Testing Bipartiteness of Geometric Intersection Graphs

David Eppstein

Univ. of California, Irvine
School of Information and Computer Science

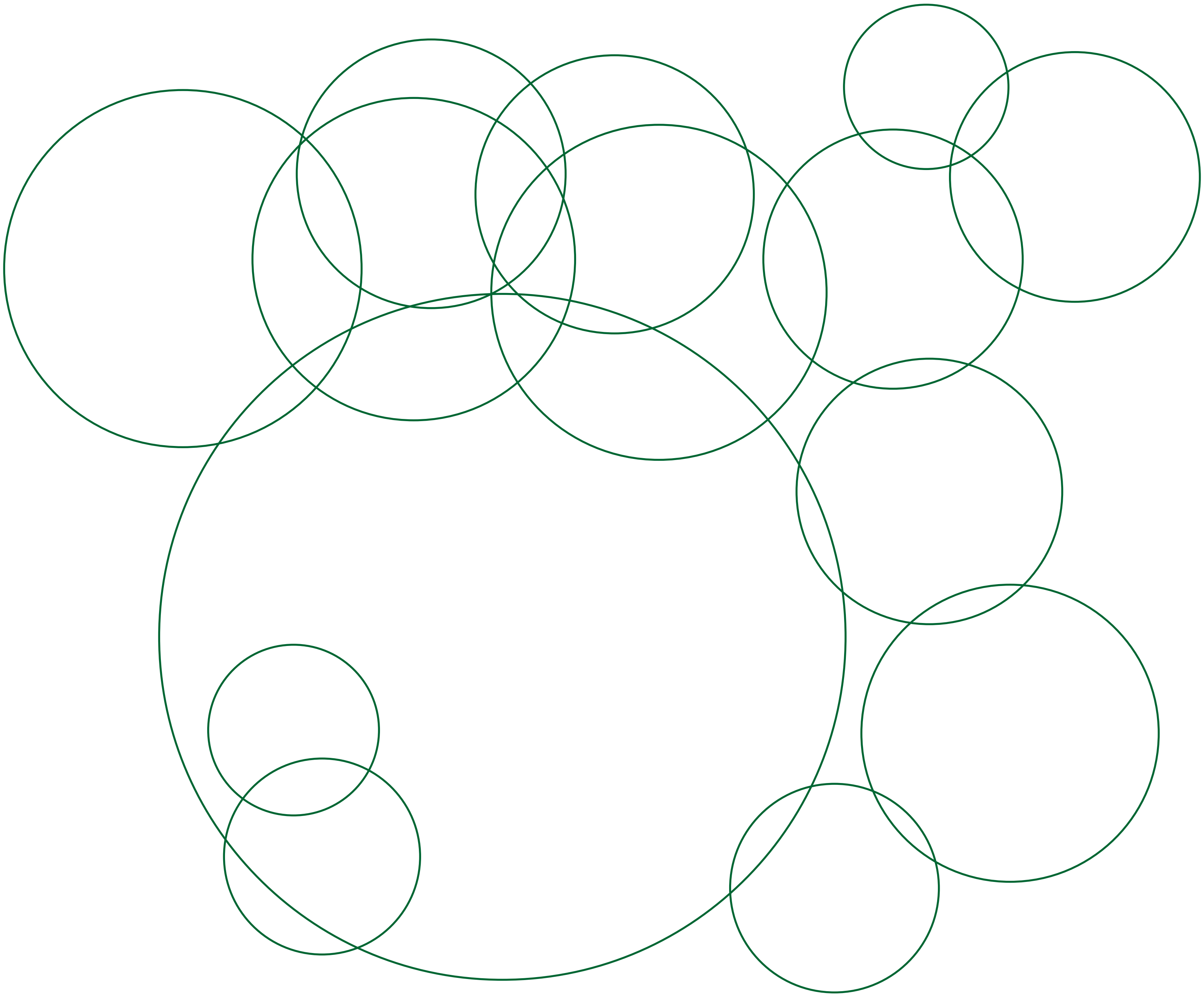
Intersection Graphs

Given arrangement of geometric objects, form undirected graph:

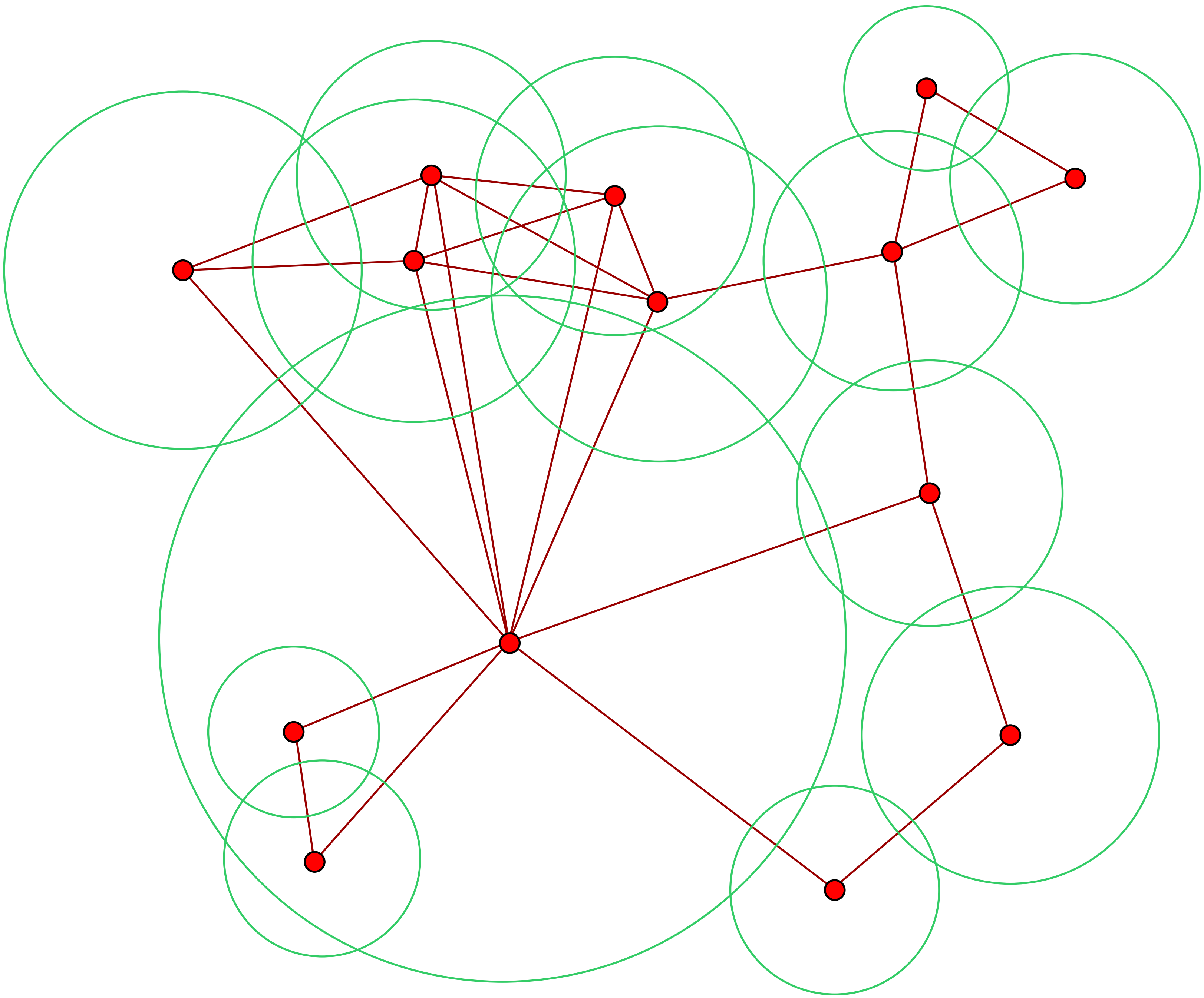
Vertex for each object

Edge for each intersecting pair of objects

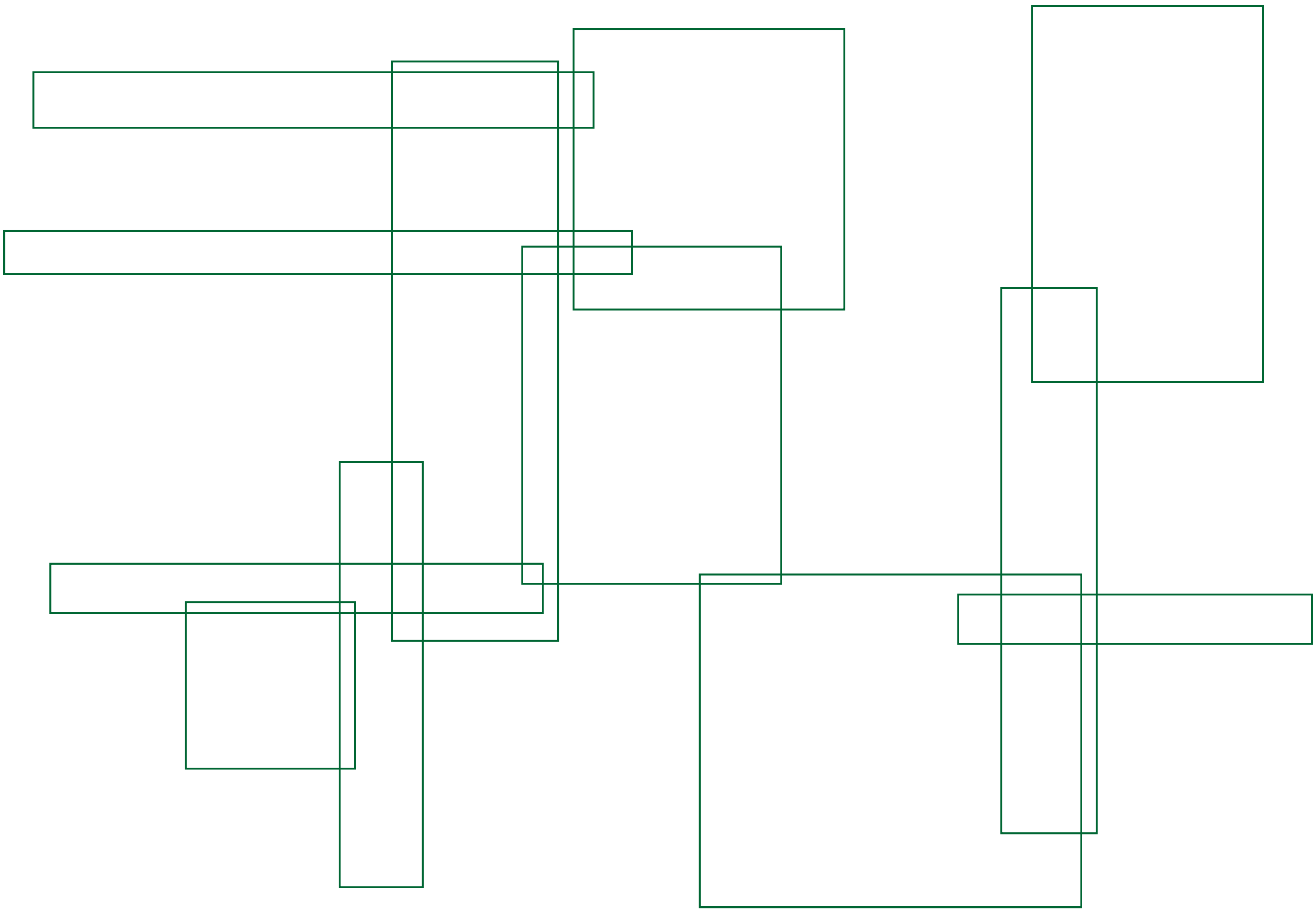
Arrangement of circles



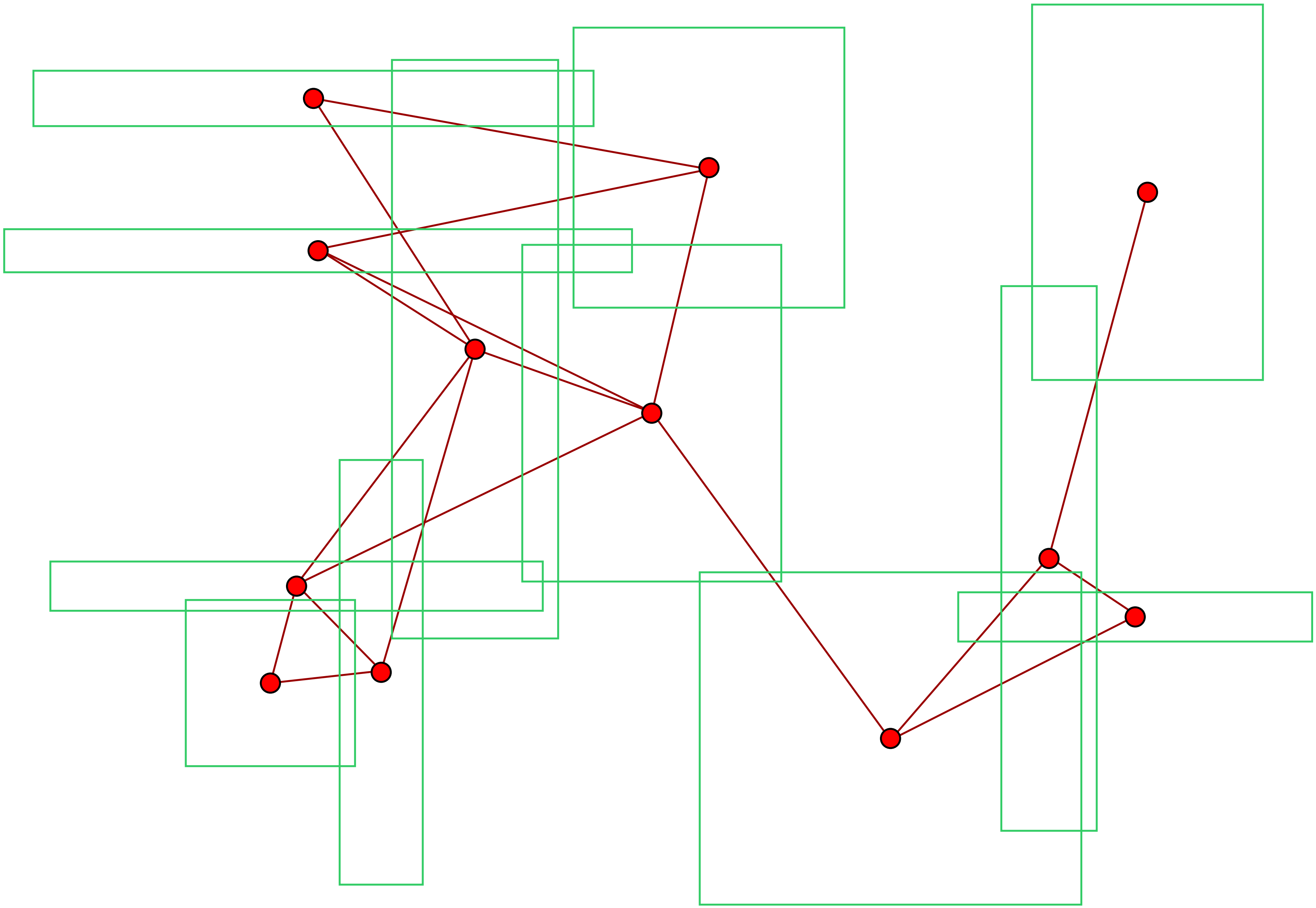
Intersection graph of circles



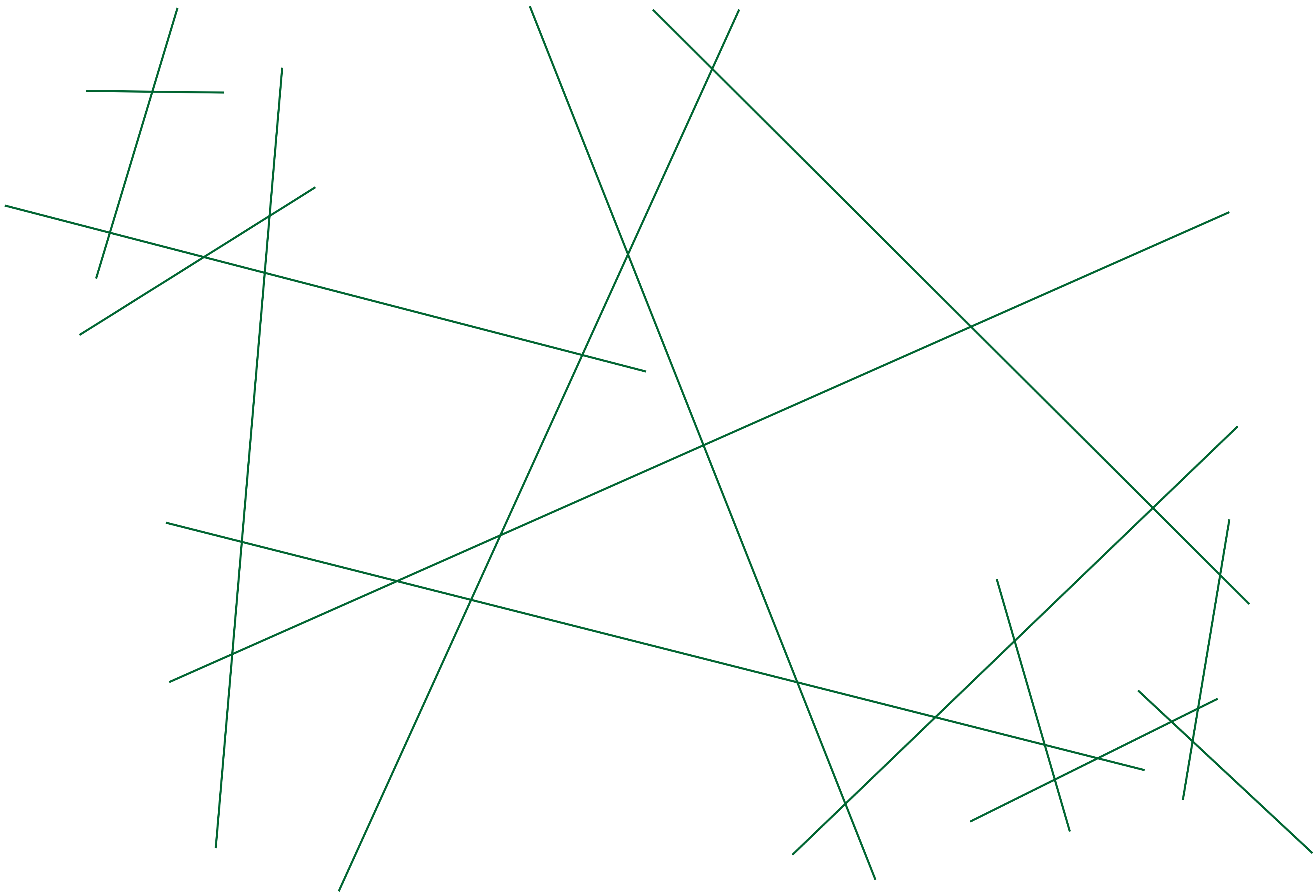
Arrangement of rectangles



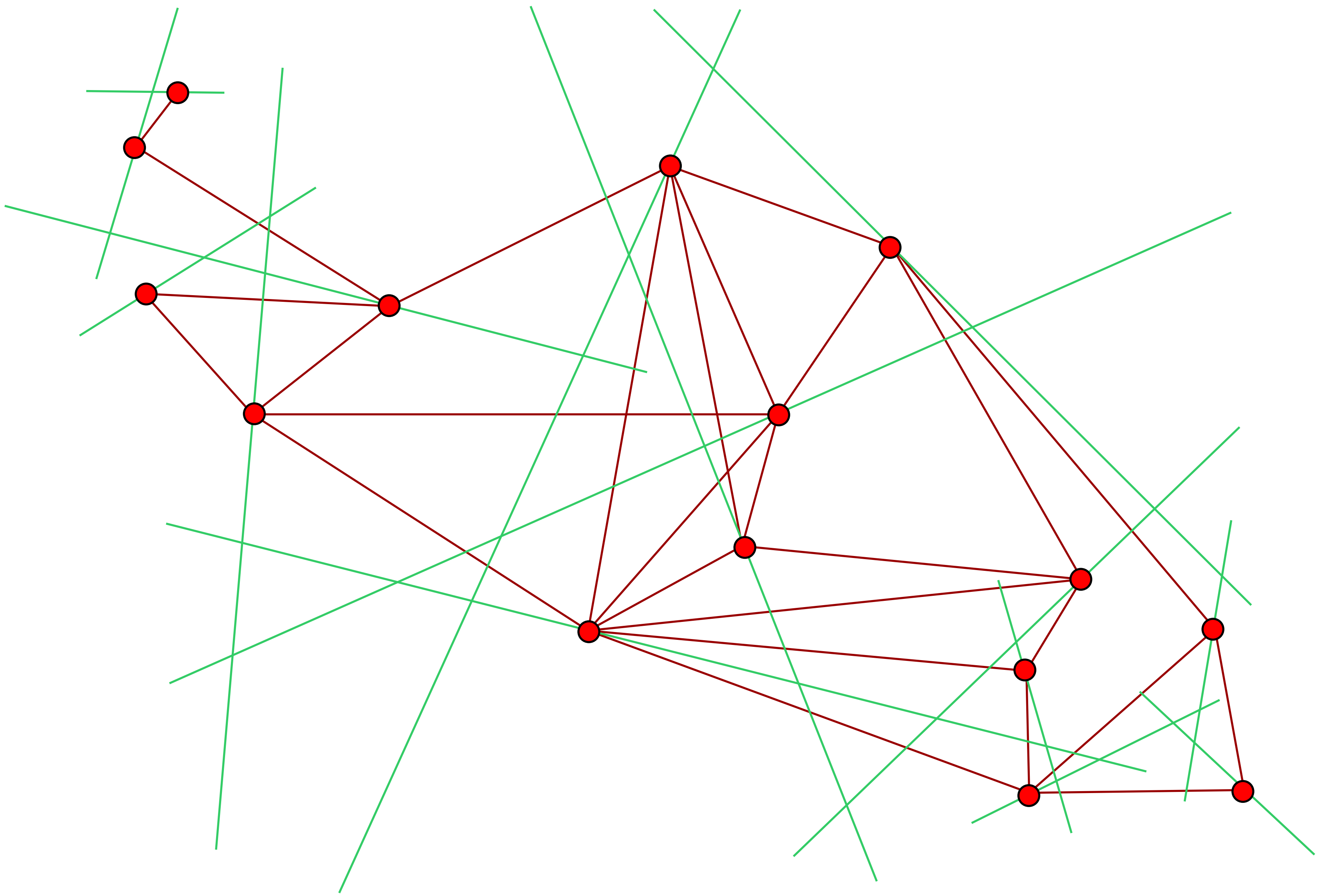
Intersection graph of rectangles



Arrangement of line segments



Intersection graph of line segments



Graph algorithm problems on intersection graphs

e.g. connectivity, diameter, coloring...

Obvious approach:

Construct arrangement graph

Apply non-geometric graph algorithm

Intersection graphs can have large complete subgraphs

So, obvious approach requires $\Omega(n^2)$ time

For which problems can we do better?

New results:

Geometric objects of bounded description complexity

Spanning forest: $O(n^{2-\varepsilon})$, ε depends on object type

(standard geometric range searching data structure techniques)

Bipartiteness: same upper bound

Spherical balls in $\mathbf{R}^d$

Spanning forest: $\Omega(n^{(2d-2)/d})$

(unless Euclidean minimum spanning trees can be solved more quickly)

Bipartiteness: $O(n \log n)$

Line segments or polygons in $\mathbf{R}^2$

Spanning forest: $\Omega(n^{4/3})$

(unless Hopcroft's problem on point-line intersection can be solved more quickly)

Bipartiteness: $O(n \log n)$

Bipartiteness is easier than spanning forest construction!

Why consider intersection graph bipartiteness?

Color intersection graph of balls in R^2:

channel assignment in ad hoc networks

Color intersection graph of line segments in R^2:

speed up geometric data structures

(e.g. nearest neighbor queries O(log n) for O(1)-colored arrangement)

partition graph drawing into planar layers

(find geometric thickness)

3-coloring is NP-complete in both cases,

so 2-coloring is last polynomial-time-solvable case

All our algorithms find either a 2-coloring or an obstacle (odd cycle)

Lower bound for connectivity of spheres

Connected components of intersection graph of unit spheres

Bichromatic closest pair decision problem

[Chan's randomized optimization technique]

Bichromatic closest pair

[Agarwal, Edelsbrunner, Schwarzkopf, & Welzl]

Euclidean minimum spanning tree construction

Upper bound for bipartiteness of spheres

Key idea [Teng]:

Either some point in the plane is covered by many spheres
or the sphere intersection graph has small separators

Algorithm:

try:

use separator-based divide-and-conquer
to construct intersection graph in O(n log n) time
[Eppstein, Miller, & Teng]

apply general graph bipartiteness algorithm

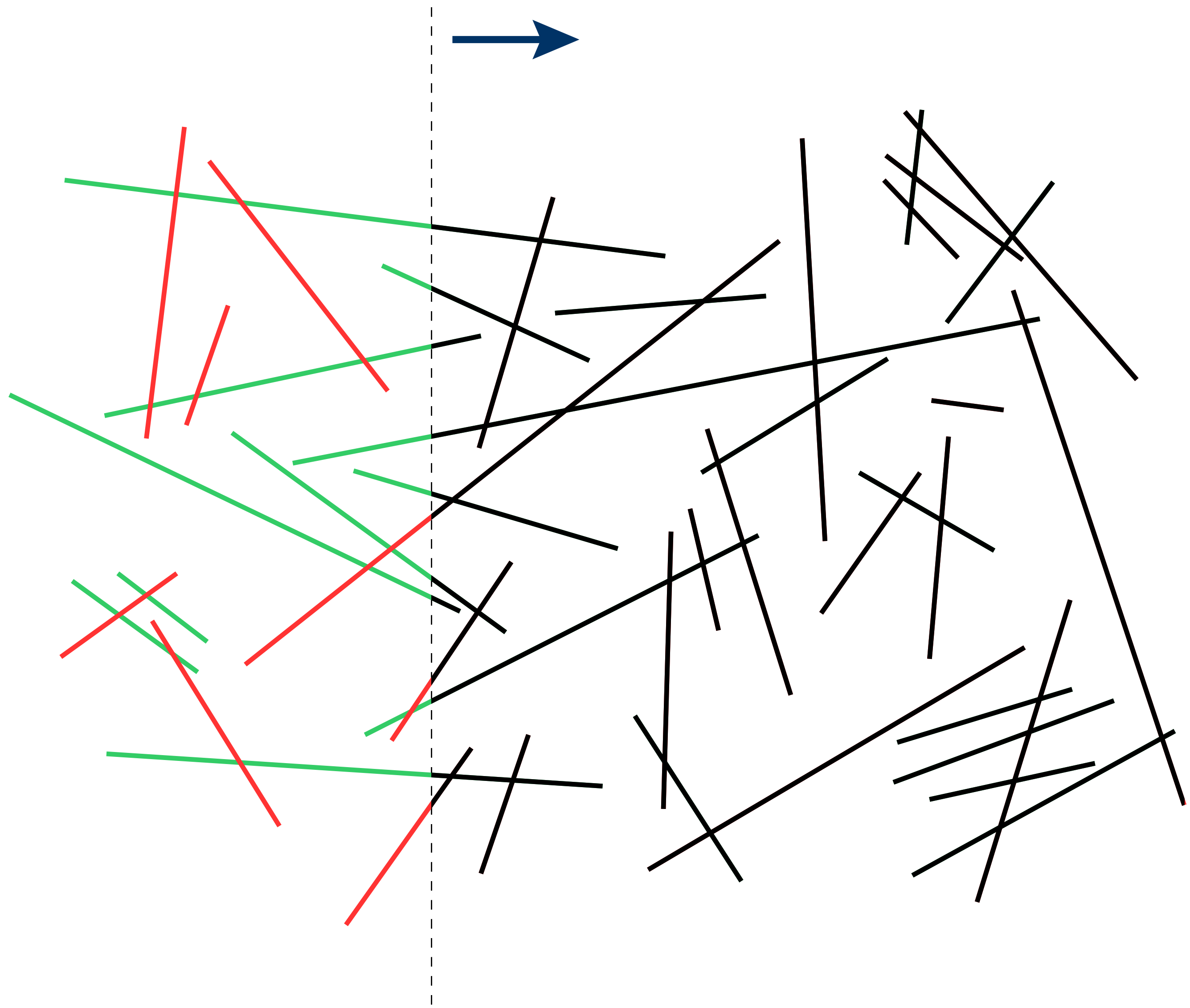
except if densely covered region prevents finding a separator:

use geometric prune-and-search to find
point covered by three or more spheres

return triangle in intersection graph

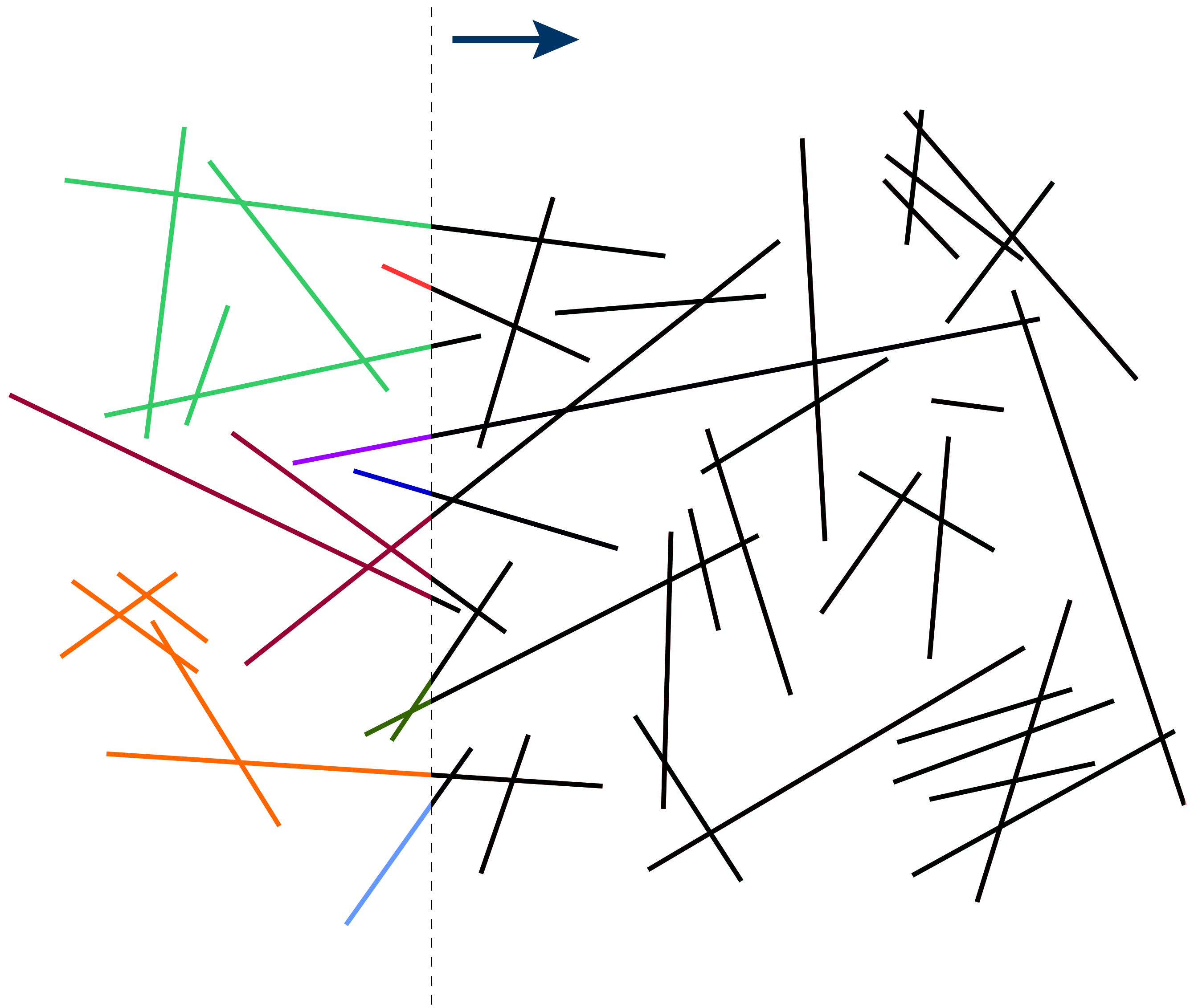
Upper bound for bipartiteness of segments: main idea

Sweep vertical line left-right across arrangement
Maintain 2-coloring of halfplane to left of sweep line



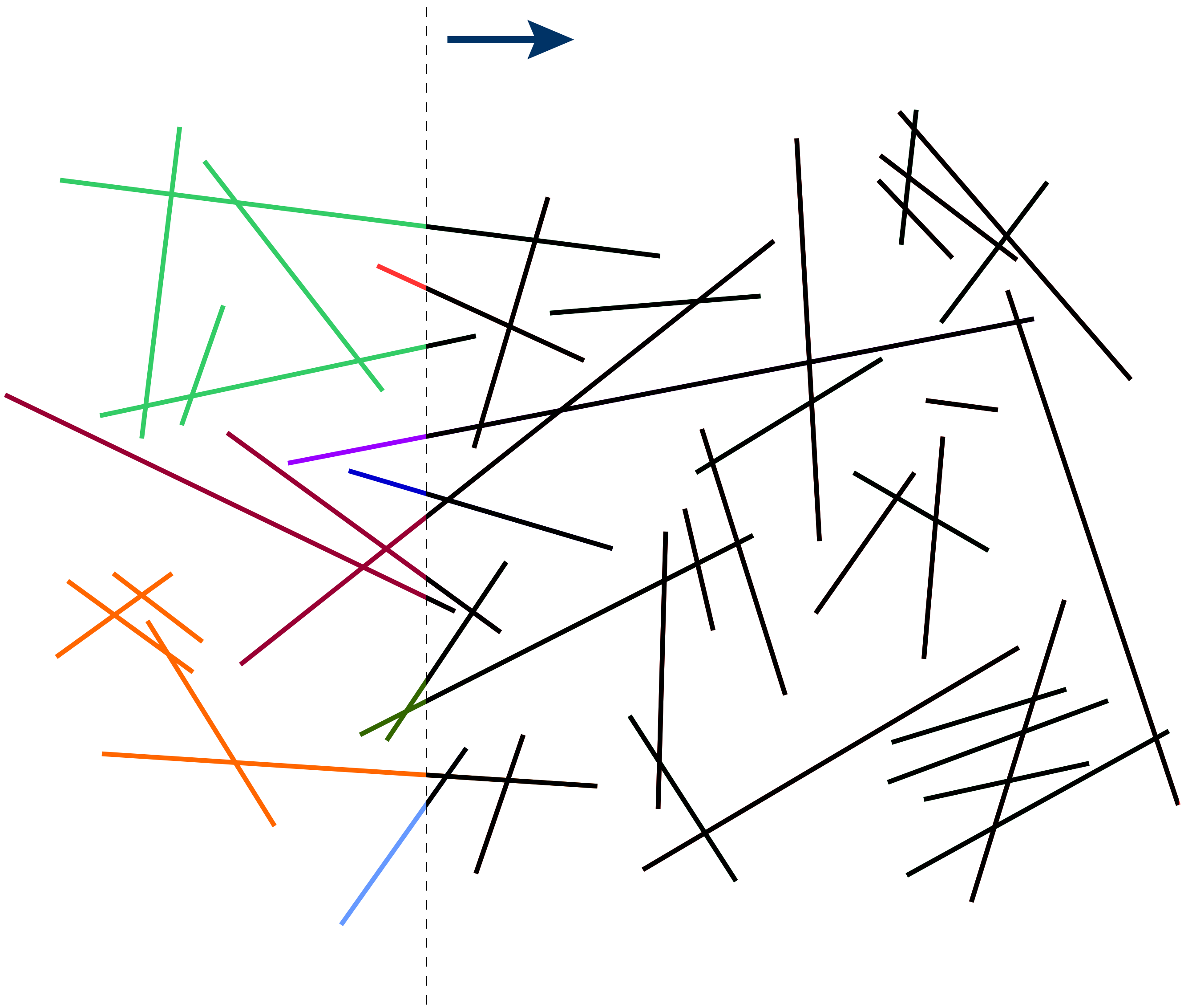
Upper bound for bipartiteness of segments: key objects

Component: subset of segments connected by crossings left of sweep line



Upper bound for bipartiteness of segments: key objects

Component: subset of segments connected by crossings left of sweep line
Bundle: segments from single component crossing sweep line consecutively



Upper bound for bipartiteness of segments: data structures

Spanning tree of each component
(Sleator-Tarjan dynamic tree, can look up parity of paths)

Top and bottom edge of each color in each bundle

Priority queue of potential sweep events
(line segment endpoints and selected crossings)

"Bundle tree": binary search tree with one node per bundle
(ordered by crossing sequence with sweep line)

"Color trees": binary search trees for segments of each color in each component
(must allow merge and split operations;
also ordered by crossing sequence with sweep line)

Upper bound for bipartiteness of segments: events

Left endpoint of segment: create new component and bundle
(possibly splitting existing bundle in two)

Right endpoint of segment: remove from bundle
(possibly changing top/bottom edge, or
removing whole bundle, merging two other bundles)

Crossing between two adjacent edges in same color tree:
graph is not bipartite, abort algorithm

Crossing between top edge of one bundle and bottom edge of another:
merge bundles and components
(possibly merging other pairs of bundles from same two components)

Upper bound for bipartiteness of segments: analysis

O(n) events from endpoints of segments

O(n) bundle-merging events
(because O(n) bundles created by other events)

At most one odd cycle event

All other arrangement crossings are ignored

Each event causes O(1) updates to data structures
(including priority queue of potential future events)
and can be handled in O(log n) time

Conclusions and open problems

New algorithms for bipartiteness of intersection graphs

Evidence that bipartiteness is faster than connectivity

Other intersection graphs?
(plane sweep handles most 2d cases; 3d?)

Other natural graph problems?

Dynamic? [Hershberger-Suri '99: connectivity of rectangles]