A new heuristic for rectilinear crossing minimization

François Doré and Enrico Formenti

Université Côte d'Azur, CNRS, I3S, France

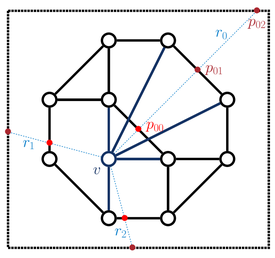
Objectives

The problem of graph design has been studied for more than sixty years. Several criteria are commonly accepted as characterizing a good drawing, such as the angle resolution, the distribution of vertices in the plane and the number of edge crossings. We focus on the last one. Due to the complexity of the rectilinear crossing number problem, one expects to describe only a heuristic. Ours is based on a greedy approach with a system of rays casted from some vertices allowing to move them to better locations in the drawing. Rays can either pass through an edge or bounce in a billiard-like manner. Indeed, the main idea of the algo comes from a classical result a discrete dynamical systems which says that in a square billiard, balls shot along irrational angles visit any open subset of the billiard. In our context, this implies that our solution can be as close as desired to the optimum.

The algorithm

Step 1: ray casting

To move a vertex v to a better location, we start by casting R rays from v. For each ray, we compute all the intersections with the edges of the graph G plus four arbitrary false edges which delimits the bounding box of G. We keep the closest one and go to the second step.

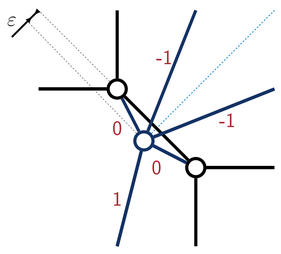


For example, if $R = 3$ and the node v is considerd. For r_0, all the intersection points with G or the 4 added edges (dotted lines), are p_{00}, p_{01} and p_{02}. The only point considered for the next step for this ray is p_{00}, which is the first one met starting from v.

Step 2: edge opacity

When a ray, cast from a node v is about to cross an edge e, a value is attributed to e to determine if the ray goes through the edge or if it reflects. We call this value *edge opacity* and denote it by $Op(v, e)$. To compute it, we start by assigning to each edge $e_i \in E(v)$ a weight w_{e_i} as follows:

$$w_{e_i} = \begin{cases} 0, & \text{if } e_i \text{ shares a vertex with } e \\ -1, & \text{if } e_i \text{ does not share a vertex with } e \text{ and } e_i \text{ crosses } e \\ 1, & \text{if } e_i \text{ does not share a vertex with } e \text{ and } e_i \text{ does not cross } e \end{cases}$$



After we computed the weights, the *opacity* of e according to v is defined as

$$Op(v, e) = \begin{cases} \frac{\sum_{e_i \in E(v)} w_{e_i}}{|E^*(v)|}, & \text{if } E^*(v) \neq \varnothing \\ 0, & \text{otherwise} \end{cases}$$

where $E^*(v)$ denotes the set of non-null edges.

Step 3: crossing the edge

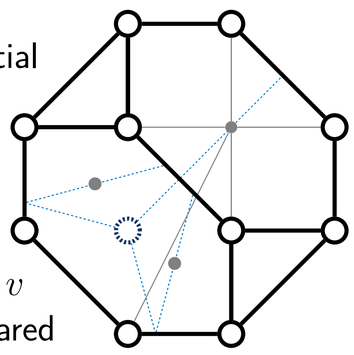
Two different strategies for the reflection or crossing of an edge, deterministic or randomized. The ray reflects when it crosses a false edge or when one of the following conditions are satisfied:

Deterministic
$O_p(v, e) \leq 0$

Randomized
$\chi > O_p(v, e)$
with χ a uniformly distributed random variable in $[-1-\varepsilon, 1+\varepsilon]$

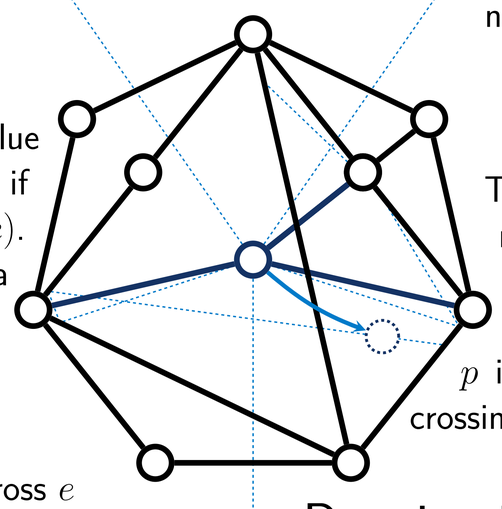
Step 4: choosing the location

A *ray* r is a sequence $p_0, p_1, \ldots, p_{n_r}$, where p_0 is the intial position of v and p_i, $i > 0$ are the intersections or the reflection points of r with the edges of G. For each ray, we define a final point p as the middle of the last ray segment, *i.e.* $p_{n_r-1}p_{n_r}$. Two criteria are used to find the best one among all the rays: first, the number of crossings of v evaluated on the position p; second, the sum of the squared norms of the Hooke's law forces applied to the endpoints of the $e_i \in E(v)$ where v is, again, evaluated on p.



Complexity

To move one node v the algorithm runs in $O(kERn_r)$, where E is the number of edges and k is the degree of v. Moreover, at each iteration, it runs through the nodes until one can be moved to a better location. This leads to a total complexity of $O(E^2Rn_r)$ per iteration.



Results

Dataset

We evaluated our algorithm on 100 randomly selected graphs per class provided on `www.graphdrawing.org/data.html`

Parametrization

The refinement of the parameters offers a compromise between efficiency and time consumption. Our algorithm depends on several parameters: R and n_r the number of rays and ray segments, $\mathbb{P}_\chi$ the distribution law of χ and two more parameters Δe and w defined below. These parametes does not change the functioning of the algorithm but its behavior, namely the way it convergers.

Energy delta

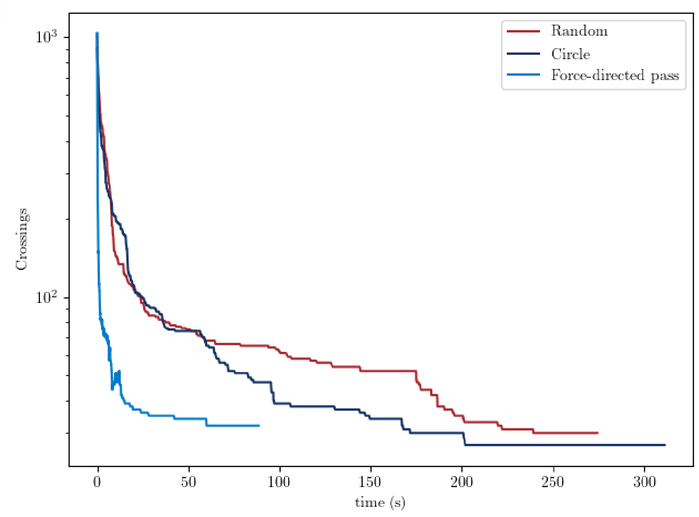
The parameter Δe is the amount of energy decrease necessary for a vertex v to move to another location p if p does not improve the crossing number.

Prohibition window

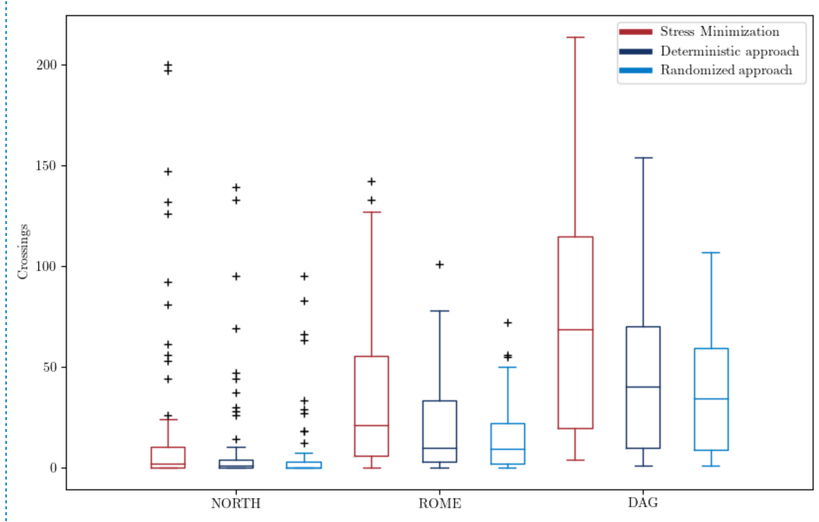
The parameter w defines, for each vertex v that has been moved, the number of iterations required before it can be moved again. It is expressed as a ratio of the total number of vertex of G.

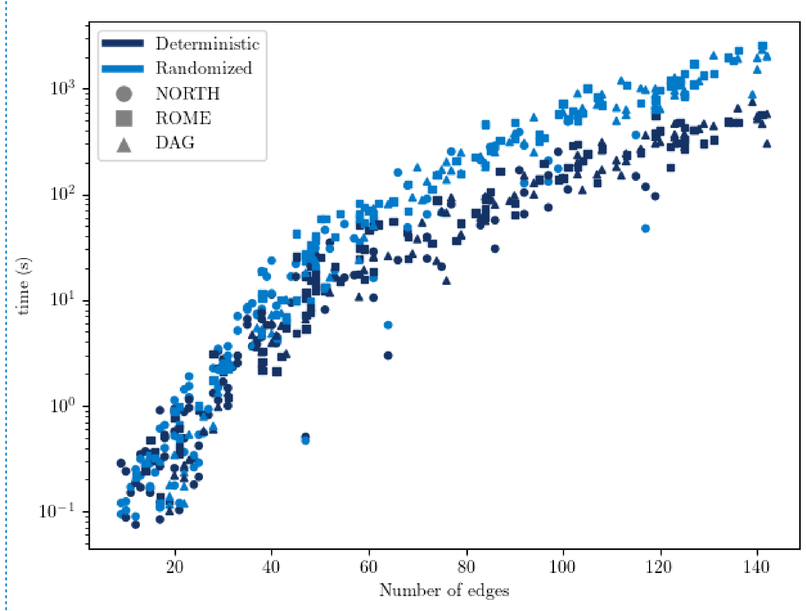
Drawing initialization

For each initial configuration: random initialization, all vertices on a circle, or the application of attraction and repulsion forces, our algorithm finds a similar result, only the time differs. We will keep the fastest one.



Experiments





Both the deterministic and randomized approaches significantly improve the results of the best rectilinear drawing algorithm present in OGDF and implemented in *Tulip*, namely, Stress Minimization, with a slight advantage for the second one. The randomized heuristic needs notably more time to converge than the deterministic one as it allows itself to visit less straightforward paths among the possible drawings to find minima. Anyway, more experiments are needed to assess the viability of the two heuristics. Comparisons with similar approaches are also planned.