A Path 3-List Coloring Algorithm for Plane Graphs

April 20, 2016

Abstract

We present an algorithm to path 3-list-color plane graphs based on the work by Skrekovski [2] and Hartman [1]. Furthermore, we describe an implementation in linear time.

Implementation

The following table describes the variables and data structures required for our implementation. Note that the *plane_graph* structure is a double adjacency graph, which is similar to a standard adjacency graph but allows constant time backwards traversal. An embedded double adjacency graph can be constructed from a plane graph in linear time.

variable	**data structure**	**interpretation**
plane_graph	vertex array of vertex index pair arrays	double adjacency list graph in embedded order
color_lists	vertex array of color lists	color lists assigned to each vertex
color	vertex array of colors	colors assigned to each vertex
state	vertex array of states	state of each vertex
face_location	vertex array of integers	id of the outer face segment each vertex belongs too
face_location_sets	disjoint set structure of integers	outer face segment ids that may be unioned together
neighbor_range	vertex array of index pairs	range of neighbors in current subgraph

Table 1: Variables

Algorithm 1: *hartman_skrekovski*

```
Input: vertices x, y, p, integers c_xp, c_py, c_yx
assign_color(p, color_list[p].first)
x′ ⟵ x, y′ ⟵ y
for i_p ∈ neighbor_range[p] do
    (n, i_n) ⟵ plane_graph[p][i_p]
    if state[n] = interior then
        initialize(n, i_n, c_xp)
        remove color[p] from color_list[n], remove first from neighbor_range[n]
    else if i_p is the first in neighbor_range[p] then
        if i_p is the last in neighbor_range[p] then
            if n ≠ x ∧ n ≠ y then
                remove color[p] from color_list[n]
            assign_color(n, color_list[n].first), stop
        else if n = y then
            assign_color(n, color[p])
            if x = p then
                x′ ⟵ n
            hartman_skrekovski(x′, p, y, −1, −1, c_py), stop
        else
            if x = p then
                x′ ⟵ n
                set_face_location(n, c_xp)
            remove color[p] from color_list[n], remove last from neighbor_range[n]
    else
        (b_p, e_p) ⟵ neighbor_range[p], (b_n, e_n) ⟵ neighbor_range[n]
        if p = y then
            remove color[p] from color_list[n]
            if p = x then
                neighbor_range[n] ⟵ (i_n, e_n − 1)
                hartman_skrekovski(x′, n, x′, −1, c_xp, c_py)
                if i_n ≠ b_n then
                    neighbor_range[n] ⟵ (b_n, i_n), neighbor_range[p] ⟵ (i_p, e_p)
                    hartman_skrekovski(n, p, p, −1, −1, c_py)
                stop
            else if i_p = e_p then
                assign_face_location(n, c_xp), neighbor_range[n] ⟵ (i_0, e_n − 1)
                hartman_skrekovski(x′, n, x′, −1, c_xp, c_yx), stop
            y′ ⟵ n
        if n = y ∨ compare(face_location[n], c_py) then
            p′ ⟵ n
            if p ≠ y ∧ color[p] in color_list[n] then
                assign_color(n, color[p])
            else if state[n] ≠ colored ∨ color[n] ≠ color[p] then
                p′ ⟵ x′, c_py ⟵ union(c_xp, c_py), c_xp ⟵ −1
            neighbor_range[n] ⟵ (i_n, e_n − 1)
            hartman_skrekovski(x′, y′, p′, c_xp, c_py, c_yx)
            if i_n ≠ b_n then
                neighbor_range[n] ⟵ (b_n, i_n), neighbor_range[p] ⟵ (i_p, e_p)
                hartman_skrekovski(p, n, p, −1, c_py, −1)
        else if compare(face_location[n], c_yx) then
            remove color[p] from color_list[n]
            neighbor_range[n] ⟵ (i_n, e_n − 1)
            hartman_skrekovski(x′, n, x′, −1, c_xp, c_yx)
            neighbor_range[n] ⟵ (b_n, i_n), neighbor_range[p] ⟵ (i_p, e_p)
            hartman_skrekovski(n, y, p, −1, c_py, c_yx)
        else
            remove color[p] from color_list[n]
            neighbor_range[n] ⟵ (b_n, i_n), neighbor_range[p] ⟵ (i_p, e_p)
            hartman_skrekovski(x, y, p, c_xp, c_py, c_yx)
            neighbor_range[n] ⟵ (i_n, e_n − 1)
            hartman_skrekovski(n, n, n, −1, c_xp, −1)
        stop
```