

nwplot: Flexible, publication-quality network visualization in Stata

Thomas U. Grund

Institute of Sociology, RWTH Aachen University, Germany

Abstract. I describe `nwplot`, the network-drawing command of the `nwcommands` package for the analysis of network data in Stata. A single `nwplot` call turns a network held in memory into an ordinary Stata `twoway` graph. Three things follow from that design. Node and edge attributes map onto standard graphical channels – position, size, color, symbol, line pattern – with legends built automatically, rather than requiring a separate step to compute marker coordinates by hand. A layout computed once can be exported and reused across several panels, so repeated or related networks on the same actors stay visually comparable instead of each acquiring its own, unrelated arrangement. And because `nwplot` operates on the same network objects as the rest of `nwcommands`, a network returned by another command – a shortest path, say – plots directly, with no export step in between. I walk through `nwplot`’s syntax and full option set and demonstrate each of these design choices in turn, using Padgett and Ansell’s Florentine-family marriage and business networks as the running example throughout.

Keywords: `st0002`, `nwplot`, network visualization, graph drawing, multidimensional scaling, force-directed layout, Stata graphics, `nwcommands`

1 Introduction

Visualizing a network is rarely as simple as handing an adjacency structure to a generic plotting command. A useful picture needs node *positions* that were never part of the data to begin with, has to decide what a tie looks like when it is reciprocated versus one-directional, and typically needs several visual channels – position, size, color, symbol, line width – pressed into service at once to show both the network’s structure and the attributes attached to its nodes and edges. Packages built specifically for network analysis have long supplied dedicated drawing routines for this: `igraph` (Csárdi and Nepusz, 2006) and the `statnet` suite’s `plot.network` (Handcock et al., 2008) in R are the tools most Stata users encounter this comparison against. Stata’s own graphics engine has no such counterpart: a network held as an adjacency matrix or edgelist has no native “plot this as a graph” command. Because `nwplot` works entirely inside Stata, network preparation, statistical analysis, and the resulting figure can all live in one do-file: there is no export/re-import step, and no need to reconcile two different software environments’ conventions for node identifiers, missing data, or random-number seeds along the way.

`nwplot` is that command for `nwcommands` (Grund, 2014),¹ the package that supplies

¹The cited 2014 release was an early, single-file prototype. The package has since been substantially rewritten – a sparse computational backend, a far larger command set, the visualization layer described here, among other additions – and is under active, continuing development, described in a companion article (Grund, 2026a). The software itself, not the 2014 note, is the authoritative description of the

Stata-native storage, manipulation, and analysis of network data as an extension of Stata's ordinary data-management vocabulary. Its design as a drawing command rests on four choices, each worked through at length below:

- **Stata-native graphics.** `nwplot` takes whichever network is current (or a named one), computes node coordinates under one of five layout algorithms, and draws the result as an ordinary `twoway` graph: every plot it produces accepts standard Stata `twoway` options such as `scheme()`, `legend()`, and `title()`, opens in the Graph Editor like any other graph, and exports with `graph export` exactly as usual (`nwplot`'s own `export()` is a convenience wrapper around exactly that call, Section 4.5).
- **Attributes map onto standard graphical channels.** A node- or edge-level variable becomes color, symbol, size, or line pattern with one option and an automatically built legend (Section 4.1).
- **Reproducible layouts across comparisons.** A layout computed once can be exported and reused, so related networks on the same actors stay visually comparable instead of each acquiring its own, unrelated arrangement (Section 4.6).
- **Integration with the rest of `nwcommands`.** Because `nwplot` operates on the same network objects as every other `nwcommands` command, a network returned by, say, `nwpath` plots directly, with no export step (Section 4.4).

This article is a self-contained, single-command deep dive into `nwplot`: its syntax, its full option set, and worked examples of each of the four choices above.

2 Installation

`nwplot` is one of more than 130 commands in `nwcommands` (Grund, 2014), described in a companion article (Grund, 2026a); installation is given here in full rather than by reference to that article, since the two are submitted together. From within Stata:

```
. net from  
"https://raw.githubusercontent.com/thomas-u-grund/nwcommands/master"  
. net install nwcommands  
. nwininstall, all
```

The first line points Stata at the package's source; the second installs a small, permanent bootstrap package – just `nwininstall` itself and a handful of landing help topics, so `help nwcommands` works even before anything else is on disk. `nwininstall, all` then downloads and installs everything else – the full command set, all help files, and the extension and dialog-box packages – in one call.² The `master` branch named above is

version this article covers.

²Internally this still happens as several numbered packages (`nwcommands-ado1`, `nwcommands-ado2`, ..., and likewise for the help files) because Stata's `.pkg` format caps how many files one package may list, a limit `nwcommands`' size exceeds; `nwininstall` downloads chunk 1, chunk 2, and so on until a chunk genuinely doesn't exist, so a reader never needs to know the chunk names or how many there are.

the package’s current, tested release line: active development happens on a separate `develop` branch and is merged into `master` only once tested, so `master` is what a reader should install from rather than a branch still in motion. `help nw_intro` and `help nw_topical` (both installed by `nwinstall`, `all`) give a general introduction and a full command index.

3 The nwplot command

3.1 Syntax

```
nwplot [netname] [if] [, node_options label_options edge_options
      arrow_options layout_options other_options export_options twoway_options]
```

If *netname* is omitted, `nwplot` draws the current network (the one most recently created or referenced by another `nwcommands` command). An `if` expression subsets which nodes are drawn. Every option group above is itself optional; a bare `nwplot` with the current network in memory is a complete, valid call, as the example below shows.

3.2 Options

Table 1 lists every documented option, grouped exactly as in `help nwplot` within Stata. Sub-options in *node_sub* apply inside `size()`, `color()`, and `symbol()`; sub-options in *edge_sub* apply inside `edgesize()` and `edgecolor()`; sub-options in *layout_sub* apply inside `layout()`.

Table 1: `nwplot` options, by group.

Option	Description
node_options	
<code>size(varname [, node_sub])</code>	size of the nodes
<code>color(varname [, node_sub])</code>	color of the nodes
<code>symbol(varname [, node_sub])</code>	symbol of the nodes
<code>nodefactor(float)</code>	multiply all node sizes by a factor
node_sub	
<code>norescale</code>	no automatic rescale
<code>legendoff</code>	no legend for this attribute
<code>forcekeys(int ...)</code>	list of keys to be used in the legend
<code>colorpalette(colorstyle ...)</code>	change the color palette
<code>symbolpalette(symbolstyle ...)</code>	change the symbol palette
<code>foreground(int ...)</code>	values to be plotted in the foreground
<code>sizebin(int)</code>	finetune the size of nodes
<code>mlcolor(colorstyle)</code>	line color of nodes
<code>mlwidth(linewidthstyle)</code>	line width of nodes
<code>nodeclash(real)</code>	separate overlapped nodes in <code>mdsclassical</code>
label_options	
<code>lab</code>	display node labels saved with the network
<code>label(varname)</code>	display node labels from a variable
<code>labelopt(marker_label_options)</code>	size, color, position of node labels
edge_options	
<code>edgesize(netname [, edge_sub])</code>	use another network’s edge values as edge width
<code>edgecolor(netname [, edge_sub])</code>	use another network’s edge values as edge color
<code>edgefactor(float)</code>	multiply all edge sizes by a factor

continued on next page

Table 1 (continued)

Option	Description
edge_sub	
<code>legendoff</code>	no legend for this attribute
<code>forcekeys(int ...)</code>	list of keys to be used in the legend
<code>edgecolorpalette(colorstyle ...)</code>	change the color palette
<code>edgepatternpalette(pattern ...)</code>	vary line pattern along with <code>edgecolor()</code>
<code>foreground(int ...)</code>	values to be plotted in the foreground
arrow_options	
<code>arcstyle(arcstyle)</code>	automatic (default), curved, or straight
<code>arcbend(float)</code>	degree of bend for curved arcs; default 2
<code>arcsplines(int)</code>	resolution for curved arcs
<code>arrowfactor(float)</code>	multiply the arrowhead by a factor
<code>arrowgap(float)</code>	gap between arrowhead and node
<code>arrowbarbfactor(float)</code>	control the look of the arrowhead
layout_options	
<code>layout([layoutstyle] [, layout_sub])</code>	overall arrangement of nodes
<code>nodexy(xvar yvar)</code>	force node coordinates from two variables
<code>generate(newxvar newyvar)</code>	export the node coordinates just computed
layout_sub	
<code>lgc</code>	plot only the largest component
<code>iterations(int)</code>	max. iterations for <code>mds</code> and <code>frucht</code> ; default 1,000
<code>columns(int)</code>	number of columns for <code>grid</code>
<code>noreyscale</code>	do not rescale <code>nodexy()</code> coordinates
layoutstyle	
<code>mds</code>	modern multidimensional scaling; default for fewer than 50 nodes
<code>mdsclassical</code>	classical multidimensional scaling; default for 50 or more nodes
<code>frucht</code>	Fruchterman–Reingold force-directed layout
<code>circle</code>	circle layout
<code>grid</code>	grid layout
<code>nodexy</code>	use the coordinates given in <code>nodexy()</code>
other_options	
<code>aspectratio(float)</code>	height/width ratio
<code>lineopt(line_options)</code>	options passed directly to the edge line plots
<code>scatteropt(scatter_options)</code>	options passed directly to the node scatter plots
<code>legendopt(legend_options)</code>	options passed directly to the legend
export_options	
<code>export(filename)</code>	export the plot directly; format inferred from the extension
<code>replace</code>	overwrite <i>filename</i> if it already exists
<code>exportopt(export_options)</code>	options passed to the underlying <code>graph export</code> call

3.3 Purpose-built network schemes

`nwplot` ships three graphics schemes purpose-built for network plots – `s1network`, `s2network`, and `s3network` – and defaults to `s1network` unless `scheme()` is specified explicitly. An ordinary Stata scheme typically gives a single data series’ marker and connecting line the same color, which is the right choice for a conventional statistical graph but the wrong one for a network plot, where node fill and edge line are two visually distinct elements that should not merge into each other. Section 4.7 compares the shipped default against an ordinary scheme directly.

4 Remarks and examples

The examples below use a single published network dataset that ships with `nwcommands` and is loaded with `nwebuse`: Padgett and Ansell's 16-family Florentine data (Padgett and Ansell, 1993), which records two distinct tie types among the same 16 families – marriage alliances (`flomarriage`) and business partnerships (`flobusiness`) – and carries three family-level attributes: `wealth` (net wealth in 1427, in thousands of lira), `priorates` (number of seats held on the city's civic council between 1282 and 1344), and `seat` (whether the family held any council seat at all).

4.1 A first plot

`nwplot`'s design is easiest to see in a single call: `size()`, `color()`, and `symbol()` each take a *varname* and map it onto the corresponding visual channel, with a legend built automatically, so three family-level attributes of the Florentine marriage network can be shown at once, each contributing its own legend key:

```
. nwebuse florentine, nwcLEAR
. recode wealth (0/30=1 "low wealth") (31/60=2 "mid wealth") (61/200=3 "high we
> alth"), ///
> generate(wealthtier)
(16 differences between wealth and wealthtier)
. label values wealthtier wealthtier
. set seed 42
. nwplot flomarriage, color(wealthtier) symbol(seat) size(priorates, forcekeys(
> 0 20 40 60)) ///
> layout(mds) legendopt(pos(3) cols(1) size(small)) xsize(6) ysize(5)
Calculating node coordinates...
Plotting network...
```

The *node_sub* options in Table 1 give finer control over any one channel: `colorpalette()`/`symbolpalette()` replace the default palette, `foreground()` draws specified values on top of the rest, `legendoff` suppresses one attribute's legend while keeping others, and `mlcolor()`/`mlwidth()` set the outline of the node markers themselves.

Every option above is optional, though: with no *netname* and no options at all, `nwplot` draws the current network under its own defaults –

```
. set seed 12345
. nwcLEAR
. nwrandom 20, prob(.2)
. nwplot, xsize(6) ysize(5)
Calculating node coordinates...
Generating splines...
Plotting network...
```

4.2 Choosing a layout

`layout()` selects among five algorithms (Table 1, *layoutstyle*): `mds`, modern multidimensional scaling (Kruskal, 1964; Borg and Groenen, 2005), is the default below 50 nodes; `mdsclassical` (Borg and Groenen, 2005) takes over at 50 nodes and above,

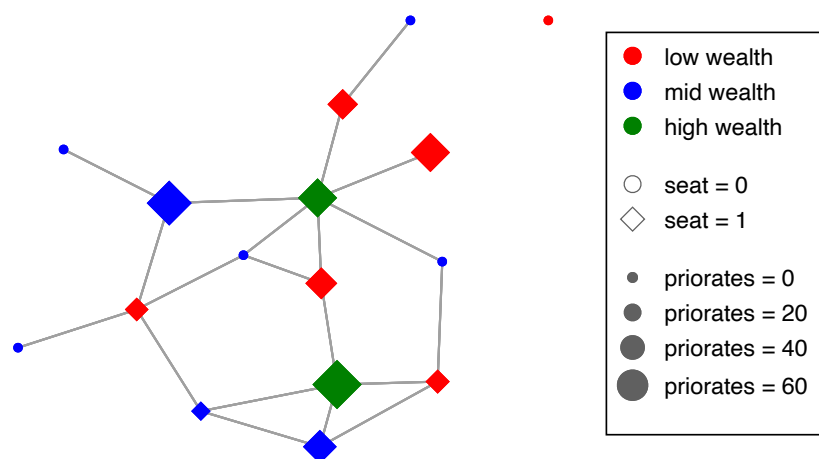


Figure 1: The Florentine marriage network: node fill color for a three-tier recode of **wealth**, symbol for whether the family ever held a council seat, and size for number of priorates. `forcekeys()` keeps the continuous **priorates** legend to four representative values rather than one key per distinct count.

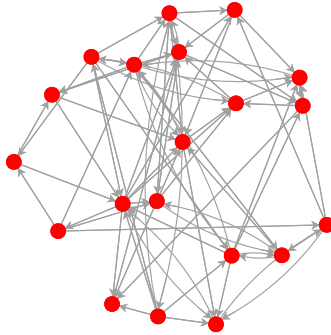


Figure 2: A 20-node random network, drawn with every default in effect.

since it is markedly faster on larger networks; **frucht** implements the Fruchterman–Reingold force-directed algorithm (Fruchterman and Reingold, 1991); **circle** and **grid** place nodes on a circle or a rectangular grid regardless of tie structure. **mds** and **frucht** are both iterative, bounded by **iterations()** (default 1,000 for both); **mdsclassical** is not, which is the practical reason it is faster and becomes the default from 50 nodes up. None of the three structure-aware layouts is immune to visual clutter on a large, dense network, though – Section 4.8’s **lgc** restriction to the largest component is one direct way to reduce it; this article does not benchmark **nwplot** itself at scale; a reader working with networks of several thousand nodes or more should time **layout()** on their own data before committing to a particular algorithm. Figure 3 compares three of the five on the 16-node Florentine business network:

```
. nwwebuse florentine, nwcLEAR
. nwplot flobusiness, layout(circle) title("circle") name(g1, replace)
Calculating node coordinates...
Plotting network...
. nwplot flobusiness, layout(grid) title("grid") name(g2, replace)
Calculating node coordinates...
Plotting network...
. set seed 43
. nwplot flobusiness, layout(mds) title("mds") name(g3, replace)
Calculating node coordinates...
Plotting network...
```

circle and **grid** ignore the tie structure entirely, which is precisely why the five business isolates (families with no recorded business tie) are pushed to the visual periphery under **mds** but scattered evenly under the other two – a direct illustration of what a structure-aware layout buys over an arbitrary one.

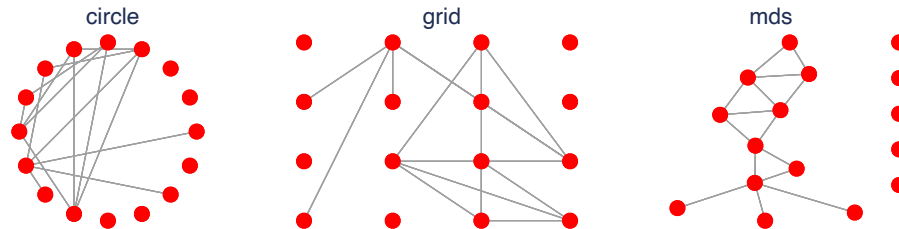


Figure 3: The Florentine business network under three layouts.

4.3 Directed networks and arcs

When a network is directed, `nwplot` draws arrowheads automatically and distinguishes a reciprocated (mutual) dyad from a one-directional one: `arcstyle(automatic)`, the default, draws only reciprocated dyads as a pair of curved arcs – so both directions of a mutual tie remain visible rather than overlapping into a single line – while every asymmetric tie is drawn straight. `arcstyle(curved)` and `arcstyle(straight)` override this and apply one style to every tie. `arcbend()` and `arcsplines()` control the degree of bend and the resolution of curved arcs. A small directed network with two reciprocated dyads ($A \leftrightarrow B$ and $C \leftrightarrow D$) and two asymmetric ties ($D \rightarrow E$ and $E \rightarrow A$) makes the difference concrete:

```
. nwclean
. nwset, mat((0,1,0,0,0\1,0,1,0,0\0,0,0,1,0\0,0,1,0,1\1,0,0,0,0)) ///
>      directed name(toy) labs(A,B,C,D,E)
. nwplot toy, lab arcstyle(automatic) title("automatic") name(g1, replace)
Calculating node coordinates...
Generating splines...
Plotting network...
. nwplot toy, lab arcstyle(straight) title("straight") name(g2, replace)
Calculating node coordinates...
Plotting network...
. nwplot toy, lab arcstyle(curved) title("curved") name(g3, replace)
Calculating node coordinates...
Generating splines...
Plotting network...
```

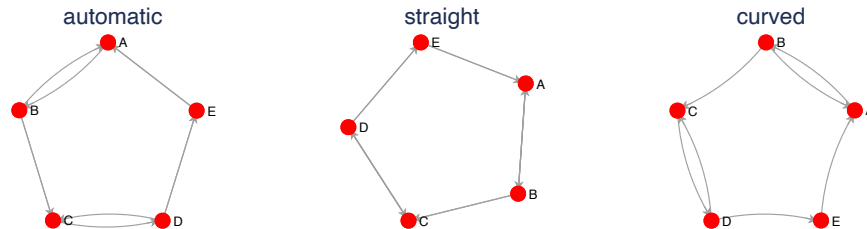


Figure 4: The same directed network under the three `arcstyle()` settings. Only `automatic` distinguishes reciprocated from asymmetric ties.

4.4 Encoding edge attributes: highlighting a shortest path

`edgesize()` and `edgecolor()` work the same way for edges, except that the values they map come from a *second* network of the same dimensions rather than a node-level variable – typically one generated by another `nwcommands` command. Here, `nwpath` computes the shortest path between two families in the Florentine marriage network and returns each tied shortest path as its own same-dimension network (`sp_1`, `sp_2`, ...) whose only nonzero cells are the edges that lie on that path. Passing one of those to `edgecolor()` alone – without also setting `edgesize()` – recolors exactly those edges while leaving every other edge in the network at its normal width and color, so the highlighted path reads in context rather than in isolation; `edgesize()` would instead scale every edge's *width* by the referenced network's values, and since the non-path edges are 0 there, adding it would shrink them to nothing. The two panels below make this explicit, plotting the same network first with no highlighting and then with the Medici–Peruzzi path picked out by `edgecolor()` alone:

```
. nwwebuse florentine, nwcLEAR
. nwpath flomarriage, ego(medici) alter(peruzzi) generate(sp)
```

```
Network: flomarriage
```

```
Ego           : medici
Alter          : peruzzi
Shortest path length : 3
```

```
Path 1: medici <=> barbadori <=> castellani <=> peruzzi
Path 2: medici <=> ridolfi <=> strozzi <=> peruzzi
. set seed 42
. nwplot flomarriage, lab title("full network") ///
```

```

> labelopt(mlabsize(small) mlabposition(12)) name(g1, replace)
Calculating node coordinates...
Plotting network...

. set seed 42

. nwplot flomarriage, lab edgecolor(sp_1, legendoff) ///
> labelopt(mlabsize(small) mlabposition(12)) title("Medici-Peruzzi path
> ") name(g2, replace)
Calculating node coordinates...
Plotting network...

```

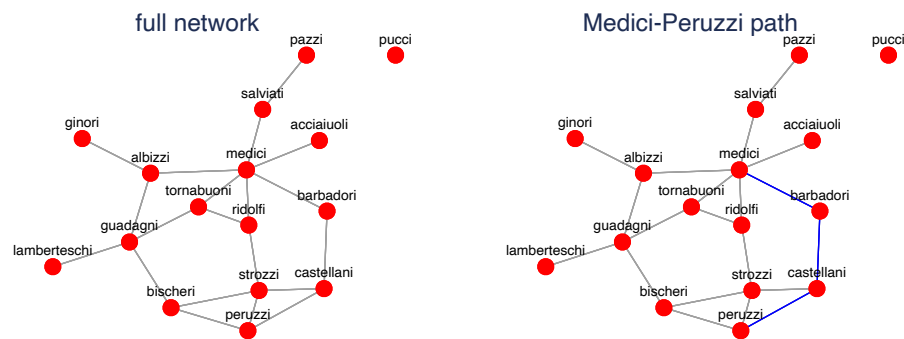


Figure 5: The Florentine marriage network in full, and with one of the (tied) shortest paths between the Medici and the Peruzzi highlighted via `edgecolor()` alone – every other edge stays visible at its normal color and width.

4.5 Publication-quality export

`export(filename)` saves the plot directly, inferring the format (SVG, PDF, EPS, PNG, TIF, ...) from `filename`'s extension – exactly what a manual `graph export` call afterward would do, since `export()` never bypasses `graph export` itself. SVG and PDF are scalable: the plot stays crisp at any zoom level, and both open in vector editors such as Illustrator or Inkscape with node, edge, and label elements as separate, still-editable objects. `exportopt()` passes options straight through to the underlying `graph export` call, most commonly `width()/height()` to set resolution for a raster format. `nwplot` is `rclass` and stores one macro in `r()`, the filename actually used, in `r(export)`:

```

. nwplot flomarriage, edgecolor(sp_1, legendoff) export("flomarriage_path.pdf")
> replace
Calculating node coordinates...
Plotting network...
Exporting graph to flomarriage_path.pdf...
file /Users/tgrund/FILES_NEW/RESEARCH/2026 nwplot/paper/flomarriage_path.pdf

```

```

        saved as PDF format
    . display "`r(export)'"
    flomarriage_path.pdf

```

The graph itself is unaffected by `export()`: it remains the normal, currently active graph afterward, so the Graph Editor, `graph save`, and a second `graph export` in a different format all continue to work exactly as they would without `export()`.

4.6 Reusing coordinates across panels

`generate()` and `nodexy()` are a matched pair: one exports the node coordinates a layout just produced, the other forces a later plot to reuse them. This is how to compare several networks on the same set of nodes – several tie types among the same actors, say – side by side, without each panel re-deriving its own, visually unrelated layout. `mds` and `frucht` start from random initial coordinates and re-optimize from there on every call, so their layouts can differ across repeated calls unless the random-number state is controlled: fixing the seed reproduces a given layout exactly, but two calls with different (or unset) seeds are not guaranteed to agree even on an identical network. Without `generate()/nodexy()`, then, two independently plotted networks on the same 16 families would typically end up in two unrelated layouts even if the ties themselves were identical. The Florentine data's marriage and business networks are two distinct tie types on that same set of 16 families, which makes them a natural pair to compare directly:

```

. nwwebuse florentine, nwcLEAR
. set seed 42
. nwplot flomarriage, generate(x y) layout(mds) title("marriage ties") name(g1,
> replace)
Calculating node coordinates...
Plotting network...
Export coordinates...
. nwplot flobusiness, nodexy(x y) title("business ties, marriage-tie coordinate
> s") name(g2, replace)
Plotting network...

```

4.7 Purpose-built schemes versus an ordinary Stata scheme

Section 3.3 noted that `nwplot` ships schemes purpose-built for network graphics because an ordinary scheme's usual assumption – that one series' marker and connecting line should match – is exactly wrong for a network plot. Figure 7 makes the point directly by comparing the shipped default with `stcolor`, Stata's current default scheme:

```

. nwwebuse florentine, nwcLEAR
. recode wealth (0/30=1 "low wealth") (31/60=2 "mid wealth") (61/200=3 "high we
> alth"), ///
> generate(wealthtier)
(16 differences between wealth and wealthtier)
. label values wealthtier wealthtier
. set seed 42
. nwplot flomarriage, color(wealthtier) size(priorates, forcekeys(0 20 40 60))

```

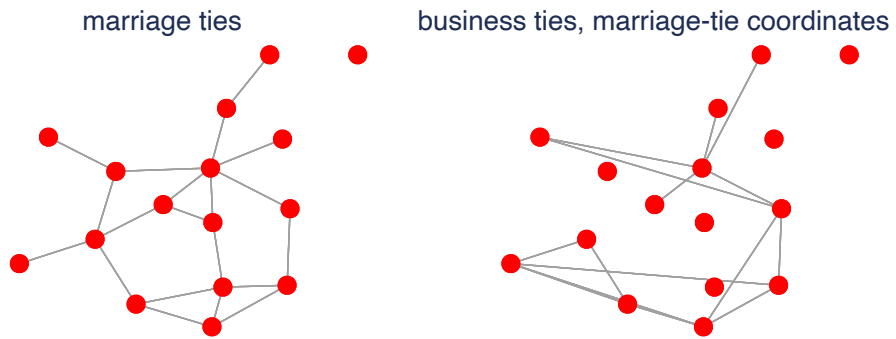


Figure 6: The same 16 Florentine families under two tie types, both drawn at the marriage network’s node coordinates. Because node position is now held fixed, the two panels can be compared directly, and the differences between them reflect real differences between who a family married into and who it did business with – not an artifact of two independently derived layouts.

```

> scheme(s1network) ///
>         layout(mds) title("scheme(s1network)") legendopt(pos(3) cols(1) size(
> small)) name(g1, replace)
Calculating node coordinates...
Plotting network...

. set seed 42

. nwplot flomarriage, color(wealthtier) size(priorates, forcekeys(0 20 40 60))
> scheme(stcolor) ///
>         layout(mds) title("scheme(stcolor)") legendopt(pos(3) cols(1) size(sm
> all)) name(g2, replace)
Calculating node coordinates...
Plotting network...

```

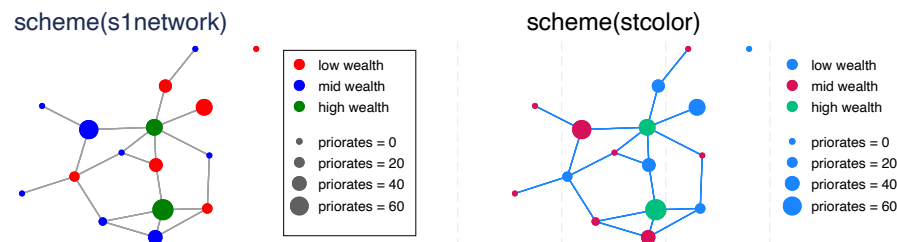


Figure 7: The same node-attribute plot under `scheme(s1network)` (the `nwplot` default) and under `scheme(stcolor)` (Stata's own current default scheme).

4.8 Restricting the plot to the largest component

`layout()`'s `lgc` sub-option restricts the plot to the largest connected component, dropping isolates and small peripheral components that would otherwise clutter a large, sparse network:

```

. nwwebuse florentine, nwclean
. nwcomponents flobusiness, generate(comp)

```

Network name: flobusiness
Components: 6

comp	Freq.	Percent	Cum.
1	1	6.25	6.25
2	1	6.25	12.50
3	11	68.75	81.25

4	1	6.25	87.50
5	1	6.25	93.75
6	1	6.25	100.00
Total	16	100.00	

```
. set seed 43
. nwplot flobusiness, layout(mds) title("all components") name(g1, replace)
Calculating node coordinates...
Plotting network...
. set seed 43
. nwplot flobusiness, layout(mds, lgc) title("largest component only") name(g2,
> replace)
(5 observations deleted)
Calculating node coordinates...
Plotting network...
```

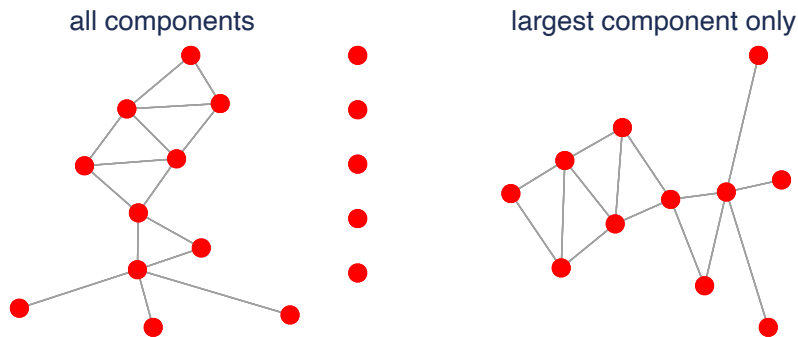


Figure 8: The Florentine business network in full (6 components: an 11-family giant component plus 5 isolated families with no recorded business tie) versus restricted to its largest component.

4.9 Legends and passthrough options

`legendopt()` accepts any ordinary `legend_options` and applies them to `nwplot`'s automatically built legend – Figures 1 and 7 both use `legendopt(pos(3) cols(1))` to place a one-column legend to the right of the plot. For control beyond what `nwplot`'s own options expose, `scatteropt()` and `lineopt()` forward options directly to the underlying node `scatter` and edge `line` calls, and `nwplot` accepts every ordinary `twoway_option`

in addition to its own – `title()` is already used throughout the figures above, and `xscale(off) yscale(off)` is a common addition for a network plot, whose axes rarely carry meaningful units.

5 Conclusions

The four design choices set out in the introduction – Stata-native graphics, attributes mapped onto standard graphical channels, reproducible layouts across comparisons, and integration with the rest of **nwcommands** – are really one choice restated four ways: a plotted network never has to leave Stata. That matters less for any single figure than for a whole workflow: network preparation, the statistical analysis built on it, and the figures that go into a paper can live in one do-file and one version-control history, without a round trip through a separate plotting package’s own file formats and node-identifier conventions.

That design also sets the limits of what **nwplot** draws well. **nwplot** plots binary and weighted networks alike (a weighted network’s own values are exposed to `edgesize()/edgecolor()` exactly as in Section 4.4). Directed networks are its native case: arrowheads are automatic, and `arcstyle(automatic)`’s reciprocated/asymmetric distinction (Section 4.3) is a direct consequence of how the package already represents directed dyads internally. Undirected is the default. A two-mode network’s nodes and ties plot correctly, since a bipartite network’s ties are simply a subset of the same one-mode adjacency structure every other network uses – but the two modes are not distinguished automatically, so the reader is expected to pass the network’s own mode indicator to `color()` or `symbol()` to tell the two node sets apart at a glance. Self-loops are the one unsupported case: a self-loop is a zero-length tie and currently has no visible effect on the plot, a known limitation recorded on the package’s own visualization roadmap rather than addressed here. And no layout algorithm rescues a node-link diagram from a large, dense-enough network: for sufficiently large or dense networks, **nwcommands**’ **nwplotmatrix** provides a complementary matrix representation instead; see `help nwplotmatrix`. **nwplot** is one of more than 130 commands in **nwcommands**; its network-object architecture, sparse computational backend, and common command interface – the source of the rest of what those other commands return for **nwplot** to draw – are described in a companion article (Grund, 2026a), submitted alongside this one.

Also see

nwcommands (Grund, 2014), **nwplotmatrix**

Acknowledgments

The examples in this article use the Florentine-family data of Padgett and Ansell (1993), which ships with **nwcommands** via **nwwebuse**.

6 References

- Borg, I., and P. J. F. Groenen. 2005. *Modern Multidimensional Scaling: Theory and Applications*. 2nd ed. New York: Springer.
- Csárdi, G., and T. Nepusz. 2006. The igraph software package for complex network research. *InterJournal, Complex Systems* 1695. URL <https://igraph.org>.
- Fruchterman, T. M. J., and E. M. Reingold. 1991. Graph Drawing by Force-Directed Placement. *Software: Practice and Experience* 21(11): 1129–1164.
- Grund, T. U. 2014. nwcommands: Software Tools for the Statistical Modeling of Network Data in Stata. Computer software. URL <https://github.com/thomas-u-grund/nwcommands>.
- . 2026a. nwcommands: Scalable and Extensible Network Analysis in Stata. Manuscript under review.
- Handcock, M. S., D. R. Hunter, C. T. Butts, S. M. Goodreau, and M. Morris. 2008. statnet: Software Tools for the Representation, Visualization, Analysis and Simulation of Network Data. *Journal of Statistical Software* 24(1): 1–11.
- Kruskal, J. B. 1964. Nonmetric Multidimensional Scaling: A Numerical Method. *Psychometrika* 29(2): 115–129.
- Padgett, J. F., and C. K. Ansell. 1993. Robust Action and the Rise of the Medici, 1400–1434. *American Journal of Sociology* 98(6): 1259–1319.

About the author

Thomas U. Grund is Professor at the Institute of Sociology, RWTH Aachen University, Germany.