXTENSION
```

Algorithm 5: Replace feasible candidate with incident faces

```
updateBelt
    if singular(CANDIDATE.STOPPER) then
        remove neighboring items with same singleton from BELT
    PRED ← predecessor(CANDIDATE)
    SUCC ← successor(CANDIDATE)
    if SUCC ≠ ∅ then remove first edge from SUCC.CHAIN
    EXTENSION ← BeltExtension(CANDIDATE.CHAIN)
    replace CANDIDATE by EXTENSION
    if EXTENSION ≠ ∅ then
        CANDIDATE ← first item of EXTENSION
    else
        CANDIDATE ← SUCC
    if PRED ≠ ∅ then
        remove last edge (v, w) from PRED.CHAIN
        if v = PRED.STOPPER or w = source(first edge of PRED.CHAIN) then
            PRED.STOPPER ← NIL
            CANDIDATE ← PRED
```

Let v be the rightmost stopper of P_f and assume v is forbidden. Let f_L be the leftmost and f_R be the rightmost cut face containing v. We can conclude by the proof of Theorem 1 that all occurrences of v between f_L and f are consecutive and that Algorithm 2 finds the locally feasible candidates between f and f_R in the belt until the belt contains only v between f and f_R. It follows that all occurrences of v in the belt are consecutive. Let now v be singular. Then, the only two incident cut faces $f = f_L$ and f_R of v would share a cut edge $\{v, w\}$ that would not have been a cut edge before. Hence, w would have been a stopper of f to the right of v. □

Note that the algorithm for computing the leftist canonical ordering can also be used to compute the rightist canonical ordering. In that case, we store for each cut face the leftmost stopper and we scan the belt from right to left.

References

1. J. Barbay, L. C. Aleardi, M. He, and I. Munro. Succinct Representation of Labeled Graphs. In *Proc. 18th Int. Symp. on Algorithms and Computation (ISAAC'07)*, volume 4835 of *LNCS*, pages 316–328. Springer, 2007.
2. G. Barequet, M. T. Goodrich, and C. Riley. Drawing Planar Graphs with Large Vertices and Thick Edges. *J. of Graph Algorithms and Applications*, 8(1):3–20, 2004.
3. T. C. Biedl. Drawing Planar Partitions I: LL-Drawings and LH-Drawings. In *Proc. 14th Symp. on Computational Geometry*, pages 287–296. ACM Press, 1998.
4. T. C. Biedl and M. Kaufmann. Area-Efficient Static and Incremental Graph Drawings. In *Proc. 5th European Symp. on Algorithms (ESA'97)*, volume 1284 of *LNCS*, pages 37–52. Springer, 1997.
5. P. Bose, J. Gudmundsson, and M. Smid. Constructing Plane Spanners of Bounded Degree and Low Weight. *Algorithmica*, 42(3–4):249–264, 2005.
6. E. Brehm. 3-Orientations and Schnyder 3-Tree-Decompositions. Master's thesis, FU Berlin, 2000.
7. Y.-T. Chiang, C.-C. Lin, and H.-I. Lu. Orderly Spanning Trees with Applications to Graph Encoding and Graph Drawing. In *Proc. 12th ACM–SIAM Symp. on Discrete Algorithms (SODA'01)*, pages 506–515, 2001.
8. M. Chrobak and G. Kant. Convex Grid Drawings of 3-Connected Planar Graphs. *Int. J. of Computational Geometry and Applications*, 7(3):211–223, 1997.
9. M. Chrobak and S.-I. Nakano. Minimum-Width Grid Drawings of Plane Graphs. *Computational Geometry*, 11(1):29–54, 1998.
10. M. Chrobak and T. H. Payne. A Linear-Time Algorithm for Drawing a Planar Graph on a Grid. *Information Processing Letters*, 54(4):241–246, 1995.
11. R. C.-N. Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I. Lu. Compact Encodings of Planar Graphs via Canonical Orderings and Multiple Parentheses. In *Proc. 25th Int. Colloquium on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *LNCS*, pages 118–129. Springer, 1998.
12. H. de Fraysseix and P. O. Mendez. Regular Orientations, Arboricity, and Augmentation. In *DIAMCS Int. Workshop*, volume 894 of *LNCS*, pages 111–118. Springer, 1995.

13. H. de Fraysseix, J. Pach, and R. Pollack. Small Sets Supporting Fáry Embeddings of Planar Graphs. In *Proc. 20th ACM Symp. on the Theory of Computing (STOC'88)*, pages 426–433. ACM Press, 1988.
14. H. de Fraysseix, J. Pach, and R. Pollack. How to Draw a Planar Graph on a Grid. *Combinatorica*, 10(1):41–51, 1990.
15. G. Di Battista, R. Tamassia, and L. Vismara. Output-Sensitive Reporting of Disjoint Paths. *Algorithmica*, 23(4):302–340, 1999.
16. E. Di Giacomo, W. Didimo, and G. Liotta. Radial Drawings of Graphs: Geometric Constraints and Trade-Offs. *J. of Discrete Algorithms*, 6(1):109–124, 2008.
17. E. Di Giacomo, W. Didimo, G. Liotta, and S. K. Wismath. Curve-Constrained Drawings of Planar Graphs. *Computational Geometry*, 30(2):1–23, 2005.
18. V. Dujmović, M. Suderman, and D. R. Wood. Really Straight Graph Drawings. In GD'04 [21], pages 122–132.
19. C. Erten and S. G. Kobourov. Simultaneous Embedding of Planar Graphs with Few Bends. In GD'04 [21], pages 195–205.
20. U. Fößmeier, G. Kant, and M. Kaufmann. 2-Visibility Drawings of Planar Graphs. In *Proc. 4th Int. Symp. on Graph Drawing (GD'96)*, volume 1090 of *LNCS*, pages 155–168. Springer, 1997.
21. *Proc. 12th Int. Symp. on Graph Drawing (GD'04)*, volume 3383 of *LNCS*. Springer, 2005.
22. M. T. Goodrich and C. G. Wagner. A Framework for Drawing Planar Graphs with Curves and Polylines. *J. of Algorithms*, 37(2):399–421, 2000.
23. C. Gutwenger and P. Mutzel. Planar Polyline Drawings with Good Angular Resolution. In *Proc. 6th Int. Symp. on Graph Drawing (GD'98)*, volume 1547 of *LNCS*, pages 167–182. Springer, 1999.
24. D. Harel and M. Sardas. An Algorithm for Straight-Line Drawing of Planar Graphs. *Algorithmica*, 20:119–135, 1998.
25. X. He. On Floor-Plan of Plane Graphs. *SIAM J. on Computing*, 28(6):2150–2167, 1999.
26. X. He, M.-Y. Kao, and H.-I. Lu. Linear-Time Succinct Encodings of Planar Graphs via Canonical Orderings. *SIAM J. on Discrete Mathematics*, 12(3):317–325, 1999.
27. G. Kant. Drawing Planar Graphs using the lmc-Ordering. Technical Report RUU-CS-92-33, Dep. of Information and Computing Sciences, Utrecht University, 1992.
28. G. Kant. Drawing Planar Graphs Using the Canonical Ordering. *Algorithmica*, 16(4):4–32, 1996.
29. G. Kant. A More Compact Visibility Representation. *Int. J. of Computational Geometry and Applications*, 7(3):197–210, 1997.
30. G. Kant and X. He. Regular Edge Labeling of 4-Connected Plane Graphs and its Applications in Graph Drawing Problems. *Theoretical Computer Science*, 172(1-2):175–193, 1997.
31. K. Miura, M. Azuma, and T. Nishizeki. Canonical Decomposition, Realizer, Schnyder Labeling and Orderly Spanning Trees of Plane Graphs. *Int. J. of Foundations of Computer Science*, 16(1):117–141, 2005.
32. S.-I. Nakano. Planar Drawings of Plane Graphs. *IEICE Transactions on Information and Systems*, E83-D(3):384–391, 2000.
33. W. Schnyder. Embedding Planar Graphs on the Grid. In *Proc. 1st ACM–SIAM Symp. on Discrete Algorithms (SODA'90)*, pages 138–148, 1990.
34. K. Wada and W. Chen. Linear Algorithms for a k-Partition Problem of Planar Graphs. In *Proc. 24th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG'98)*, LNCS, pages 324–336. Springer, 1998.