Visualization

SparseMatrixColorings provides some internal utilities for visualization of matrix colorings via the un-exported function `show_colors`.

> **Warning**
>
> This function makes use of the Julia Images ecosystem. Using it requires loading ColorTypes.jl.
>
> We recommend loading the full Images.jl package for convenience, which includes ColorTypes.jl.

Basic usage

To obtain a visualization, simply call `show_colors` on a coloring result:

```
using Images
using SparseMatrixColorings, SparseArrays
using SparseMatrixColorings: show_colors

S = sparse([
    0 0 1 1 0 1
    1 0 0 0 1 0
    0 1 0 0 1 0
    0 1 1 0 0 0
]);

problem = ColoringProblem(; structure=:nonsymmetric, partition=:column)
algo = GreedyColoringAlgorithm(; decompression=:direct)
result = coloring(S, problem, algo)
show_colors(result)
```

Customization

The visualization can be customized via keyword arguments. By setting `pad` to a value higher than `0`, gaps are added between matrix entries. The size of the entries can be customized via `scale`. This parameter is also useful to upscale PNGs for later saving (see below).

We recommend using the ColorSchemes.jl catalogue to customize the `colorscheme`. Finally, a background color can be passed via the `background` keyword argument. To obtain transparent backgrounds, use the `RGBA` type.

```
using ColorSchemes
julia_colors = ColorSchemes.julia
white = RGB(1, 1, 1)

show_colors(result; colorscheme=julia_colors, background=white, scale=5, pad=1)
```

> **Terminal support**
>
> Loading ImageInTerminal.jl will allow you to show the output of `show_colors` within your terminal.

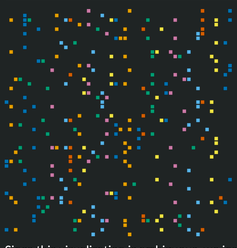
Working with large matrices

When working with large matrices, the default color scheme might be smaller than the number of matrix colors. In this case, a warning will be thrown, notifying you of a reuse of colors. To disable this warning, call `show_colors` with `warn=false`.

Let's demonstrate this by sampling a larger random matrix:

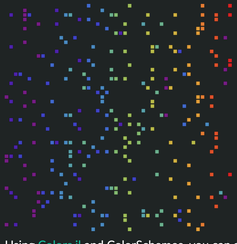
```
S = sprand(50, 50, 0.1) # sample sparse matrix

problem = ColoringProblem(; structure=:nonsymmetric, partition=:column)
algo = GreedyColoringAlgorithm(; decompression=:direct)
result = coloring(S, problem, algo)
show_colors(result; warn=false, scale=5, pad=1)
```



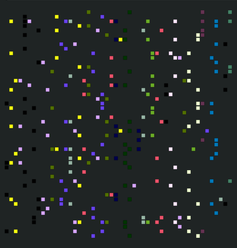
Since this visualization is ambiguous, we instead recomment subsampling a continuous colorscheme from ColorSchemes.jl:

```
ncolors = maximum(column_colors(result)) # for partition=:column
colorscheme = get(ColorSchemes.rainbow, range(0.0, 1.0, length=ncolors))
show_colors(result; colorscheme=colorscheme, scale=5, pad=1)
```



Using Colors.jl and ColorSchemes, you can also generate `distinguishable_colors`:

```
using Colors, ColorSchemes
colorscheme = distinguishable_colors(ncolors, transform=protanopic)
show_colors(result; colorscheme=colorscheme, scale=5, pad=1)
```



Saving images

The Julia Images ecosystem requires you to load a separate package to save images. ImageIO.jl is one of several options for PNG files:

```
using ImageIO

img = show_colors(result)
save("coloring.png", img)
```

Refer to the Julia Images documentation for more information.