Wir erhalten so den Satz: Dann und nur dann, wenn das Schema des Graphen I) durch die angegebene Umschaltungsoperation U eine BZ-Reihe liefert, II) durch die Umkehrungsoperation wieder aus der BZ-Reihe entsteht, stellt das Schema einen auf der Kugelflhäche realisierbaren Graphen dar. *Damit ist dan Gaussische Problem für den allgemeinsten Graphen vierter Ordnung gelöst.*

[Thus we have the following theorem: A graph schema represents a planar graph if and only if I) applying the switching operation U to the schema produces a tree-onion code, and II) applying the inversion operation to the tree-onion code recovers the original schema. *Thus, Gauss's problem is solved for 4-regular graphs.]*

— Max Dehn, "Über Combinatorishe Topologie" (1936)

2 Planar Curves

2.1 Definitions and Representations

Recall that a *closed curve in the plane* is a continuous function $\gamma\colon S^1 \to \mathbb{R}^2$. Previously we considered *simple* closed curves, where the function γ is injective; for this lecture we remove this restriction.

▷▷▷▷▷ **⟨⟨paths, concatenation, reversal⟩⟩**

Arbitrary closed curves are nasty horrible things that will eat your face off; if you thought topologist's sine curves and Peano and Hilbert curves were bad, just think of the shenanigans you can get up to when the curve is allowed to self-intersect! Fortunately, though, this is a *computational* topology class, which means we only need to consider curves that have some finite representation that can be manipulated algorithmically. I'll focus on three common representations:

2.1.1 Polygons

Polygons are the simplest (and oldest) model for planar curves. Recall that a *polygon* is a piecewise-linear closed curve. Any polygon is naturally represented by a finite sequence of points $p_0, p_1, \ldots, p_{n-1}$ called its *vertices*; the polygon itself is composed of the line segments $p_i p_{i+1 \bmod n}$ for each index i, called its *edges*. We can usually (but *not* always!) assume without loss of generality that the polygon is in *general position*, meaning that all vertices are distinct, no vertex lies in the interior of an edge, and at most two edges intersect at any point.

2.1.2 Walks in graphs

Closed curves can also be represented as closed walks in some underlying plane graph G. The planar embedding of G may be specified either purely combinatorially (as a rotation system) or piecewise-linearly (by specifying coordinates for each vertex). Here we usually cannot make any general positions assumptions; in particular, the same walk may revisit the same vertex multiple times, or traverse the same edge multiple times in either direction. The edges of the underlying graph G may have weights; the length of a walk is the sum of the weights of its edges, counted with appropriate multiplicity.

A variant of this representation considers any curve to lie *arbitrarily close* to G rather than in G itself. Replace each vertex of of G with a small disk and each edge of G with a ribbon between the disks of its endpoints. Within this "ribbon graph", we can perturb any curve γ into general position, so that γ intersects only within the vertex disks, and only transversely. Thus, the intersection of the curve with any edge ribbon consists of disjoint parallel paths. We often don't care how the curve behaves inside the vertex disks, but if necessary, we can assume without loss of generality that any two curve segments inside a vertex disk intersect once transversely or not at all. The actual representation stores the walk in G, together with a left-to-right ordering of the curve segments within each edge ribbon.

Equivalently, we can specify a curve by its intersections with the edges of the dual map G^*; Éric Colin de Veridère and I named this the *cross-metric* model. We assume that the represented curve γ avoids the vertices of G^*, that each edge of G^* intersects γ transversely at distinct points, and (typically) that any two segments of γ within the same face intersect at most once. The ***overlay*** of G^* and γ has a vertex for each intersection of γ with edges of G^*, and an edge for every segment of γ or G^* between the overlay vertices. Then γ itself is represented as a *simple* walk in this overlay graph. The length of a curve γ in the cross-metric model is the sum of the weights of the edges of G^* that γ *crosses*, counted with appropriate multiplicity.

Any polygon can be converted into a walk in its image graph; conversely, if the underlying graph G is embedded piecewise-linearly (or in the cross-metric model, if every segment within a face is a line segment), then every walk in G *is* a polygon.

2.1.3 Generic curves

Finally, we can represent sufficiently "nice" curves directly by ignoring their geometry entirely. A self-intersection $\gamma(s) = \gamma(t)$ of a curve γ is ***transverse*** if, for some $\varepsilon > 0$, the subpaths $\gamma(s-\varepsilon, s+\varepsilon)$ and $\gamma(t-\varepsilon, t+\varepsilon)$ are homeomorphic to two orthogonal lines. A closed curve is ***generic*** if every self-intersection is transverse; compactness implies that a generic immersion has a finite number of self-intersection points. Generic curves are sometimes called *immersions* or *regular curves*, but that term more commonly refers to closed curves with continuous non-zero derivatives [11]. In fact, the most common word used to describe this class of curves is *"curve"*!

The term "generic" is justified by the observation that *every* closed curve is arbitrarily close to a generic closed curve. More formally:

Lemma 2.1. *For any closed curve $\gamma\colon S^1 \to \mathbb{R}^2$ and* any $\varepsilon > 0$, *there is a generic closed curve $\gamma'\colon S^1 \to \mathbb{R}^2$ such that $\|\gamma(t) - \gamma'(t)\| < \varepsilon$ for all $t \in S^1$.*

Proof: Fix a closed curve $\gamma\colon S^1 \to \mathbb{R}^2$ and a real number $\varepsilon > 0$. By choosing an arbitrary basepoint on the circle, we can regard γ as a periodic function $\gamma\colon \mathbb{Z} \to \mathbb{R}^2$ where $\gamma(t) = \gamma(t+1)$ for all t. Compactness implies that we can cover γ with a finite number of balls $b_0, b_1, b_2, \dots, b_n$ of radius $\varepsilon/4$, where each ball is centered at some point $\gamma(t)$, and in particular, b_0 is centered at $\gamma(0) = \gamma(1)$. For any index i, let B_i denote the ball of radius $\varepsilon/2$ with the same center as b_i.

We inductively define a finite sequence of real numbers $t_0 < t_1 < \cdots < t_N$ and indices $j_0, j_1, j_2, \dots, j_N$ as follows. First, let $t_0 = 0$ and $j_0 = 0$. Then for any index $i > 0$, define

$$t_i = \min\bigl\{1,\ \min\{t > t_{i-1} \mid \gamma(t) \notin B_{j_{i-1}}\}\bigr\}$$

If $t_i = 1$, we set $j_i = 0$; otherwise, let j_i be any index such that $\gamma(t_i) \in b_{j_i}$. Compactness implies that γ is uniformly continuinuous: There is a real number $\delta > 0$ such that for any t and t', if $\|\gamma(t) - \gamma(t')\| < \varepsilon/4$, then $|t - t'| < \delta$. It follows that $t_N = 1$ for some integer $N \le 1/\delta$.

The points $\gamma(t_0), \gamma(t_1), \dots, \gamma(t_N)$ define an n-vertex polygon $\tilde{\gamma}$, which we can parametrize so that $\tilde{\gamma}(t_i) = \gamma(t_i)$ for every index i. By construction, we have $\|\gamma(t) - \tilde{\gamma}(t)\| < \varepsilon/2$ for all $t \in S^1$.

This polygon is not necessarily generic; in particular, it may contain repeated vertices. However, with probability 1 we can obtain a *generic* polygon γ' by moving each vertex of $\tilde{\gamma}$ a distance of $\varepsilon/4$ in a random direction (chosen uniformly from S^1). Convexity of the Euclidean norm implies that $\|\gamma'(t) - \tilde{\gamma}'(t)\| < \varepsilon/2$ for all $t \in S^1$. Thus, the triangle inequality implies $\|\gamma(t) - \gamma'(t)\| < \|\gamma(t) - \tilde{\gamma}(t)\| + \|\tilde{\gamma}(t) - \gamma'(t)\| < \varepsilon$ for all $t \in S^1$. □

Any non-simple generic curve γ can be naturally represented by its *image graph*, which is a connected 4-regular plane graph, whose vertices are the points of pairwise self-intersection.[1] Any generic curve γ is a certain canonical Euler tour of its image graph; whenever the tour enters a vertex through a dart d, it exits that vertex through the opposite dart $rev(succ(succ(d)))$. Thus, the graph can be represented by the image graph itself, with no additional information. As usual, the image graph can be represented either combinatorially or geometrically.

Alternatively, any non-simple generic curve can be represented by a ***Gauss code***, defined by Gauss as follows. Assign each self-intersection point of γ a unique label; the gauss code of γ is the sequence of labels encountered by a point moving once around γ, starting at an arbitrary *basepoint*. A ***signed*** Gauss code also records how the curve crosses itself at each vertex: positive for right-to-left crossings and negative for left-to-right crossings.

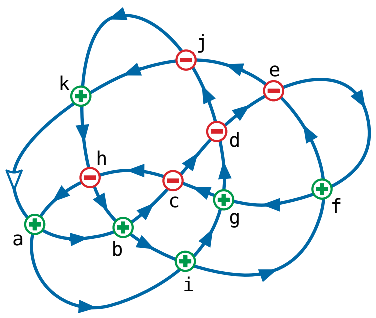


Figure 2.1. A curve with signed Gauss code $\mathtt{a^+b^+c^-d^-e^-f^+g^+c^+h^-a^-i^+g^-d^+j^-k^+h^+b^-i^-f^-e^+j^+k^-}$.

2.2 Gauss Code Planarity

Around 1830, Gauss [5] asked how to determine whether a given Gauss code represents a planar curve, or more succinctly, whether a given Gauss code is planar. Francis described the first algorithmic solution for *signed* Gauss codes, which consisted of just twelve instructions [4]; Carter later described a much more natural and general (if somewhat less efficient) solution [1].[2] Carter observed that every signed Gauss code corresponds to a unique 4-regular graph with a unique rotation system. Every 4-regular graph in the plane with n vertices has exactly $n+2$ faces, by Euler's formula. Thus, a signed Gauss code is planar if and only if the induced graph embedding has exactly $n+2$ faces.[3]

Gauss actually asked a much more difficult question: Which *unsigned* codes correspond to planar curves? This is arguably the very first nontrivial computational topology problem!Every unsigned Gauss code also represents a unique 4-regular graph with a well-defined cyclic order of edges around each vertex, but leaves unspecified which rotations are clockwise or counterclockwise. Gauss's question is to determine whether the cyclic orders can be oriented so that the resulting map is planar.

[1]The image of a simple generic curve is obviously a simple cycle.

[2]It is possible that Francis's algorithm is actually equivalent to Carter's, but I haven't succeeded in disassembling Francis's uncommented machine code.

[3]More generally, any Gauss code induces an embedding of a 4-regular graph onto *some* orientable surface, which later authors call the *Carter surface* of the Gauss code.

2.2.1 Gauss's Parity Condition

Gauss observed without proof that the Gauss code of every planar curve satisfies a simple parity condition: Every substring that starts and ends with the same symbol has even length, or equivalently, each symbol appears once at an even index and once at an odd index. This parity condition was first proved necessary by Nagy [8], ultimately by reduction to the Jordan curve theorem; an elementary proof was later given by Rademacher and Toeplitz [9]. Gauss also observed that the sequences `abcadcedbe` and `abcabdecde` satisfy his parity condition but cannot be realized by planar curves, so the parity condition is not sufficient.

In fact, Nagy [8] actually described a complete algorithm to reconstruct a planar curve from its Gauss code, and thus to determine whether a Gauss code is planar. Nagy's algorithm is very different from the approach described below, and it's unclear whether it can be implemented to run as quickly, so I will defer a detailed description of Nagy's approach to the homework.

2.2.2 Dehn's Tree-Onion Condition

About 100 years after Gauss, Dehn [2] described *das Gaussische Problem der Trakte* and proposed another algorithm to detect planar Gauss codes. Dehn first "uncrosses" the given Gauss code by reversing every substring between identical letters, in arbitrary order. If the given Gauss code is planar, each reversal corresponds to smoothing a vertex of the curve, reversing one of the two subcurves that start and end at that vertex, so that the overall smoothed curve remains connected.

For example, given the Gauss code `abcdefgchaigdjkhbifejk`, Dehn might proceed as follows, reversing the substrings in alphabetical order:

$$\begin{array}{lcl}
\underbrace{\texttt{abcdefgcha}}\texttt{igdjkhbifejk} & \rightsquigarrow & \underbrace{\texttt{ahcgfedcba}}\texttt{igdjkhbifejk} \\
\texttt{ahcgfedc}\underbrace{\texttt{baigdjkhb}}\texttt{ifejk} & \rightsquigarrow & \texttt{ahcgfedc}\underbrace{\texttt{bhkjdgiab}}\texttt{ifejk} \\
\texttt{ah}\underbrace{\texttt{cgfedc}}\texttt{bhkjdgiabifejk} & \rightsquigarrow & \texttt{ah}\underbrace{\texttt{cdefgc}}\texttt{bhkjdgiabifejk} \\
\texttt{ahc}\underbrace{\texttt{defgcbhkjd}}\texttt{giabifejk} & \rightsquigarrow & \texttt{ahc}\underbrace{\texttt{djkhbcgfed}}\texttt{giabifejk} \\
\texttt{ahcdjkhbcgf}\underbrace{\texttt{edgiabife}}\texttt{jk} & \rightsquigarrow & \texttt{ahcdjkhbcgf}\underbrace{\texttt{efibaigde}}\texttt{jk} \\
\texttt{ahcdjkhbcg}\underbrace{\texttt{fef}}\texttt{ibaigdejk} & \rightsquigarrow & \texttt{ahcdjkhbcg}\underbrace{\texttt{fef}}\texttt{ibaigdejk} \\
\texttt{ahcdjkhbc}\underbrace{\texttt{gfefibaig}}\texttt{dejk} & \rightsquigarrow & \texttt{ahcdjkhbc}\underbrace{\texttt{giabifefg}}\texttt{dejk} \\
\texttt{a}\underbrace{\texttt{hcdjkh}}\texttt{bcgiabifefgdejk} & \rightsquigarrow & \texttt{a}\underbrace{\texttt{hkjdch}}\texttt{bcgiabifefgdejk} \\
\texttt{ahkjdchbcg}\underbrace{\texttt{iabi}}\texttt{fefgdejk} & \rightsquigarrow & \texttt{ahkjdchbcg}\underbrace{\texttt{ibai}}\texttt{fefgdejk} \\
\texttt{ahk}\underbrace{\texttt{jdchbcgibaifefgdej}}\texttt{k} & \rightsquigarrow & \texttt{ahk}\underbrace{\texttt{jedgfefiabigcbhcdj}}\texttt{k} \\
\texttt{ah}\underbrace{\texttt{kjedgfefiabigcbhcdjk}} & \rightsquigarrow & \texttt{ah}\underbrace{\texttt{kjdchbcgibaifefgdejk}}
\end{array}$$

The resulting string encodes a weakly simple closed curve that touches itself tangentially at each of the original vertices. Because we can reverse substrings in any order, the resulting touch code (and the corresponding curve) is not unique. If we reverse the substrings one at a time by brute force, the entire untangling process requires $O(n^2)$ time, but as we'll see shortly, it is possible to untangle any Gauss code in $O(n)$ time.

The ***Gauss diagram*** of the touch code consists of a cycle of $2n$ vertices, labeled by the symbols in the touch code in order, plus edges joining each pair of identical symbols. Dehn proved that if a Gauss code is planar, then the Gauss diagram of the resulting touch code is a planar graph; that is, we can embed some of the chords inside the circle and the rest of the chords outside the circle so that no pair of chords intersects. Dehn referred to planar Gauss diagrams as "Baum-Zweibel Figuren" ["tree-onion diagrams"] and their corresponding codes as "Baum-Zweibel Reihen" ["tree-onion strings"], because they can also be used to describe tree-cotree decompositions of arbtirary plane graphs.

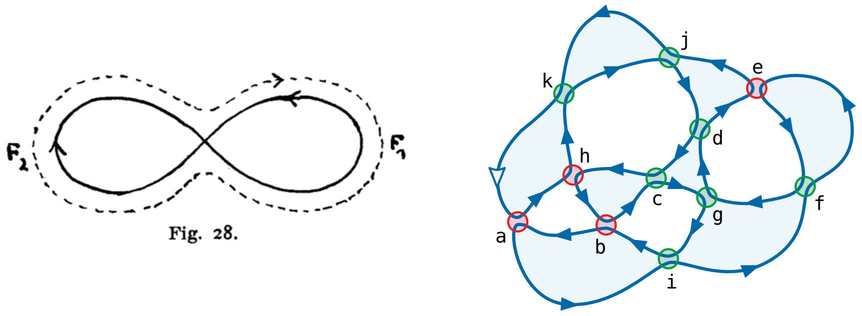


Figure 2.2. Left: Smoothing a vertex of a planar curve, from Dehn [2]. Right: A smoothed curve with Gauss touch code ahkjdchbcgibaifefgdejk.

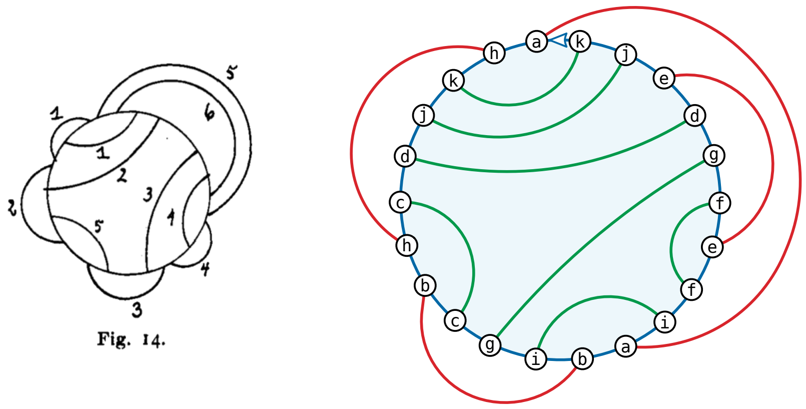


Figure 2.3. Left: A tree-onion figure, from Dehn [2]. Right: The planar Gauss diagram of ahkjdchbcgibaifefgdejk.

Dehn's condition can expressed more efficiently in terms of a different graph, called the ***interleave graph*** of the code. The interleave has n vertices, one for each distinct symbol in the touch code, and any edge between any two symbols that interleave $x \ldots y \ldots x \ldots y$ in the code. A Gauss diagram is planar if and only if its interleave graph is *bipartite*. The interleave graph has $O(n^2)$ vertices and edges, and we can easily check bipartiteness in $O(n^2)$ time.

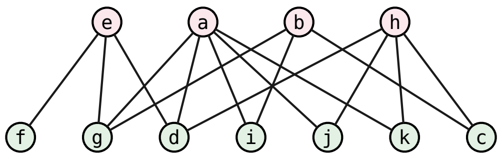
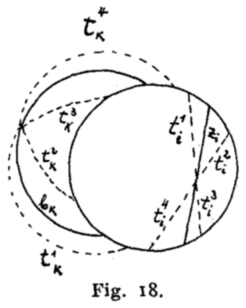


Figure 2.4. The bipartite interleave graph of ahkjdchbcgibaifefgdejk.

To complete his algorithm, Dehn observed that we can transform any Gauss diagram into a 4-regular graph by replacing each chord with a pair of crossing chords with a crossing, as shown in Figure 2.5. This recrossing process yields a single closed curve consistent with our original Gauss code then, trivially, the original Gauss code is planar. Otherwise, the original Gauss code is not planar.

Dehn observed that his tree-onion condition is necessary but not sufficient for planarity—consider the Gauss code abab—and asked whether Gauss's parity condition and Dehn's tree-onion condition are sufficient. In fact, these two conditions *are* sufficient, as proved by Dowker and Thistlethwaite [3] almost 50 years later.



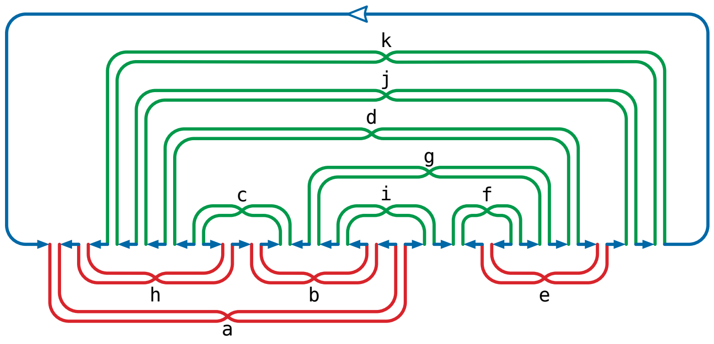


Figure 2.5. Left: Building a closed curve from a tree-onion diagram, from Dehn [2]. Right: A planar curve consistent with the original Gauss code `abcdefgchaigdjkhbifejk`, after Dehn [2] and Kaufmann [7].

2.2.3 Modern Proof

Rather than repeating Dowker and Thistlethwaite's argument, I'll give a simple self-contained characterization of planar Gauss codes with a complete proof. This characterization is ultimately based on the conditions of Gauss and Dehn, but uses more modern algorithmic tools described by Rosensteihl and Tarjan [10]. (Rosensteihl and Tarjan [10] describe an efficient implementation of Dehn's algorithm, including the second untangling phase, without exploiting Gauss's parity condition.)

We start with a simple consequence of the Jordan curve theorem.

Lemma 2.2. *Every pair of generic closed curves that intersect only transversely intersect at an even number of points.*

Proof: Let α and β be a generic pair of closed curves. Recall the parity test for deciding whether a point is in the interior of a simple closed curve or not. The same parity test allows us to color the faces of β alternately black and white, so that any two faces that share an edge have opposite colors.

Now imagine a point moving around α; each time this point crosses β, it moves from a white face to a black face or vice versa. The moving point starts and ends in the same face, and therefore must change color an even number of times. □

Now let X be a string of length $2n$, in which each of the n unique symbols appears twice.

Lemma 2.3. *If X is the Gauss code of a planar curve, then every substring of X that starts and ends with the same symbol has even length.*

Proof: Let γ be a planar closed curve. Smoothing γ at any vertex produces two subcurves α and β. Up to a cyclic shift (reflecting a change of basepoint), the Gauss code for γ can be written as $\mathtt{a}x\mathtt{a}y$, where string x encodes the vertices along α and string y encodes the vertices along β. Each self-intersection point of α is encoded in x twice, and the other symbols of x encode the intersections between α and β. We conclude that x has even length, which completes the proof. □

The string X defines a 4-regular graph $G(X)$ whose vertices are the n distinct symbols in X, and whose edges correspond to (cyclic) substrings of length 2. Moreover, X defines a particular Euler tour of $G(X)$; the edges of this tour are alternately directed forward and backward. Said differently, $G(X)$ is a *directed* graph with edges $x_i \to x_{i+1}$ and $x_i \to x_{i-1 \bmod 2n}$ for every even index i. For example, if $X = $ `abcdefgchaigdjkhbifejk`, the graph $G(X)$ consists of the following edges:

`a→b←c→d←e→f←g→c←h→a←i→g←d→j←k→h←b→i←f→e←j→k←a`

See Figure 2.6.

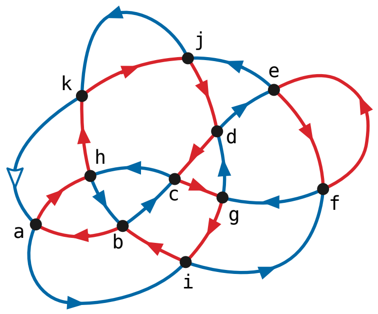


Figure 2.6. A curve with alternately oriented segments.

Theorem 2.4. *X is a planar Gauss code if and only if $G(X)$ has a planar embedding that contains a weakly simple Euler tour.*

Proof: First, suppose X is the Gauss code of a planar curve γ. Except for the edge orientations, $G(X)$ is the image graph of γ. Lemma 2.3 implies that every vertex of $G(X)$ has in-degree 2 and out-degree 2, so $G(X)$ has an Euler tour. Moreover, the edges incident to each vertex of $G(X)$ alternate in, out, in, out. It follows that *every* Euler tour of $G(X)$ is weakly simple.

Conversely, suppose $G(X)$ has a planar embedding that contains a weakly simple Euler tour T. Winding number arguments imply that the edges incident to each vertex of $G(X)$ alternate in, out, in, out. The original string X defines an *undirected* Euler tour U of $G(X)$, which traverses edges alternately forward and backward. U crosses itself at every vertex of $G(X)$. It follows immediately that U is a closed curve with Gauss code X. □

2.3 The Pile of Twin Stacks

Theorem 2.4 gives a complete characterization of planar Gauss codes, but it leaves open the question of how to decide planarity algorithmically. Most of the algorithm is straightforward; we can build the directed graph $G(X)$ and compute an Euler tour T in $O(n)$ time, and once we have a planar embedding of $G(X)$ that makes T weakly simple, we can extract the curve U in $O(n)$ time. The hard part is deciding whether T is a planar touch curve, or equivalently, whether the interlace graph of T is bipartite. We've already seen how to answer this question in $O(n^2)$ time. In the remainder of this node, I'll describe an algorithm of Rosenstiehl and Tarjan [10] that solves this planarity problem in $O(n)$ time.[4]

First we adopt a variant notation proposed independently by Dowker and Thistlethwaite [3]. We encode the Gauss code in an array $t[0\,..\,2n-1]$ that encodes for each index i the index of the other occurrence of symbol $T[i]$. Thus, for each index i, we have $T[t[i]] = T[i]$ but $t[i] \neq i$. For example, the Gauss code `ahkjdchbcgibaifefgdejk` defines the array

$$[12, 6, 21, 20, 18, 8, 1, 11, 5, 17, 13, 7, 0, 10, 16, 19, 14, 9, 4, 15, 3, 2]$$

[4]This is not the *first* algorithm to solve this planarity problem in linear time. About ten years earlier, Hopcroft and Tarjan [6] described the first linear-time algorithm to decide if an *arbitrary* graph is planar. At a very high level, Hopcroft and Tarjan's algorithm resembles Nagy's 1927 algorithm for checking curve planarity.

For the sake of illustration, we orient this index array vertically, and we process the indices in the array from the top down.

Each distinct symbol in the code represents a chord in the Gauss diagram. We want to classify these chords into two classes—"left" and "right"—which can be respectively embedded inside and outside the main circle of the Gauss diagram without intersecting. If we knew the left-right classification in advance, we could verify that no two chords intersect in $O(n)$ time using two stacks, as follows. Here we assume the chord symbols are the integers 1 through n, and the boolean array *IsLeft* indicates which symbols correspond to "left" chords.

```
VERIFYLR($T[0..2n-1]$, $t[0..2n-1]$, $IsLeft[1..n]$):
  $L \leftarrow$ new stack
  $R \leftarrow$ new stack
  for $i \leftarrow 0$ to $2n-1$
      if $t[i] > i$          ⟨⟨first endpoint⟩⟩
          if $IsLeft[T[i]]$
              PUSH($L, T[i]$)
          else
              PUSH($R, T[i]$)
      else                   ⟨⟨second endpoint⟩⟩
          if $IsLeft[T[i]]$
              $x \leftarrow$ POP($L$)
          else
              $x \leftarrow$ POP($R$)
          if $x \neq T[i]$
              return FALSE
      return TRUE
```

Of course we don't know the left-right classification in advance; that's what we need to compute! Rosenstiehl and Tarjan's algorithm builds this classification using a *stack* of pairs of stacks; to help avoid confusion, the outer stack is called a "pile". Each pair of stacks on the pile consists of a *left* stack L_i and a *right* stack R_i.

At the end of the jth iteration of the inner loop, the pile of stacks contains all values $t[i]$ such that $i < j$ and $t[i] \geq j$. In particular, at the start and end of the algorithm, the pile is empty. Each iteration of the algorithm maintains the following data structure invariants:

- For each level i, either L_i or R_i is non-empty.
- For each level i, all chords in $L_i \cup R_i$ lie in the same component of the interleave graph.
- For all level $i \neq j$, no chord in $L_i \cup R_i$ is interleaved with any chord in $L_j \cup R_j$.
- For each level i, the stacks L_i or R_i are each sorted in increasing order (that is, indices are pushed in decreasing order).
- The pile itself is also sorted in increasing order: For each level i, all indices in $L_i \cup R_i$ are smaller than all indices in $L_{i+1} \cup R_{i+1}$.

Intuitively, the stacks L_i and R_i in each pair contain chords of the Gauss diagram, represented by their higher-numbered endpoints, that must lie on opposite sides of the main circle. Equivalently, the interleave graph contains edges between every index in L_i and every index in R_i. Moreover, the chords in each stack are properly nested. However, the chords at each level can be independently embedded on either side, either with L_i on the left and R_i on the right, or L_i on the right and R_i

on the left. Thus, if the pile contains k levels, there are (at least) 2^k consistent embeddings of the chords in the pile.

★★★ Still need to actually describe the algorithm.

i	$T[i]$	$t[i]$	Pile of twin stacks	Operations	Interleaves found
0	a	12	$[12 \mid \bullet]$	new pair, push left	
1	h	6	$[6 \mid \bullet]\,,\,[12 \mid \bullet]$	new pair, push left	
2	k	21	$[6,12 \mid 21]$	meld, push right	ka, kh
3	j	20	$[6,12 \mid 20,21]$	push right	jh
4	d	18	$[6,12 \mid 18,20,21]$	push right	dh
5	c	8	$[6,12 \mid 8,18,20,21]$	push right	ch
6	h	1	$[12 \mid 8,18,20,21]$	pop left	
7	b	11	$[11,12 \mid 8,18,20,21]$	push left	bc
8	c	5	$[11,12 \mid 18,20,21]$	pop right	
9	g	17	$[11,12 \mid 17,18,20,21]$	push right	gb
10	i	13	$[11,12 \mid 13,17,18,20,21]$	push right	ib
11	b	7	$[12 \mid 13,17,18,20,21]$	pop left	
12	a	0	$[\bullet \mid 13,17,18,20,21]$	pop left	
13	i	10	$[\bullet \mid 17,18,20,21]$	pop right	
14	f	16	$[16 \mid \bullet]\,,\,[\bullet \mid 17,18,20,21]$	new pair, push left	
15	e	19	$[19 \mid 16,17,18,20,21]$	swap top pair, meld, push right	ef, eg
16	f	14	$[19 \mid 17,18,20,21]$	pop right	
17	g	9	$[19 \mid 18,20,21]$	pop right	
18	d	4	$[19 \mid 20,21]$	pop right	
19	e	15	$[\bullet \mid 20,21]$	pop left	
20	j	3	$[\bullet \mid 21]$	pop right	
21	k	2	$\varnothing$	pop right, pop empty pair	

Figure 2.7. Rosenstiehl and Tarjan's pile of twin stacks algorithm running on the string `ahkjdchbcgibaifefgdejk`

References

[1] J. Scott Carter. Classifying immersed curves. *Proc. Amer. Math. Soc.* 111(1):281–287, 1991.

[2] Max Dehn. Über kombinatorishe Topologie. *Acta Math.* 67:123–168, 1936.

[3] Clifford H. Dowker and Morwen B. Thistlethwaite. Classification of knot projections. *Topology Appl.* 16(1):19–31, 1983.

[4] George K. Francis. Null genus realizability criterion for abstract intersection sequences. *J. Comb. Theory* 7(4):331–341, 1969.

[5] Carl Friedrich Gauß. Nachlass. I. Zur Geometria situs. *Werke*, vol. 8, 271–281, 1900. Teubner. Originally written between 1823 and 1840.

[6] John Hopcroft and Robert E. Tarjan. Efficient planarity testing. *J. Assoc. Comput. Mach.* 21(4):549–569, 1974.

[7] Louis H. Kauffman. Virtual knot theory. *Europ. J. Combin.* 20(7):663–691, 1999. arXiv:math/9811028.

[8] Julius v. Sz. Nagy. Über ein topologisches Problem von Gauß. *Math. Z.* 26(1):579–592, 1927.

[9] Hans Rademacher and Otto Toeplitz. On closed self-intersecting curves. *The Enjoyment of Mathematics: Selections from Mathematics for the Amateur*, chapter 10, 61–66, 1990. Dover Publ. Originally published by Princeton Univ. Press, 1957.

[10] Pierre Rosenstiehl and Robert E. Tarjan. Gauss codes, planar Hamiltonian graphs, and stack-sortable permutations. *J. Algorithms* 5(3):375–390, 1984.

[11] Hassler Whitney. On regular closed curves in the plane. *Compositio Math.* 4:276–284, 1937.