Drawing of sparse graphs via hexagonal tilings and iterative linear assignment

Carl Kingsford[1]

Carnegie Mellon University, Pittsburgh PA 15213, USA,
carlk@cs.cmu.edu,
WWW home page: http://www.cs.cmu.edu/~ckingsf

Abstract. We present a method for drawing graphs that assigns nodes to tiles in a hexagonal planar tiling. Adjacencies of the tiles are used to represent node adjacencies in order to reduce the number of explicit edges that must be drawn. We show how an alternative method of drawing the tiles can be used to relax the requirement that nodes assigned to touching tiles be adjacent in the input graph. We present an approach to finding a layout that attempts to minimize the number of explicit edges drawn and to maximize other aesthetic qualities. The approach is based on the combination of several heuristics, the most central of which is the repeated solution of a maximum weighted bipartite matching to find a good assignment of nodes to tiles. The technique is shown to produce compact, readable drawings for sparse graphs.

1 Introduction

Drawing edges so that they can be easily tracked, so that adjacencies in a graph can be quickly seen, and so that edges do not occlude nodes or each other are central challenges in graph drawing. Various solutions to this problem have been developed (e.g., minimizing edge crossings [5] and using edge bundling [8], to name just two of many). Here, we consider an approach that represents some edges as adjacencies between tiles in a hexagonal, planar tiling. Because it is not always possible to embed a graph into the tiling, the resulting drawings will contain a mixture of edges represented by tile adjacencies and those represented by curved line segments.

Formally, we investigate the following problem:

Problem 1. Given a graph $G = (V, E)$ and a hexagonal tiling $\mathcal{T} = (C, F)$, where C are tiles and $F \subseteq C \times C$ are adjacencies between tiles, find a function $f : V \mapsto C$ such that as many edges in E as possible are mapped to adjacent tiles in F. In other words, find f to maximize $|\{\{f(u), f(v)\} \in F : \{u, v\} \in E\}|$.

Another way to view Problem 1 is as a maximum graph homomorphism problem restricted to the case of a hexagonal tiling. Consider $\mathcal{T}$ to be the graph representing adjacencies of tiles. We seek a mapping f from the nodes of G to the nodes of $\mathcal{T}$ so that for as many edges $\{u, v\} \in E$ as possible, we have $\{f(u), f(v)\} \in F$.

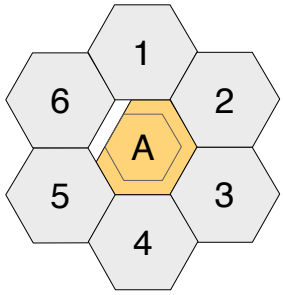

Fig. 1. Hexagon tiling showing that the adjacency between any tile A and any number of its neighbors (here, just neighbor 6) can be removed by pulling an edge of the hexagon inward. This change does not affect any of the other tile adjacencies.

Notice we do not require that $\{f(u), f(v)\} \in \mathcal{T} \implies \{u, v\} \in G$. This is because hexagonal tilings have the property that any side of the hexagonal tile can be moved inward independently without affecting any other adjacencies of the tile (Figure 1). This allows any non-adjacent nodes to be assigned to adjacent tiles without introducing false edges, and permits the creation of very compact drawings. While Problem 1 could be defined for any planar tiling, this property of hexagonal tiling (and other tilings) makes it particularly attractive. They are also attractive because they are symmetric, fill the plane, and each tile has a relatively large number of adjacencies. Further, common motifs such as cliques, bicliques, and stars show up directly in the layout, and nodes with high clustering coefficient can also be seen as they will tend to have adjacent tiles that are also adjacent.

Figure 2 provides an example of the type of drawing we will produce. These drawings reduce clutter by reducing the number of edges that must be drawn and are intuitively readable. They are particularly suited to situations when local adjacencies are of more interest than global structure.

To solve Problem 1 without placing restrictions on the graph G to be drawn, we develop an EM-like algorithm that iteratively solves linear assignment instances (maximum bipartite matching) to refine a mapping f from vertices to tiles. A randomized approach that is used to avoid becoming trapped in local optima is presented and tested as well. We also test an approach to edge drawing and coloring for edges that are not handled by tile adjacencies.

Related work. The tessellation representation of planar graphs [14] is similar to the tiling approach described above in that it omits directly drawing edges, though it is restricted to planar graphs, requires nodes to be represented by rectangles, and — arguably — is not as intuitive to read. Planar, 4-regular, graphs can be drawn by associating nodes with the intersections and touching points of circles the arcs of the circles as edges [12]. The visibility approach [13, 2] to drawing graphs provides another alternative in which edges can be implicitly represented. Again, this approach has been restricted to planar graphs, and it can

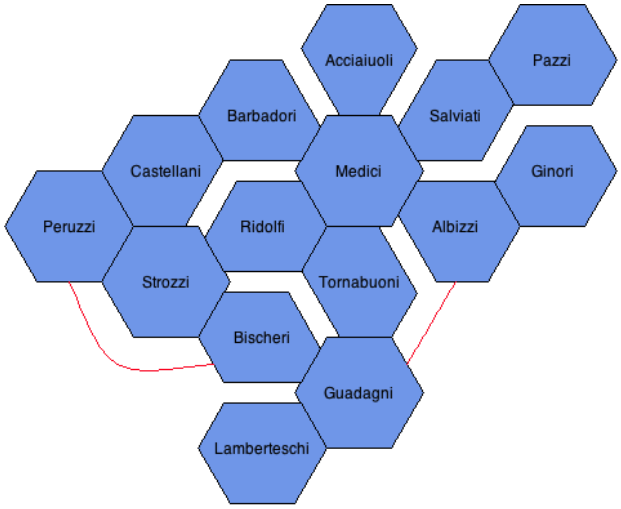

Fig. 2. The Florentine families graph ($n = 15$, $m = 20$) from [1].

produce drawings that are inorganic or unintuitive to read. A similar iterative, linear-assignment-based algorithm was given in [15] for the different problem of finding good orderings of the rows and columns of a matrix.

This paper is not a theoretical paper in the sense that we give any bounds or guarantees. Instead a number of algorithmic ideas are combined to produce a practical heuristic for laying out graphs using the tiling scheme described above. We show that in combination, these ideas can be applied to efficiently produce hexagonal tilings for non-planar, sparse graphs that are high-quality under several metrics such as small area, limited node-edge crossings, and a large number of edges implicitly represented.

2 Hexagonal Tiling Layout Approach

2.1 Iterative, randomized linear assignment for assigning nodes to tiles

Iterative linear assignment. To find an assignment that uses tile adjacency to represent as many edges as possible, we introduce an EM-like, randomized procedure that iteratively improves an initial assignment. This procedure, shown in Algorithm 1, computes the benefit of moving each node u to each tile assuming that all other nodes are fixed in their current assignment. This results in a weighted, bipartite graph relating nodes to tiles, and a maximum weight matching [9] is found in this graph to choose a new assignment based on these weights. Algorithm 1 gives an overview of the procedure, and more detail on the various steps is given in the sections below.

Algorithm 1 Iterative, randomized linear assignment heuristic to solve Problem 1.

1: Initialize σ_0.
2: $f'' \leftarrow \textsc{InitialAssignment}(G, \mathcal{T})$
3: **repeat**
4: $\quad f \leftarrow f''$
5: $\quad M_G(f, \sigma_t) \leftarrow \textsc{CreateMatchGraph}(G, \mathcal{T}, f, \sigma_t)$
6: $\quad f' \leftarrow \textsc{MaximumWeightMatching}(M_G(f, \sigma_t))$
7: $\quad f'' \leftarrow \textsc{PlaceUnplacedNodes}(G, \mathcal{T}, f')$
8: $\quad \sigma_{t+1} \leftarrow \sigma_t + \Delta\sigma$
9: $\quad t \leftarrow t + 1$
10: **until** $f'' = f$
11: $f \leftarrow \textsc{FloatIslands}(G, \mathcal{T}, f)$
12: $\textsc{RouteUnsatisfiedEdges}(G, \mathcal{T}, f)$

Constructing a matching graph. Let $G = (V, E)$ be the graph to draw and let $f : V \to C$ be a 1-to-1 function from its nodes to the tiles C of a planar tiling. We define a weighted bipartite graph $M_G(f) = (V \cup C, A, w)$ with edge set A and their weights w as follows.

Let $\mathcal{N}_{tile}(c)$ be the set of tiles that are adjacent to c. A tile c is *blocked* for a node u if there is some existing assignment to an adjacent tile of another node that is not an neighbor of u. In other words, a tile c is blocked for u if there is a $v \in V$ such that $f(v) \in \mathcal{N}_{tile}(c)$ but $\{u, v\}$ is not an edge of G. Similarly, we say a tile is *attractive* for u if there is at least one $v \in V$ such that $f(v) \in \mathcal{N}_{tile}(c)$ and $\{u, v\} \in E$. Such edges $\{u, v\}$ would be *satisfied* by the assignment $f(u) = c$. Define the satisfaction number $S_f(u, c)$ to be the number of satisfied edges incident to u if u were assigned to tile c.

We add edge (u, c) to M_G if either c is not blocked for u or c is attractive for u (line 5 in Algorithm 1). The benefit of edge (u, c) is set to

$$w(u, c) = r \times S_f(u, c) + \sum_{v \in \mathcal{N}_G(u)} d(c, f(v)), \tag{1}$$

where r is a parameter specifying the reward for satisfying an edge, $\mathcal{N}_G(u)$ are the neighbors of u in G, and $d(c, d)$ is a function that is decreasing as the planar distance between the tiles c and d increases. This choice for $w(u, c)$ rewards both satisfying edges and placing the endpoints of unsatisfied edges nearby (because $d(c, d)$ will be high). Finally, it also implicitly makes high-degree nodes more important because the summation for high-degree nodes will contain more terms. Equation 1 is but one choice of possible weight function, and the algorithms below work for any positive-valued function, though the quality of the layouts may differ. If the edges $\{u, v\}$ of G are weighted by their importance $i(u, v)$, the first term of (1) can be changed to include benefit $ri(u, v)$ for each satisfied edge. For the experiments reported here, we take $r = 1000$ and $d(c, d) = 500/h_{cd}$ where h_{cd} is a Euclidean distance between the centers of tiles c and d divided by the hexagon radius.

Avoiding poor local minima. The iterative linear assignment procedure described above often gets stuck in poor local minima or it oscillates between two poor solutions. To solve this, we add an inertia effect that randomly keeps a subset of nodes in their current tiles. Specifically, let σ_t be a real number in $(0, 1]$. On each iteration, for every u, with probability σ_t, instead of adding the edges to M_G described above, we add the single edge $(u, f(u))$ with weight r. This strongly encourages a subset of expected size $\sigma_t|V|$ of nodes to remain fixed; we call such nodes *inertia nodes*, and we denote the resulting bipartite graph $M_G(f, \sigma_t)$. For the experiments here, we start with $\sigma_0 = 1/200$ and increase it by $\Delta\sigma = 1/400$ each iteration.

Handling unmatched nodes. Because $M_G(f, \sigma_t)$ is not complete and f must be 1-to-1, the maximum weight matching may not match every node to a tile. This is particularly likely to happen with nodes that were chosen to be inertia nodes since they have only a single tile to which they can match, and some other node may match to that tile thus preventing the inertia node from being assigned anyplace.

To overcome this, we run a second matching just for the nodes that were unmatched during the first round (line 7). In this second matching, there are edges (u, c) between a node u and any tile c that is not occupied or blocked for u when fixing the assignments from the first matching. In this case, all the edge weights are 1. Because the tiling field is chosen to be large enough, the maximum matching in this second graph will always include an assignment of every node to some tile.

2.2 Local improvement by floating islands

To improve the layout and more easily incorporate global desiderata, we end the optimization with a phase of attempts at local improvement (line 11). We identify *islands* of tiles that are connected components of the graph after unsatisfied edges are removed. These islands are continuous regions of the tiling that we will move as a unit. To search for local improvements, each island I is considered in turn, and every possible placement I in the field of tiles is attempted while keeping the other islands fixed in place. A placement is rejected if it would cause two islands to overlap. I is left in the place that maximizes the quality q of the resulting drawing, and the next island is considered. This process continues until every island is in its locally optimal location.

For the experiments here, q is taken to be the area of the drawing so that this local improvement step will tend to push islands together and create more compact drawings. Although rotation of the islands could be executed as well, we have not yet implemented it.

2.3 Routing non-satisfied edges

We must draw a curve connecting $f(u)$ with $f(v)$ for any edge $\{u, v\}$ that f does not map to adjacent tiles. To do this (line 12), we create a graph R by repeating

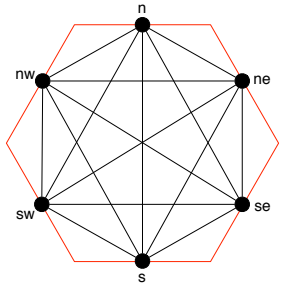

Fig. 3. Subgraph created for routing edges for each tile.

the K_6 subgraph shown in Figure 3 for every tile. The nodes of each K_6 are identified with the edges of the hexagon tile, and adjacent tiles share nodes. Edges in R that are within tiles to which f assigns some vertex of G are given weight 10, while those in unassigned tiles are given weight 1. In this way, edges drawn as curves will avoid crossing occupied cells. Although these weights are somewhat arbitrary, the 10 is chosen sufficiently bigger than the circumference of a tile so that routing around a tile is preferred over routing through one.

For every unsatisfied edge $\{u, v\} \in E$, we find the shortest path in R that connects a vertex representing a side of tile $f(u)$ to a vertex representing a side of tile $f(v)$. The edge $\{u, v\}$ is then drawn by interpolating a curve along this shortest path. This shortest path is called the *route* of edge $\{u, v\}$. (We also experimented with a variant in which unsatisfied edges are routed in arbitrary order and once an edge of R is used for a route, its weight is increased by a small amount; this tended to produce routes with complex and arbitrary curves, selected simply to avoid reusing an edge of R. Hence, in the layouts below, we allow arbitrary reuse of edges of R without additional penalty.)

2.4 Coloring non-satisfied edges

The above procedure often draws a number of routes using the same edge of R. To visually distinguish such routes, we assign colors to routes so that if two routes share an edge of R they are assigned different colors. To do this, we create a coloring graph $L = (V_L, E_L)$ where the vertices of L are the routes (corresponding to unsatisfied edges of G), and two routes are connected by an edge in E_L if they use at least one of the same edges of R. We then use a simple greedy approximation [16] to find the logical graph coloring of L that uses the fewest colors. This gives a coloring of the routes such that if two routes overlap, they are given different colors. Actual colors are selected by choosing colors in hue-lightness-saturation space with uniformly-spaced hues and near-constant (but slightly randomly perturbed) lightness and saturation.

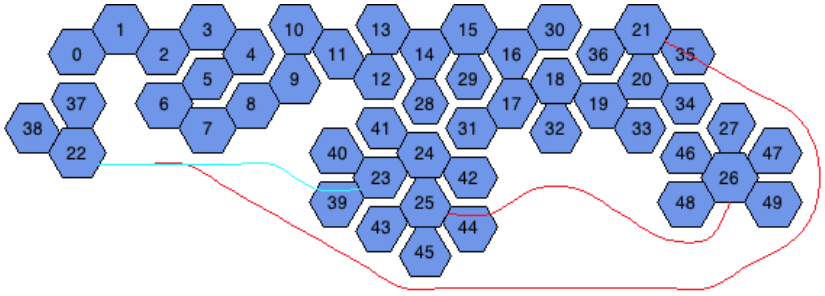

Fig. 4. A random tree with 50 nodes.

3 Results

3.1 Example tiled layouts

The hexagonal tiling layout can produce very readable and attractive drawings, as shown in Figure 2, which shows the relationship between Florentine families from a classic social network analysis [1]. Here a complex graph is drawn without edge crossings or node occlusion in a compact area.

Figure 4 shows the layout of a 50-node, random tree with a powerlaw degree distribution (computed via [7]). This graph is planar, and nearly all edges are satisfied by tile adjacencies. The layout is compact and readable. The long chain of degree-2 nodes at the root of the tree (nodes 0 through 13) remains visible, and the high-degree nodes (26, 24, 25) are easily recognizable. The ability to recognize this immediately as a tree, however, is somewhat sacrificed.

The number of times edges cross nodes is also typically much smaller for these hexagonal tiling layouts. Compare, for example, Figure 5 with Figure 6, which both show a circuit graph representing the Apple II video controller from the 2010 Graph Drawing Contest [6]. While the hexagonal tiling cannot avoid node-edge crossings entirely, the layout produced by the force-directed approach of graphviz (neato) [4] includes far more such crossings. High-degree nodes (such as `A5` and `A8`) are also more apparent in Figure 5 since they are drawn as flowers with a number of petals.

Figures 7 and 8 provide two more examples of successful layouts produced by the hexagraph tiling approach. Figure 7 is a layout of the call graph for functions called by the program that produced the tiling layout itself (only those functions that represented more than 10% of the execution of the program). Figure 8 is the "`angular`" graph from the 2011 graph drawing contest [6]. Note, for example, the excellent packing of nodes $0-6$, $8-11$, and 13 in Figure 8.

3.2 Quality of layouts on several graphs

Table 1 shows the ability of the approach described above to produce good layouts under several metrics. The graphs listed include those in Figures 2, 4, 5,

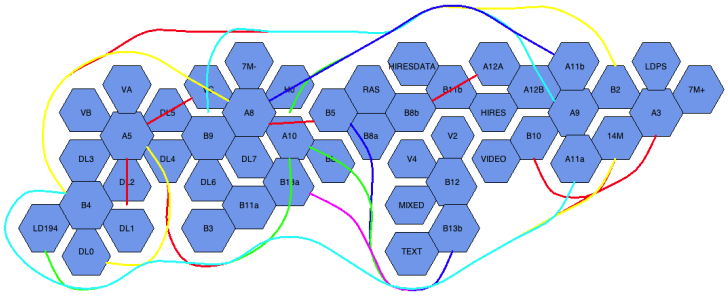

Fig. 5. Circuit graph ($n = 48$, $m = 73$) from the 2010 Graph Drawing contest.

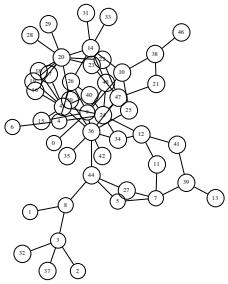

Fig. 6. Circuit graph of Figure 5 drawn using neato [4].

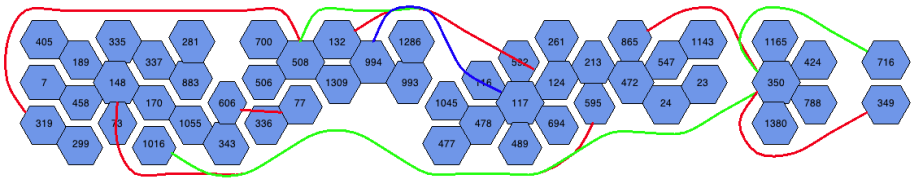

Fig. 7. Graph of function calls for the program which computes the hexagon tiling layout ($n = 52$, $m = 54$).

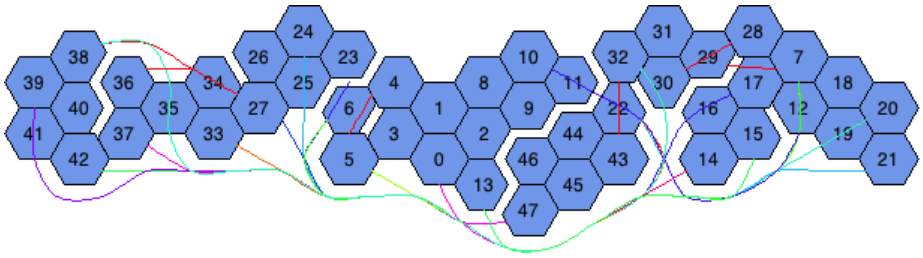

Fig. 8. The `angular A` graph ($n = 48$, $m = 102$) from the 2011 Graph Drawing Contest [6]. The graph is planar.

7, and 8, as well as a dolphin association network [11], the much-studied karate club social network [17], and a graph representing the spatial structure of human chromosome 10 [10]. The most important metric, the one for which we are most directly optimizing, is satisfying as many edges as possible via tile adjacencies. In these tests, on average 62 % of the edges can be represented by tile adjacencies, which represents a significant reduction in the number of edges that must be drawn using curves.

The number of edge colors needed is usually small, but it is larger than ideal and can sometimes be quite large. This occurs when there are bottleneck passages between assigned tiles or when there are many long-range routes to be drawn. These high-traffic routing edges introduce some edge-edge occlusion that — even despite the different colors — can make edges more difficult to track. However, the use of alternative edge glyphs or edge offsets to reduce confusion created when two edge routes use the same edge of R could be considered to mitigate this problem.

The greedy approximation for coloring works extremely well in the tested graphs. A lower bound on the number of colors needed is the number of routes that pass through the most-used edge of R. In only 3 instances, out of the 80 runs on the graphs in Table 1, were the number of colors selected more than this lower bound (in one instance each of `chrom10`, `Florentine`, and `karate`, 138, 4, and 17 colors were used when the lower bounds were 136, 3, 14, respectively). Even in these 3 instances, it may be that the greedy algorithm finds an optimum coloring if the lower bound is not tight.

The minimum area required to display a graph of n nodes is n tiles. On average, the layout is very compact, requiring an area of $1.32n$ tiles to display the nodes (the actual area may be slightly larger to accommodate edge routes around the circumference). This is evidence that the hexagon tiling approach allows very compact drawings to be created, as expected.

Another advantage of this approach is that few node-edge crossings are introduced. The median number of node-edge crossings present in the drawings is 0.23 per node. These occur when either a node is blocked on all sides by adjacent nodes, or when the endpoints of the edge are close but a non-node-crossing route would need to nearly circumnavigate the layout to reach an unblocked entry port of the node. Typically, the crossings are not uniformly distributed across the nodes and some nodes are "bottlenecks" through which many edges pass. This means that even when the number of crossings per node is higher (e.g. 0.79 for `dolphin`) the graph usually remains very readable.

3.3 Timing of the iterative linear assignment approach

Table 2 gives running time information for the graphs of Table 1. The running times are fairly practical: most graphs of ≈ 50 nodes require around 5 minutes of total computation time to produce a good layout. This requires the solution of between 26 and 389 maximum bipartite matching instances before the solution converges to a stable assignment. In addition, a local optimal is found typically after only a very few passes through the islands. Even a graph of 483 edges

Table 1. Quality of computed layouts on several graphs, averaged over 10 randomized runs. The number of nodes n and number of edges m are given. **# Satisfied** gives the edges that were satisfied by tile adjacencies. **# Colors** shows the number of colors required to color the routes. **Area** gives the area of the drawing in terms of number of tiles. **Node-Edge Crossings** are the number of times a non-satisfied edge crosses the face of some node. **Route Lengths** gives the total length of the routing of the unsatisfied edges (as the number of edges of R that they use).

Graph	n	m	# Edges Satisfied	# Colors Needed	Area	Node-Edge Crossings	Route Lengths
`circuit` [6]	48	73	49.6	6.9	63.0	15.2	182.0
`call`	52	54	40.7	5.2	72.3	7.7	110.7
`Florentine` [1]	15	20	14.0	2.6	17.8	0.6	29.4
`tree`	50	49	40.5	2.8	65.7	2.3	55.5
`dolphin` [11]	62	159	72.0	27.3	83.7	49.1	776.8
`flowchart` [6]	57	72	58.7	4.4	74.2	3.6	113.7
`karate` [17]	34	78	35.6	12.4	42.3	23.5	279.8
`chrom10` [10]	141	483	133.2	99.7	201.1	792.8	3914.9

(`chrom10`) takes only 16 minutes. These timings are evidence that the approach described above can be practical. However, the current code has not be optimized for speed: it is written in Python (using igraph [3] to compute the maximum bipartite matching). Additional engineering could likely significantly improve the speed at which layouts can be produced.

4 Discussion and Conclusion

We have shown that the grid structure allows an efficient iterative, EM-like procedure to find a compact layout with few node-edge crossings. However, there are many ways in which the "hexagraph" layout presented here could be further improved. The first is to allow non-satisfied edges to be routed through the channels introduced when two non-adjacent nodes are placed in adjacent tiles. This can be accomplished by creating a more complex routing graph R that includes edges for these channels. This will reduce the number of node-edge crossings even further.

Algorithmic improvements should also be explored to find even better assignments of nodes to tiles. Currently, edge routing and node assignment are considered almost independently: nodes are placed to satisfy edges and to minimize the Euclidean distance between endpoints of unsatisfied nodes, but the Euclidean distance can be a poor approximation to the distance an edge would have to be routed around nodes. The length of routes could be incorporated into the assignment costs in the maximum bipartite matching at the expense of recomputing routes after each matching iteration.

We have shown that high-quality, readable, compact layouts that exploit node adjacency to display the existence of edges can be computed efficiently using a

Table 2. Number of iterations of the linear assignment procedure (LAP) and the greedy floating island improvements (Island), and the number of seconds for the entire layout to be computed on a 3.4 GHz iMac in Python. Numbers are averaged over 10 randomized runs.

	Size		Iterations		
Graph	n	m	LAP	Island	Seconds
`circuit`	48	73	136.9	2.4	256.82
`call`	52	54	122.3	2.9	252.61
`Florentine`	15	20	46.7	2.2	19.63
`tree`	50	49	64.4	2.5	149.99
`dolphin`	62	159	314.1	3.1	571.25
`flowchart`	57	72	84.3	2.6	218.98
`karate`	34	78	233.9	2.5	221.50
`chrom10`	141	483	374.8	3.6	955.84

randomized, iterative linear assignment procedure, coupled with a local greedy improvement phase. These algorithms produce drawings are often produce very readable and are particularly suited for sparse graphs. We hope that this general layout scheme will find uses in several application domains.

Acknowledgements

This work has been partially funded by National Science Foundation (CCF-1256087, CCF-1053918, and EF-0849899) and National Institutes of Health (1R21AI085376 and 1R21HG006913). C.K. received support as an Alfred P. Sloan Research Fellow.

References

1. Ronald L. Breiger and Philippa E. Pattison. Cumulated social roles: The duality of persons and their algebras, 1. *Social Networks*, 8(3):215–256, 1986.
2. F Javier Cobos, J Carlos Dana, Ferrán Hurtado, Alberto Márquez, and Felipe Mateos. On a visibility representation of graphs. In *Graph Drawing*, pages 152–161. Springer, 1996.
3. Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.
4. Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*, 30(11):1203–1233, 2000.
5. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *Journal on Algebraic and Discrete Methods*, 4:312–316, 1983.
6. Graph drawing contest 2011. http://graphdrawing.de/contesthistory.html.
7. Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In Gäel Varoquaux, Travis Vaught,

and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, 2008.
8. Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE T Vis. Comput. Graphics*, 12(5), 2006.
9. Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
10. Erez Lieberman-Aiden et al. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, 2009.
11. D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson. The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54:396–405, 200.
12. Chrysanthi N. Raftopoulou Michael A. Bekos. Circle-representations of simple 4-regular planar graphs. In *Graph Drawing: Lecture Notes in Computer Science Volume 7704*, pages 138–149, 2013.
13. Roberto Tamassia and Ioannis G Tollis. A unified approach to visibility representations of planar graphs. *Discrete & Computational Geometry*, 1(1):321–341, 1986.
14. Roberto Tamassia and Ioannis G Tollis. Tessellation representations of planar graphs. In *Proc. 27th Annual Allerton Conf*, volume 43, 1989.
15. D. Tsafrir, I. Tsafrir, L. Ein-Dor, O. Zuk, D.A. Notterman, and E. Domany. Sorting points into neighborhoods (SPIN): data analysis and visualization by ordering distance matrices. *Bioinformatics*, 21(10):2301–2308, 2005.
16. D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.
17. W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.