Practical Level Planarity Testing and Layout with Embedding Constraints

Martin Harrigan* and Patrick Healy

Department of Computer Science and Information Systems,
University of Limerick, Ireland
{martin.harrigan, patrick.healy}@ul.ie

Abstract. We describe a practical method to test a leveled graph for level planarity and provide a level planar layout of the graph if the test succeeds, all in quadratic running-time. Embedding constraints restricting the order of incident edges around the vertices are allowed.

1 Introduction

A *leveled* graph, or one whose vertices have predetermined y-coordinates, is level planar if and only if it can be drawn in the plane satisfying the predetermined y-coordinates using straight-line edges without any edge crossings. We improve a previous test for level planarity [4] so that it provides a level planar layout of the graph if the test succeeds in quadratic running-time. We also handle a family of embedding constraints [3] that restrict the order of incident edges around the vertices.

There exists a level planarity testing and layout algorithm with linear running-time [2,5,6]. However, it is quite complicated, involving iterative updating of a set of PQ-trees, graph augmentations and an embedding algorithm for general planar graphs [1]. Our method, while requiring quadratic running-time is much simpler to understand and implement and can be naturally extended to handle embedding constraints.

2 Preliminaries

A *leveling* of a graph $G = (V, E)$ is a surjective mapping $\phi : V \rightarrow \{1, 2, \ldots, k\}$ such that $\phi(u) \neq \phi(v), \forall \{u, v\} \in E$. A leveling partitions the vertex set $V = V_1 \cup V_2 \cup \ldots \cup V_k$ such that $V_i = \phi^{-1}(i)$ and the edge set $E = E_1 \cup E_2 \cup \ldots \cup E_{k-1}$ such that $E_i \subseteq V_i \times V_{i+1}$. A leveling is *proper* if $\forall \{u, v\} \in E : |\phi(u) - \phi(v)| = 1$. In the following we assume all levelings to be proper.

The *vertex-exchange graph* or *ve-graph* $\mathcal{VE}(G, \phi) = (\mathcal{V}, \mathcal{E})$ of a graph $G = (V, E)$ with leveling ϕ is a graph with vertex set $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \ldots \cup \mathcal{V}_k$ where

* Supported by the Irish Research Council for Science, Engineering and Technology.

S.-H. Hong, T. Nishizeki, and W. Quan (Eds.): GD 2007, LNCS 4875, pp. 62–68, 2007.

$\mathcal{V}_j = \{\langle u, v\rangle | u, v \in V_j\}$ and edge set $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \cup \mathcal{E}_{k-1}$ where $\mathcal{E}_j = \{\{\langle t, w\rangle, \langle u, v\rangle\} | \{t, u\}, \{w, v\} \in E_j, \langle t, w\rangle \in \mathcal{V}_j, \langle u, v\rangle \in \mathcal{V}_{j+1}\}$.

In other words, $\mathcal{VE}(G, \phi)$ is constructed by taking the distinct pairs of vertices on the same level of G as vertices of $\mathcal{VE}(G, \phi)$ and joining two vertices in $\mathcal{VE}(G, \phi)$ whenever two pairs from the four corresponding vertices in G are joined by independent edges (see Fig. 1).

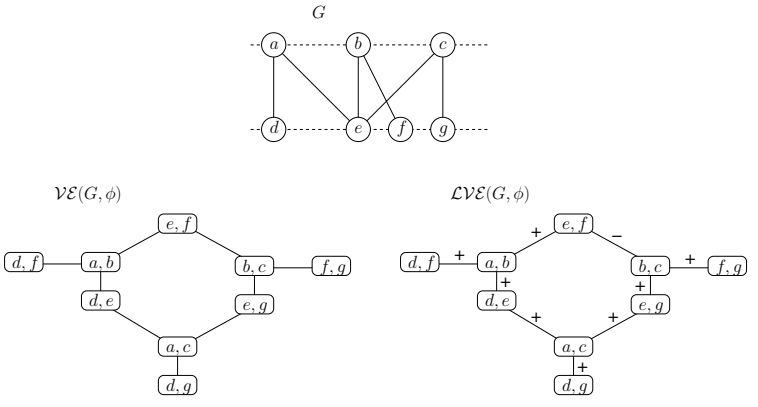

Fig. 1. A leveled graph, its ve-graph, and lve-graph

We augment the ve-graph $\mathcal{VE}(G, \phi) = (\mathcal{V}, \mathcal{E})$ with an edge labeling $\lambda : \mathcal{E} \to \{\text{'+'}, \text{'-'}\}$ to produce the *labeled vertex-exchange graph* or *lve-graph* $\mathcal{LVE}(G, \phi)$ as follows. Choose some initial level layout $\mathcal{L}$ of G. For every edge $e \in \mathcal{E}$ we set $\lambda(e) =$ '-' if the corresponding edges in G cross and $\lambda(e) =$ '+' if they do not (see Fig. 1).

3 Testing and Layout

The *ve-operation* $\texttt{ve}(\langle u, v\rangle)$ switches the labeling of every edge incident to $\langle u, v\rangle$ in $\mathcal{LVE}(G, \phi)$, *i.e.* '-' becomes '+' and '+' becomes '-'. This loosely corresponds to exchanging the position of the vertices u and v in the level layout $\mathcal{L}$ of G.

Theorem 1. *[4]*
A graph G with leveling ϕ is level planar if and only if there exists some sequence of ve-operations that removes all '-'-labeled edges from $\mathcal{LVE}(G, \phi)$, or equivalently, $\mathcal{LVE}(G, \phi)$ does not contain a cycle with an odd number of '-'-labeled edges.

Level planarity can therefore be tested in quadratic running-time using a simple depth-first search (DFS) traversal on the lve-graph. However, if the test succeeds, we need to solve the 3-*cycle problem* in order to produce a level planar layout within the same asymptotic running-time.

3.1 The 3-Cycle Problem

Suppose three vertices $\langle u, v\rangle$, $\langle v, w\rangle$ and $\langle u, w\rangle$ in some lve-graph representing the three vertices u, v and w in some leveled graph are not in the same connected component of the lve-graph. If we perform ve-operations to remove all the '-'-labeled edges using a DFS traversal then it may arise that $u \prec v \prec w \prec u$ where $\prec$ denotes the required order of the vertices along their respective level. Clearly, this is impossible.

Consider the leveled graph G and its lve-graph $\mathcal{LVE}(G, \phi)$ in Fig. 2. Note that $\langle e, f\rangle$, $\langle f, g\rangle$ and $\langle e, g\rangle$ are in different connected components of $\mathcal{LVE}(G, \phi)$. Suppose we begin the DFS traversal at $\langle e, f\rangle$, then visit $\langle b, c\rangle$ and perform $\texttt{ve}(\langle b, c\rangle)$ so that $e \prec f$. Suppose we continue the DFS traversal at $\langle f, g\rangle$, then visit $\langle a, b\rangle$ and perform $\texttt{ve}(\langle a, b\rangle)$ so that $f \prec g$. Now, suppose we continue the DFS traversal at $\langle a, c\rangle$, then visit $\langle e, g\rangle$ and perform $\texttt{ve}(\langle e, g\rangle)$ so that $g \prec e$. We have removed all '-'-labeled edges. However, $e \prec f \prec g \prec e$.

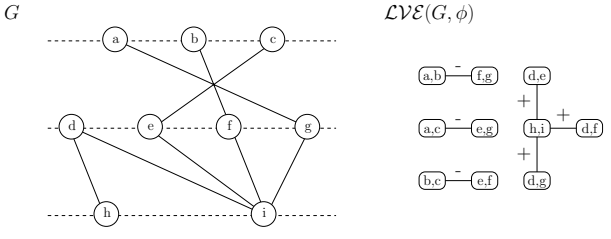

Fig. 2. An instance of the 3-cycle problem

A book-keeping solution for every such triple of vertices in the level graph has been suggested [4]. Every time we perform a ve-operation that results in a single remaining vertex being constrained by another two, we queue that vertex and its respective connected component to be traversed once we are finished with all the vertices in the current connected component. Unfortunately there are $\mathcal{O}(|V|^3)$ such triples leading to cubic running-time.

Randerath *et al.* [7] have reduced the level planarity testing and layout problems to satisfiability problems. They reduced the level planarity testing problem to a 2-SAT problem that is quadratic in the size of the leveled graph. However, if a level planar layout is required, the solution must be 'enhanced' to avoid the 3-cycle problem (or, in their terminology, to satisfy the transitivity clauses). They show that such an enhancement is always possible but they do not show how to find it within the same asymptotic running-time.

3.2 The 3-Cycle Solution

We solve the 3-cycle problem using a combination of a DFS traversal and a level-by-level traversal. In this case the DFS traversal (Algorithm 1), given an initial level layout $\mathcal{L}$, constructs a mapping π that tells us the required *relative* order of the vertices. For example, suppose $\langle u, v\rangle$ and $\langle w, x\rangle$ are in the same connected

component of the ve-graph. The algorithm may decide that $\pi(\langle u, v\rangle) = [v, u]$ and $\pi(\langle w, x\rangle) = [x, w]$. In other words, $v \prec u \Leftrightarrow x \prec w$ and $u \prec v \Leftrightarrow w \prec x$. It does not change the initial layout $\mathcal{L}$. If it returns `false` then it has found a cycle with an odd number of '-'-labeled edges (Lines 19 and 22) and the leveled graph is not level planar. If it returns `true` then it is level planar and we can proceed to the level-by-level traversal to produce a level planar layout.

Algorithm 1: `dfsTraversal`

```
   Input: VE(G, φ), L, π passed by reference
 1 visited ← {};
 2 Initialize a stack S;
 3 foreach C ∈ connectedComponents(VE(G, φ)) do
 4     Choose some vertex ⟨u, v⟩ in C;
 5     if u ≺_L v then π(⟨u, v⟩) ← [u, v];
 6     else π(⟨u, v⟩) ← [v, u];
 7     push(S, ⟨u, v⟩, false);
 8 while S not empty do
 9     ⟨u, v⟩, value ← pop(S);
10     visited(⟨u, v⟩) ← true;
11     foreach ⟨w, x⟩ ∈ neighbors(⟨u, v⟩) do
12         if λ({⟨u, v⟩, ⟨w, x⟩}) = '+' then
13             p ← [w, x], q ← [x, w];
14             if visited(⟨w, x⟩) = false then push(S, ⟨w, x⟩, value);
15         else
16             p ← [x, w], q ← [w, x];
17             if visited(⟨w, x⟩) = false then push(S, ⟨w, x⟩, ¬value);
18         if (w ≺_L x ∧ value) ∨ (x ≺_L w ∧ ¬value) then
19             if visited(⟨w, x⟩) = true ∧ π(⟨w, x⟩) ≠ p then return false;
20             π(⟨w, x⟩) ← p;
21         else
22             if visited(⟨w, x⟩) = true ∧ π(⟨w, x⟩) ≠ q then return false;
23             π(⟨w, x⟩) ← q;
24 return true;
```

The level-by-level traversal (Algorithm 2) decides on the *absolute* order of the vertices so that everything remains consistent between the connected components of the ve-graph. The vertices of the ve-graph are grouped by the level of the vertices in the leveled graph they represent. We traverse the vertices in each group $1, \ldots, k$. Within each group the vertices are traversed in descending order according to the distance between the vertices they represent in the layout $\mathcal{L}$ of the leveled graph. This traversal proceeds left-to-right and top-to-bottom along the dotted lines in Fig. 3(b). It is controlled by `rows`$(\mathcal{L})$ (the number of dotted lines), `cols`$(\mathcal{L}, i)$ (the number of vertices on the ith dotted line) and `vertexAt`$(\mathcal{L}, i, j)$ (the vertex at the jth position on the ith dotted line). Note

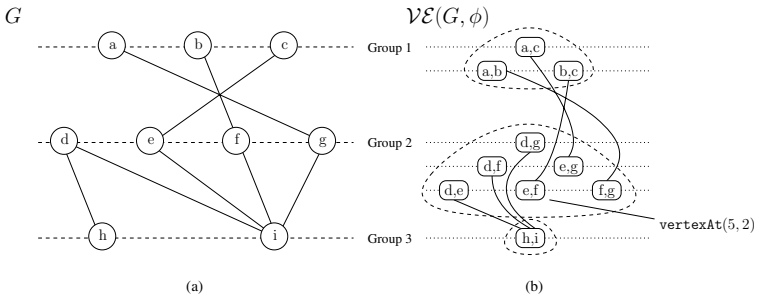

Fig. 3. (a) The initial level layout $\mathcal{L}$ of G and (b) the level-by-level traversal of $\mathcal{VE}(G,\phi)$)

that the position of some vertices may change after each update of $\mathcal{L}$. However, the traversal proceeds in this direction irrespectively.

On traversing a vertex $\langle u,v\rangle$, we count the number of neighbors it has in the previous group that are joined by '+' (`plusCnt`) and '-'-labeled edges (`minusCnt`). If $\langle u,v\rangle$ belongs to a hitherto unvisited connected component C of the ve-graph then we mark C as visited and record whether $u \prec_{\mathcal{L}} v$ or $v \prec_{\mathcal{L}} u$. If C has been previously visited and `plusCnt`+`minusCnt` $= 0$ then we use π to decide whether or not we need to exchange u and v in $\mathcal{L}$. Let $\langle w,x\rangle$ be the first vertex visited in C. Our test `match`$(\mathcal{L},\pi,visited,C,\langle u,v\rangle)$ returns `true` if any of the following are true:

- $\pi(\langle w,x\rangle) = visited(C)$ and $u \prec_{\mathcal{L}} v$ and $\pi(\langle u,v\rangle) = [u,v]$
- $\pi(\langle w,x\rangle) = visited(C)$ and $v \prec_{\mathcal{L}} u$ and $\pi(\langle u,v\rangle) = [v,u]$
- $\pi(\langle w,x\rangle) \neq visited(C)$ and $u \prec_{\mathcal{L}} v$ and $\pi(\langle u,v\rangle) = [v,u]$
- $\pi(\langle w,x\rangle) \neq visited(C)$ and $v \prec_{\mathcal{L}} u$ and $\pi(\langle u,v\rangle) = [u,v]$

and `false` otherwise. Finally, if `minusCnt` > 0 (and hence `plusCnt` $= 0$), we exchange u and v in $\mathcal{L}$ and proceed to the next vertex in the traversal. Note that it cannot arise that `minusCnt` > 0 and `plusCnt` > 0 since this would mean the presence of a cycle with an odd number of '-'-labeled edges. Algorithm 2 makes $\mathcal{L}$ level planar one level at a time. To keep its running-time linear in the size of the ve-graph we determine the label of an edge (Line 8 of Algorithm 2) dynamically instead of precomputing a lve-graph and updating the labels every time we change $\mathcal{L}$.

4 Embedding Constraints

Embedding constraints, restricting the order of incident edges around the vertices, can be specified using *constraint trees* [3]. A constraint tree $T(v)$ (see Fig. 4) is an ordered tree rooted at v where inner vertices (except the root), called *c-vertices*, impose embedding constraints on their children and the leaves are v's incident edges between v and the next successive level. There are three types of c-vertices:

Algorithm 2: `levelByLevelTraversal`

```
   Input: VE(G, φ), L passed by reference, π
 1 visited ← {};
 2 for i from 1 to rows(L) do
 3     for j from 1 to cols(L, i) do
 4         ⟨u, v⟩ ← vertexAt(L, i, j);
 5         plusCnt ← 0, minusCnt ← 0;
 6         foreach w ∈ neighbors(⟨u, v⟩) do
 7             if row(w) < i ∨ (row(w) = i ∧ col(w) <= j) then
 8                 if λ({⟨u, v⟩, w}) = '+' then plusCnt ← plusCnt + 1;
 9                 else minusCnt ← minusCnt + 1;
10         C ← connectedComponent(VE(G, φ), ⟨u, v⟩);
11         if C ∉ domain(visited) then
12             if u ≺_L v then visited(C) ← [u, v];
13             else visited(C) ← [v, u];
14         else if plusCnt + minusCnt = 0 ∧ ¬match(L, π, visited, C, ⟨u, v⟩) then
15             Exchange u and v in L;
16         if minusCnt > 0 then
17             Exchange u and v in L;
```

- *gc-vertices* (grouping) allow children to be arbitrarily permuted,
- *mc-vertices* (mirroring) allow children to be reversed, and
- *oc-vertices* (orientation) fix the order of children in $T(v)$.

$T(v)$ constrains the order of its leaves (from left to right) and thus v's incident edges between v and the next successive level. For every constraint tree $T(v)$, we expand the graph G by replacing the subgraph induced by v and its successive neighboring vertices with $T(v)$ and assigning the c-vertices to sub-levels as shown in Fig. 4. Note that this new leveling may not be proper so we add 'dummy' vertices to the long edges at the bottommost sub-level and only allow edge crossings involving the long edges to occur between this sub-level and the next successive level. This suffices since constraint trees do not share c-vertices. The ve-graph of the expanded leveled graph is constructed as before, treating c-vertices and sub-levels as regular vertices and levels respectively and with the following additions:

- A loop is added to every vertex in the ve-graph whose corresponding vertices in the leveled graph are children of the same oc-vertex.
- Every subgraph of the ve-graph induced by vertices whose corresponding vertices in the leveled graph are children of the same mc-vertex is made biconnected.

Constraining the order of v's incident edges between v and the previous level is handled analogously. When choosing an initial layout $\mathcal{L}$ of the expanded leveled graph it must satisfy the constraint trees. To begin with, the additional loops

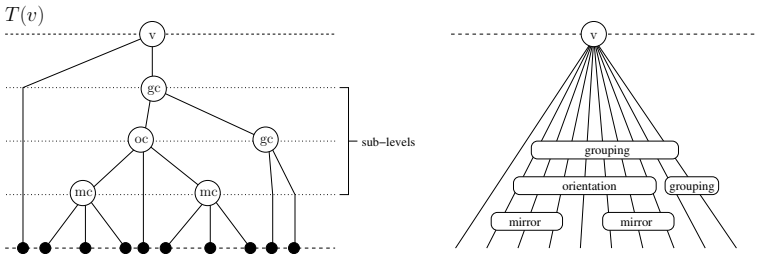

Fig. 4. A constraint tree $T(v)$ and the corresponding restriction on the order of v's incident edges between v and the next successive level

and edges are labeled '+'. The additional loops preserve the order of the children of the oc-vertices while the additional edges allow the order of the children of the mc-vertices to be reversed altogether or none at all. The level planarity testing and layout algorithms remain unchanged. If we find a level planar layout we then contract the expanded graph back to G.

References

1. Chiba, N., Nishizeki, T., Abe, S., Ozawa, T.: A Linear Algorithm for Embedding Planar Graphs using PQ-Trees. Journal of Comp. and Sys. Sci. 30(1), 54–76 (1985)
2. Di Battista, G., Nardelli, E.: Hierarchies and Planarity Theory. IEEE Trans. Sys., Man and Cybern. 18(6), 1035–1046 (1988)
3. Gutwenger, C., Klein, K., Mutzel, P.: Planarity Testing and Optimal Edge Insertion with Embedding Constraints. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 126–137. Springer, Heidelberg (2007)
4. Healy, P., Kuusik, A.: Algorithms for Multi-Level Graph Planarity Testing and Layout. Theor. Comp. Sci. 320(2-3), 331–344 (2004)
5. Heath, L., Pemmaraju, S.: Recognizing Leveled-Planar DAGs in Linear Time. In: Brandenburg, F.J. (ed.) GD 1995. LNCS, vol. 1027, pp. 300–311. Springer, Heidelberg (1996)
6. Jünger, M., Leipert, S.: Level Planar Embedding in Linear Time. Journal of Graph Alg. and App. 6(1), 67–113 (2002)
7. Randerath, B., Speckenmeyer, E., Boros, E., Hammer, P., Kogan, A., Makino, K., Simeone, B., Cepek, O.: A Satisfiability Formulation of Problems on Level Graphs. Technical report 40-2001, Rutgers Center for Operations Research, Rutgers University (2001)