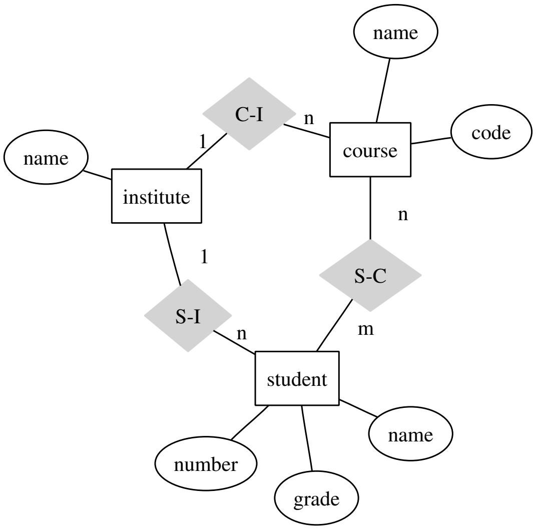
Drawing graphs with NEATO

Stephen C. North

April 26, 2004

Abstract

NEATO is a program that makes layouts of undirected graphs following the filter model of DOT. Its layout heuristic creates virtual physical models and runs an iterative solver to find low energy configurations. The intended applications are in telecommunication networks, computer programming and software engineering. Here is an example layout depicting an entity-relationship database schema. It took 0.01 seconds of user time to generate on a garden variety PC running Linux.



Entity Relation Diagram
drawn by NEATO

1 Introduction

NEATO is a utility that draws undirected graphs, which are common in telecommunications and computer programming. It draws a graph by constructing a virtual physical model and running an iterative solver to find a low-energy configuration. Following an approach proposed by Kamada and Kawai [KK89], an ideal spring is placed between every pair of nodes such that its length is set to the shortest path distance between the endpoints. The springs push the nodes so their geometric distance in the layout approximates their path distance in the graph. This often yields reasonable layouts [Ead84][FR91]. (In statistics, this algorithm is also known as multidimensional scaling. Its application to graph drawing was noted by Kruskal and Seery in the late 1970s.)

NEATO is compatible with the directed graph drawing program DOT in sharing the same input file format and graphics drivers [KN91]. Since the file format includes both undirected and directed graphs, NEATO draws graphs prepared for DOT, and vice versa. Both programs have the same options for setting labels, colors, shapes, text fonts, and pagination, and for generating code in common graphics languages (PostScript, raster formats such as GIF and PNG, SVG, FrameMaker MIF, HPGL/2, and web click maps). Both work with DOTTY, an interactive graph viewer for X windows. (The `lneato` command script runs neato from an interactive window.)

Figs. 1–4 are representative examples of NEATO's output. The timings refer to user time on a 600 Mhz Pentium Linux server. Fig. 1 was derived from a handmade drawing in an operating system tutorial. Fig. 2 shows the connectivity of a computer network. Fig. 3 shows the sharing of programmer-defined types between procedures in a C program. The program that was the source of this graph parses a text file into an internal data structure. The graph was extracted from a C program database. Its drawing shows where interactions or conversions between types may occur. Finally, Fig. 4 shows relationships between IMRs (modification requests) in an externally released software product.[1] The labeled nodes are IMRs and the small circles encode many-to-many dependencies.

---

[1] Graph courtesy of J. Hoshen, Bell Labs.

```
graph G {
    run -- intr;
    intr -- runbl;
    runbl -- run;
    run -- kernel;
    kernel -- zombie;
    kernel -- sleep;
    kernel -- runmem;
    sleep -- swap;
    swap -- runswap;
    runswap -- new;
    runswap -- runmem;
    new -- runmem;
    sleep -- runmem;
}
```

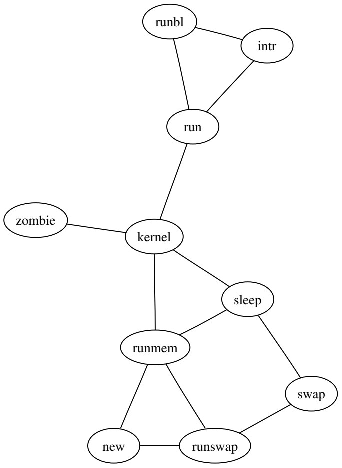


Figure 1: Process States in an Operating System Kernel (0.03 seconds)

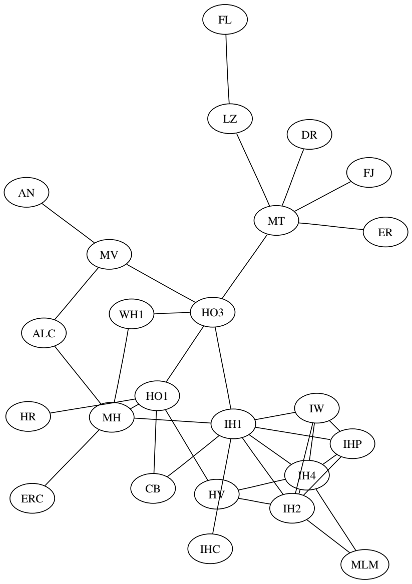


Figure 2: R&D Internet Backbone (0.08 seconds)

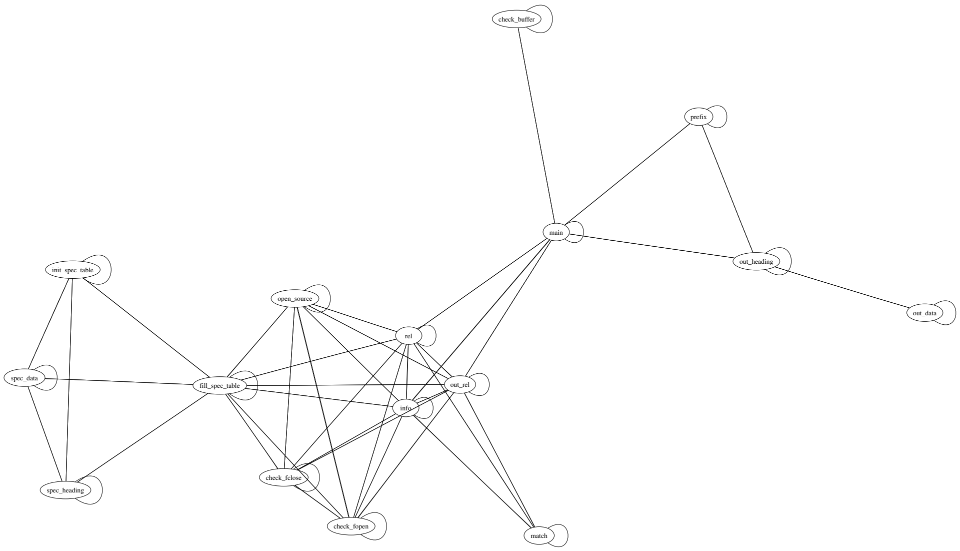


Figure 3: Type Sharing Between Procedures in a C Program (0.41 seconds)

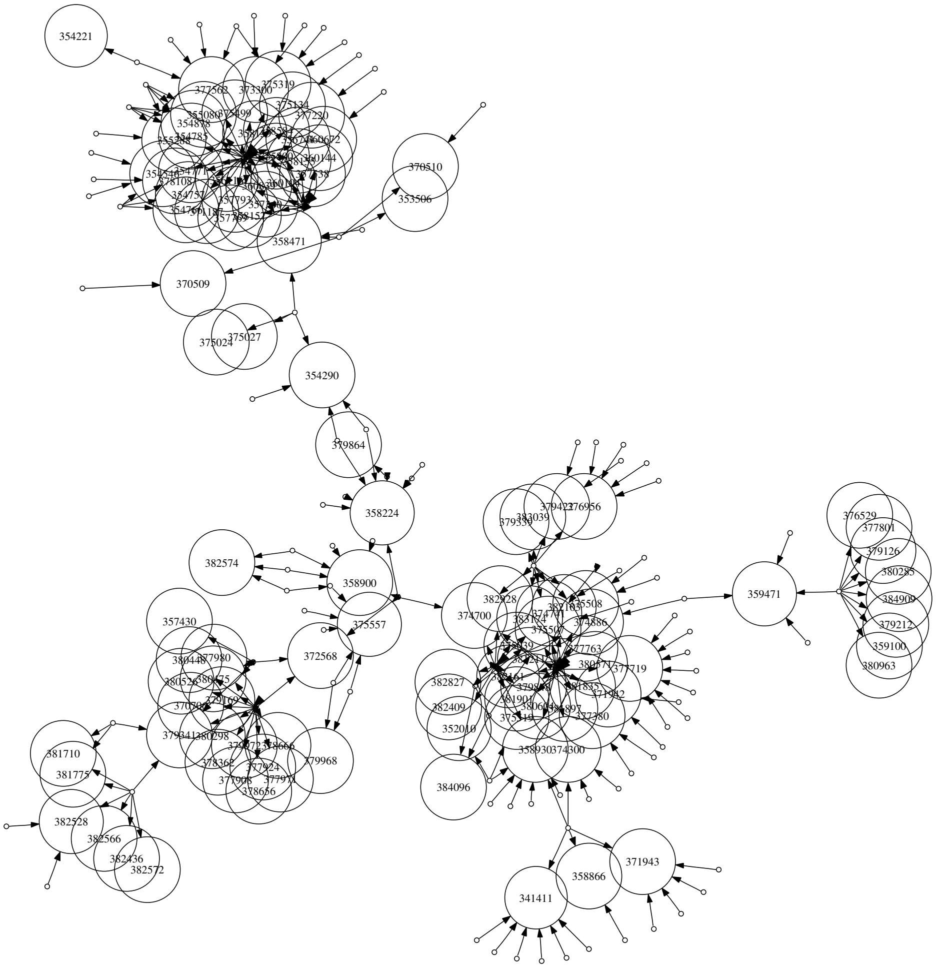


Figure 4: IMR Dependencies (6.75 seconds)

```
$ cat example.dot
graph G {
    n0 -- n1 -- n2 -- n3 -- n0;
}
$ neato -Tps example.dot -o example.ps
```

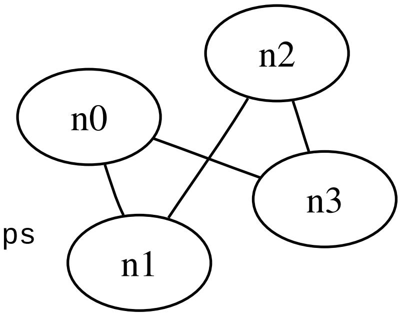


Figure 5: Example Graph Drawing

2 Graph Drawing

2.1 Basic Commands

The remainder of this memo gives a synopsis of NEATO features. Many of these should be familiar to users of DOT. Fig. 5 shows a graph file, its drawing, and the command that was executed. A graph file has a short header and a body consisting of nodes, edges, and attribute assignments. By default, nodes are drawn as ellipses labeled with node names. Undirected edges are created by the `--` operator. Edges are drawn as straight lines and tend to be all about the same length.

2.2 Drawing Options

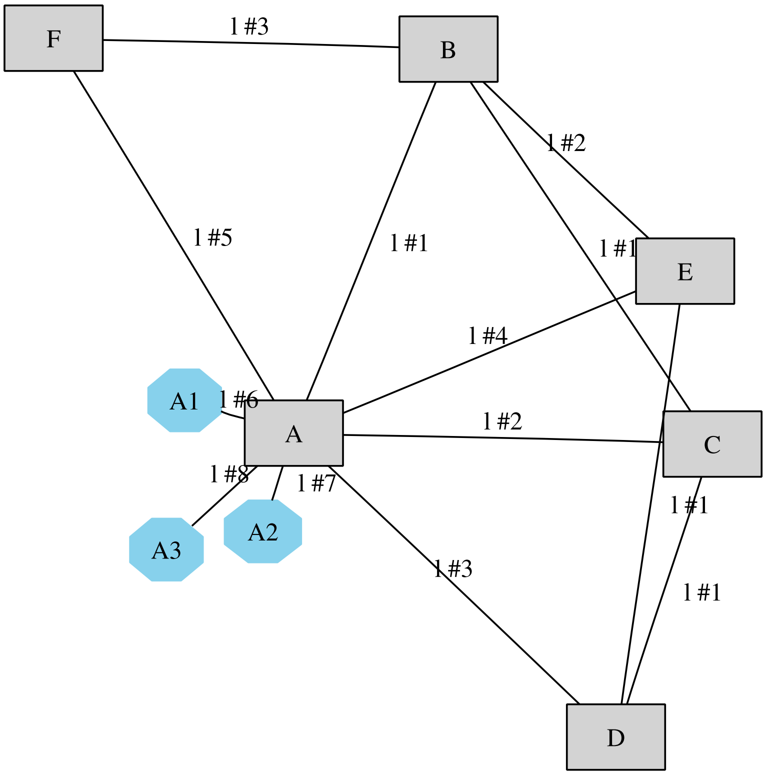
Table 1 lists the graph, node and edge attributes that affect the layout. The options to set labels, shapes, fonts, and sizes are convenient for many kinds of layouts. The drawing in figure 6 illustrates some of these features.[2] Options to set the size of the drawing, pagination, and output graphics language are also the same as in DOT.

3 Adjusting Layouts

Although layouts made by NEATO are close to a local optimum as defined by the forces the springs exert on the nodes, fine tuning or generation of alternative layouts may improve readability. Because NEATO uses unconstrained optimization, it does not enforce minimum separation constraints between nodes or between edges and

---

[2] Graph courtesy of Hector Zamora, DEFINITY.



```
graph G {
        node [shape=box,style=filled];
        {node [width=.3,height=.3,shape=octagon,style=filled,color=skyblue] A1 A2 A3}
        A -- A1 [label="l #6"];
        A -- A2 [label="l #7"];
        A -- A3 [label="l #8"];

        {edge [style=invis]; A1 -- A2 -- A3}

        edge [len=3];   /* applies to all following edges */
        A -- B [label="l #1"]; A -- C [label="l #2"]; A -- D [label="l #3"];
        A -- E [label="l #4"]; A -- F [label="l #5"]; B -- C [label="l #1"];
        B -- E [label="l #2"]; B -- F [label="l #3"]; C -- D [label="l #1"];
        D -- E [label="l #1"];
}
```

Figure 6: Node and Edge Options

nonadjacent nodes, so in dense graphs nodes and edges can be too close or overlap. There are three ways of trying to correct these errors:
1) change the initial configuration
2) adjust the solver parameters
3) edit the input edge lengths and weights.

3.1 Initial Configuration

If no options are given, NEATO always makes the same drawing of a given graph file, because its initial node placement and the solver are deterministic. Random initial placement can yield different layouts. It is sometimes reasonable to make at least several different trial layouts, and accept the best one. Random initial placement is requested by setting the value of the graph attribute `start`. If the value is a number, it is taken as a seed for the random number generator. The layout is different for each seed, but still deterministic. If the value is not a number, the process ID or current time is used. Each run potentially yields a different drawing. For example:

```
$ neato -Tps -Gstart=rand file.dot > file.ps
```

3.2 Termination Threshold

The solver is a Newton-Raphson algorithm that moves a node with a maximal δe on every iteration. The solver terminates when δe falls below some ϵ. The default $(.1)$ is low enough that the layout is usually close to a local minimum, but not so low that the solver runs for a long time without making significant progress. Smaller values of ϵ allow the solver run longer and potentially give better layouts. Larger values can decrease NEATO's running time but with a reduction in layout quality. This may be a desirable tradeoff for large graphs. ϵ is set in the graph's `epsilon` variable. You can also directly limit the number of iterations. It is convenient to do this on the command line:

```
$ neato -Tps -Gepsilon=.001 small.dot -o small.ps
$ neato -Tps -Gepsilon=1.5 big.dot -o big.ps
$ neato -Tps -Gmaxiter=1000 big.dot -o big.ps
```

3.3 Edge Lengths and Weights

Since the layout depends on the input edge lengths and their weights, these can sometimes be adjusted to good effect. The length of an edge is the preferred distance between the endpoint nodes. Its weight is the strength of the corresponding

```
graph G {
        n0 -- n1 [len=2, style=bold];
        n1 -- n2 -- n3 -- n0;
}
```

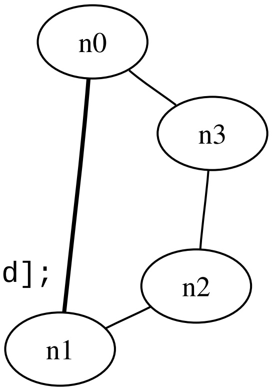


Figure 7: Example graph with an edge stretched

```
graph G {
    n0 [ pos = "0,0!" ];
    n1 [ pos = "2,0" ];
    n2 [ pos = "2,2!" ];
    n0 -- n1 -- n2 -- n3 -- n0;
}
```

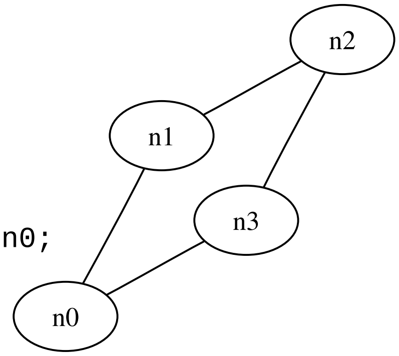


Figure 8: Example graph with nodes pinned

spring, and affects the cost if it is stretched or compressed. Invisible edges can also be inserted to adjust node placement. In figure 6, the length of some edges was set to 3 to make them longer than the default. Also, the two invisible edges affect A1, A2, and A3.

There is also a way to also give the initial or final coordinates of individual nodes. The initial position, formatted as two comma-separated numbers, is entered in a node's `pos` attribute. If `!` is given as a suffix, the node is also pinned down.

4 Eliminating Overlaps

To improve clarity, it is sometimes helpful to eliminate overlapping nodes or edges. One way to eliminate node overlaps is just to scale up the layout (in terms of the

Name	Default	Values
Node Attributes		
`shape`	`ellipse`	`ellipse`, `box`, `circle`, `doublecircle`, `diamond`, `plaintext`, `record`, `polygon`
`height`,`width`	`.5,.75`	height and width in inches
`label`	node name	any string
`fontsize`	`14`	point size of label
`fontname`	`Times-Roman`	font family name, e.g. `Courier, Helvetica`
`fontcolor`	`black`	type face color
`style`		graphics options, e.g. `bold, dotted, filled`
`color`	`black`	node shape color
`pos`		initial coordinates (append `!` to pin node)
Edge Attributes		
`weight`	`1.0`	strength of edge spring
`label`		label, if not empty
`fontsize`	`14`	point size of label
`fontname`	`Times-Roman`	font family name
`fontcolor`	`black`	type face color
`style`		graphics options, e.g. `bold, dotted, dashed`
`color`	`black`	edge stroke color
`len`	1.0	preferred length of edge
`dir`	`none`	`forward`, `back`, `both`, or `none`
`decorate`		if set, draws a line connecting labels with their edges
`id`		optional value to distinguish multiple edges
Graph Attributes		
`start`		seed for random number generator
`size`		drawing bounding box, in inches
`page`		unit of pagination, *e.g.* `8.5,11`
`margin`	`.5,.5`	margin included in `page`
`label`		caption for graph drawing
`fontsize`	`14`	point size of label
`fontname`	`Times-Roman`	font family name
`fontcolor`	`black`	type face color
`orientation`	`portrait`	may be set to `landscape`
`center`		when set, centers drawing on `page`
`overlap`	true	may be set to `false` or `scale`
`splines`	false	`true` makes edge splines if nodes don't overlap
`sep`	0	edge spline separation factor from nodes - try .1

Table 1: Drawing attributes

center points of the nodes) as much as needed. This is enabled by setting the graph attribute `overlap=scale`. This transformation preserves the overall geometric relationships in the layout, but in bad cases can require high scale factors. Another way to eliminate node overlaps employs an interative heuristic. On each iteration, a bounded Voronoi diagram of the node center points is computed, and each node is moved to the center of its Voronoi cell. This is repeated until all overlaps are eliminated. A side-effect (perhaps unwanted) is that the adjusted layout tends to fill the bounding rectangle of the Voronoi diagram. The heuristic is activated by setting `overlap=false`.

Edge overlaps (with nodes) can be prevented by drawing them with spline curves (with `splines=true`). Note that the spline drawing heuristic is expensive and probably should not be attempted on graphs that have more than a few dozen nodes.

Areas for future work include non-rectangular Voronoi boundaries, faster edge routing heuristics, and techniques to prevent unnecessary edge intersections.

5 Acknowledgments

NEATO's layout heuristic follows the work of Kamada and Kawai [KK89]. The implementation was originally part of the SALEM 3D viewer for mathematical structures written with David Dobkin and Nathaniel Thurston. In converting NEATO to a more traditional tool, the graphics code generator was borrowed from DOT. This includes code contributed by John Ellson and Emden Gansner. Steve Eick was an early user and offered some good suggestions about ways to adjust layouts. Emden Gansner also contributed the heuristics for eliminating node overlaps, based on a paper by David Rappaport and Kelly Lyons.

References

[Ead84] Peter Eades. A Heuristic for Graph Drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 1984.

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. Graph Drawing by Force-directed Placement. *Software– Practice and Experience*, 21(11):1129–1164, November 1991.

[KK89] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, April 1989.

[KN91] Eleftherios Koutsofios and Stephen North. Drawing graphs with *dot*. Technical Report 910904-59113-08TM, AT&T Bell Laboratories, Murray Hill, NJ, September 1991.