A short guide to graph drawing options in LaTeX

Michelle Ong

April 29, 2024

This document outlines some options available for drawing Ti*k*Z graphs in LaTeX. Here "graphs" refer only to vertex-and-edge graphs, not function plots. Note:

- This document does not teach you how to use LaTeX. You must already know how to compile a document and use packages in LaTeX.
- This is not a tutorial. The code examples only illustrate some basic capabilities of each option. For more details, refer to the package documentation at CTAN (or just experiment with the editor for the first option).
- The options are ordered from least to most reliant on Ti*k*Z knowledge.
- For all the options, you will need `\usepackage{tikz}` in your preamble.

1 Visual editing with mathcha.io

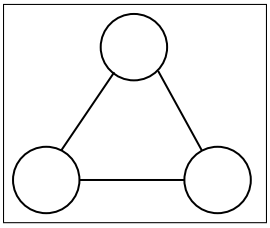
The first option requires no knowledge of Ti*k*Z at all. To get started:

1. Go to mathcha.io/editor and create an account by selecting Login > Register a New Account, then following the instructions.
2. From the editor page, create a new document from the sidebar on the left.
3. From the small floating options bar, select Create New Diagram (the icon with three shapes and a dropdown arrow) > Drawing Area.
4. To create a new vertex, select the rectangle icon > circle.
5. To create a new edge, select the line icon > line of your choice. There are options for both plain lines (for undirected graphs) and arrows (for directed graphs).

If your graph has some symmetry, you can select Options > Snap to Other Shapes, then use a regular polygon as a "skeleton" for your graph, which you can later delete. This makes it much easier to get the alignment right.

You may want to consider other visual editors which can produce Ti*k*Z output, such as GeoGebra or FreeTikZ. However, GeoGebra has little flexibility, and FreeTikZ might not give you enough control over the output.

Only use mathcha if your graphs are too complex for the other options, or if you will not need similar graphs in future. This is because a major downside of using mathcha is that the Ti*k*Z code produced is relatively opaque. For example, drawing a simple triangle generates this code (line breaks added for readability):



```
\tikzset{every picture/.style={line width=0.75pt}}
\begin{tikzpicture}[x=0.5pt,y=0.5pt,yscale=-1,xscale=1]
%Shape: Circle [id:dp5516491723986057]
\draw (165,57) .. controls (165,43.19) and (176.19,32) ..
(190,32) .. controls (203.81,32) and (215,43.19) .. (215,57) ..
controls (215,70.81) and (203.81,82) .. (190,82) ..
controls (176.19,82) and (165,70.81) .. (165,57) -- cycle ;
%Shape: Circle [id:dp1907668025451701]
\draw (228,157) .. controls (228,143.19) and (239.19,132) ..
(253,132) .. controls (266.81,132) and (278,143.19) .. (278,157)
.. controls (278,170.81) and (266.81,182) .. (253,182) ..
controls (239.19,182) and (228,170.81) .. (228,157) -- cycle ;
%Shape: Circle [id:dp702586341190387]
\draw (99,157) .. controls (99,143.19) and (110.19,132) ..
(124,132) .. controls (137.81,132) and (149,143.19) ..
(149,157) .. controls (149,170.81) and (137.81,182) .. (124,182)
.. controls (110.19,182) and (99,170.81) .. (99,157) -- cycle ;
%Straight Lines [id:da5118658505543359]
\draw (175.21,75.88) -- (135.21,134.88) ;
%Straight Lines [id:da19441153454022508]
\draw (208.21,74.88) -- (241.21,134.88) ;
%Straight Lines [id:da4522247193611004]
\draw (149,157) -- (228,157) ;
\end{tikzpicture}
```

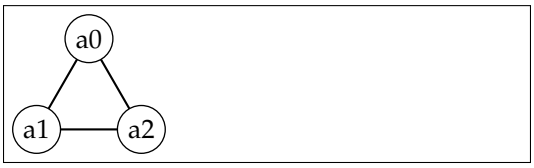
It is easy to continue editing this diagram in the mathcha editor, but much harder to modify just by looking at the code. This means you risk losing reusability of your diagrams, if mathcha ever stops being maintained.

Vertex and edge colours, as well as text (including mathematics), are easy to add.

2 Predefined graphs with tkz-berge

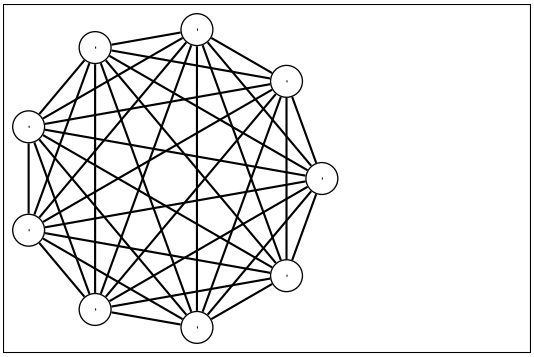
This option requires `\usepackage{tkz-berge}` in your preamble.

The package tkz-berge makes drawing certain classic graphs and common constructions easy. For example, a triangle can be produced as follows:



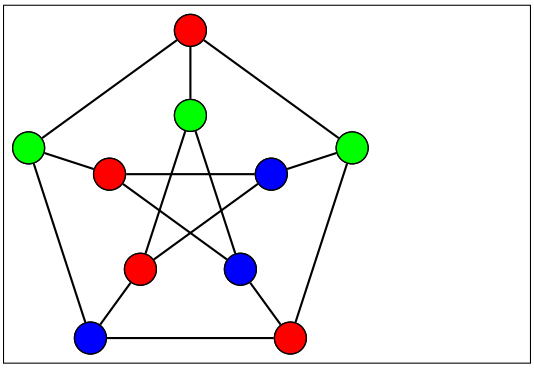

```
\begin{tikzpicture}[scale=.2, rotate=90]
  \grCycle{3}
\end{tikzpicture}
```

Similarly, a complete graph takes nearly no effort:



```
\begin{tikzpicture}
  \GraphInit[vstyle=Hasse]
  \grComplete[RA=2]{9}
\end{tikzpicture}
```

And a 3-coloring of the Petersen graph:



```
\begin{tikzpicture}[rotate=90,scale=.75]
  \GraphInit[vstyle=Hasse]
  \grPetersen[RA=3,RB=1.5]
  \AddVertexColor{red}{a0,b1,b2,a3}
  \AddVertexColor{green}{a1,b0,a4}
  \AddVertexColor{blue}{b4,b3,a2}
\end{tikzpicture}
```

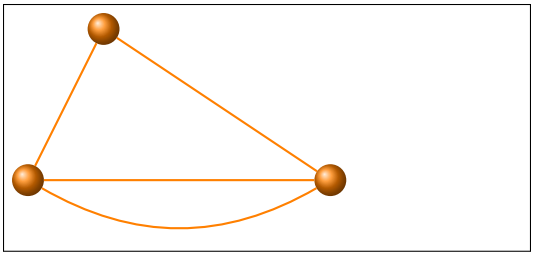
The list of all available named graphs can be found here. Additional options are available to change vertex styles, adjust the radius, etc.

3 Predefined macros with tkz-graph

This option requires `\usepackage{tkz-graph}` in your preamble.

The author of the tkz-berge package also wrote the tkz-graph package, which contains some macros to make graph drawing less verbose than in plain Ti*k*Z. This is more general and flexible than tkz-berge. Unfortunately, the documentation for tkz-graph is currently only available in French, though the commands have English names and it is possible to decipher the examples with some effort.

Here is an example taken from the documentation:

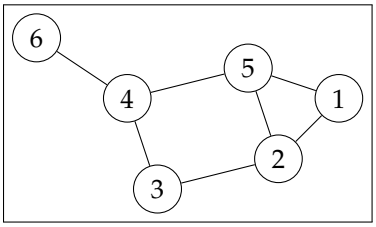


```
\begin{tikzpicture}
  \GraphInit[vstyle=Art]
  \Vertex{A}
  \Vertex[x=4,y=0]{B}
  \Vertex[x=1,y=2]{C}
  \Edge[style={bend left}](B)(A)
  \Edges(A,B,C,A)
\end{tikzpicture}
```

4 Plain Ti*k*Z

In between the extremely unreadable but flexible graphical editor option, and the extremely readable but more rigid tkz package options, are the capabilities from Ti*k*Z itself.

With no additional setup, Ti*k*Z on its own can do a lot. For example, here are 6 labelled nodes at specified positions, with a `for` loop to draw the edges:

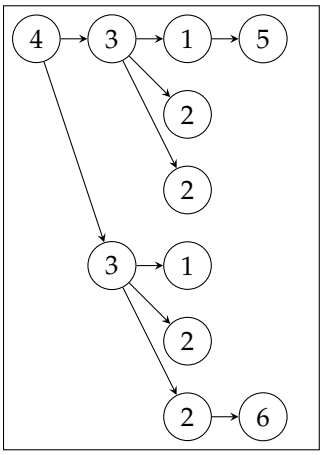



```
\begin{tikzpicture}
[scale=.4,auto=left,every node/.style={circle, draw}]
  \node (n6) at  (1,10)   {6};
  \node (n4) at  (4,8)    {4};
  \node (n5) at  (8,9)    {5};
  \node (n1) at  (11,8)   {1};
  \node (n2) at  (9,6)    {2};
  \node (n3) at  (5,5)    {3};
  \foreach \from/\to in
    {n6/n4,n4/n5,n5/n1,n1/n2,n2/n5,n2/n3,n3/n4}
  \draw (\from) -- (\to);
\end{tikzpicture}
```

4.1 Default Ti*k*Z libraries

This option requires `\usetikzlibrary{graphdrawing}` and `\usetikzlibrary{graphs}` in your preamble. You must also use LuaTeX to compile (not pdfTeX or XeTeX).

Using these libraries, you can create for example (adapted from here):




```
\begin{tikzpicture}
[>=stealth, every node/.style={circle, draw}]
  \graph [grow=down,
      fresh nodes, level distance=0.5in,
      sibling distance=0.5in]
    {
      4 -> {
        3 -> { 1 -> { 5 }, 2, 2 },
        3 -> { 1, 2, 2 -> 6 }
      }
    };
\end{tikzpicture}
```

Note that in contrast to the plain Ti*k*Z approach, no absolute coordinates are specified. This is because `graphdrawing` takes care of the exact layout, although it is possible to add certain options as hints for how you want the graph to look.

It is possible to install even more specialised graph drawing libraries with `\usegdlibrary{}` and passing a list of libraries to the command, e.g. `\usegdlibrary{trees, force}`.