Palo: Spatially-aware color palette optimization for single-cell and spatial data

Wenpin Hou, Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health
Zhicheng Ji, Department of Biostatistics and Bioinformatics, Duke University School of Medicine

Introductions

In the exploratory data analysis of single-cell or spatial genomic data, single cells or spatial spots are often visualized using a two-dimensional plot where each cluster is marked with a different color. With tens of clusters, current visualization methods will often result in visually similar colors assigned to spatially neighbouring clusters, making it hard to distinguish and identify the boundary between clusters. To address this issue, we developed Palo that optimizes the color palette assignment for single-cell and spatial data in a spatially aware manner. Palo identifies pairs of clusters that are spatially neighbouring to each other, and assigns visually different colors to those neighbouring clusters. We demonstrate that Palo results in better visualization in real single-cell and spatial genomic datasets.

Load packages

```
library(Palo)
library(ggplot2)
```

The following two packages are loaded here only because they are used in this vignette. Users can skip loading these two packages if they are not used.

```
library(Seurat)
library(SeuratData)
```

Single-cell RNA-seq example

We demonstrate Palo on an example single-cell RNA-seq dataset of T cell subsets described in the manuscript. The dataset has been further subsetted to reduce its size.

First, load the data, which is a data.frame.

```
d <- readRDS(paste0(system.file('data',package = 'Palo'),'/sc.rds'))
```

Get the UMAP coordinates. If the data is a Seurat object, the UMAP coordinates can be obtained by 'd[["umap"]]@cell.embeddings'

```
u <- d[,c('UMAP_1','UMAP_2')]
```

Get the cell clusters. If the data is a Seurat object, the cluster information can be obtained by 'Idents(d)'

```
cl <- as.character(d$CellType)
```

Generate a color palette which is used by ggplot2. Colorblind friendly palettes can be generated using the RColorbrewer package.

```
gg_color_hue <- function(n) {
  hues = seq(30, 500, length = n + 1)
  hcl(h = hues, l = 65, c = 100)[1:n]
}
pal <- gg_color_hue(length(unique(cl)))
```

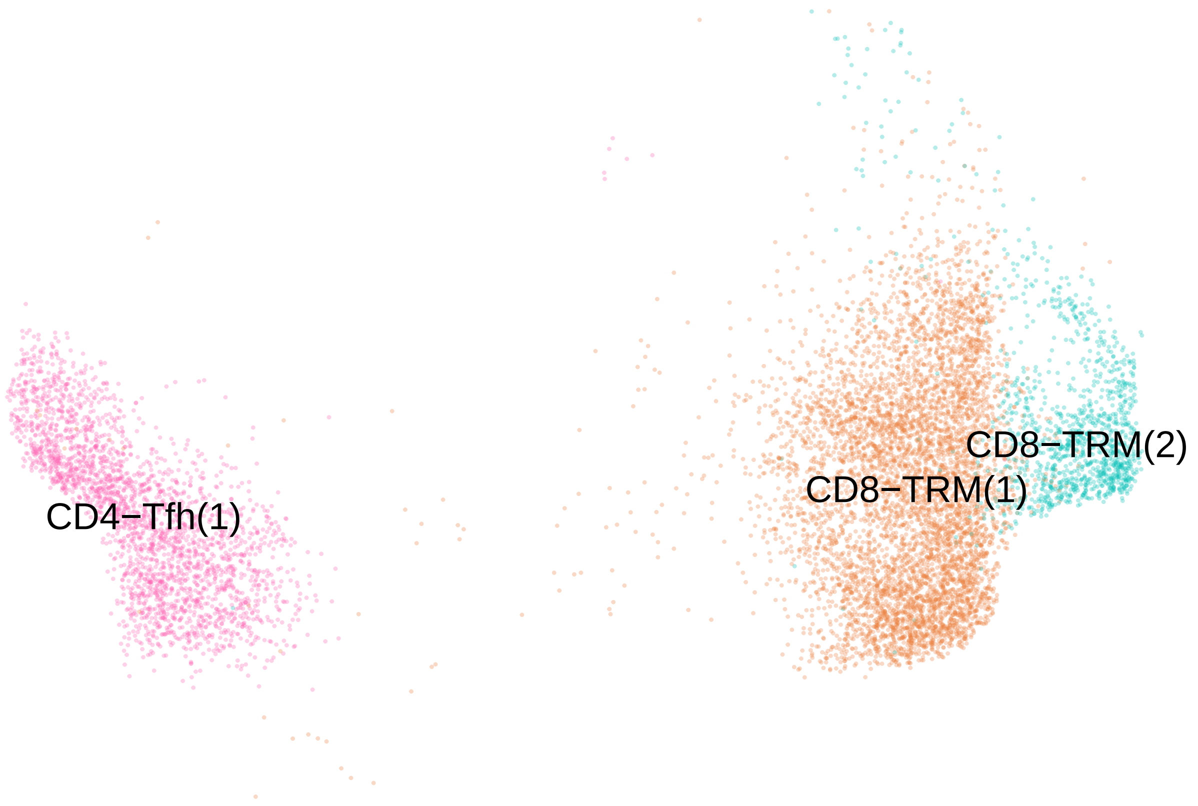
Run Palo to get the optimized palette:

```
palopal <- Palo(u,cl,pal)
palopal
```

```
## CD8-TRM(1) CD4-Tfh(1) CD8-TRM(2)
##  "#ED813E"  "#FF64B4"  "#00C0B6"
```

Visualize the UMAP with the Palo palette using ggplot2.

```
med <- aggregate(u,list(cl),median)
colnames(med)[1] <- 'CellType'
ggplot() +
  geom_point(data=d,aes(x=UMAP_1,y=UMAP_2,col=CellType),size=0.1,alpha=0.3) +
  geom_text(data=med,aes(x=UMAP_1,y=UMAP_2,label=CellType),size=5) +
  theme_void() +
  scale_color_manual(values=palopal) +
  theme(legend.position = 'none')
```



Palo accepts different weights for RGB colors when calculating color distances. This can make a difference with a large number of colors. Some typical choices of weights include (3,4,2) and (2,4,3). For example, to run Palo with RGB weights of (3,4,2).

```
palopal <- Palo(u,cl,pal,rgb_weight=c(3,4,2))
palopal
```

```
## CD8-TRM(1) CD4-Tfh(1) CD8-TRM(2)
##  "#FF64B4"  "#ED813E"  "#00C0B6"
```

Finally, Palo also accommodates colorblind-friendly visualizations. Palo will first convert the colors for colorblindness visualizations, and calculate the color distances using the converted colors.

```
palopal <- Palo(u,cl,pal,color_blind_fun='deutan')
palopal
```

```
## CD8-TRM(1) CD4-Tfh(1) CD8-TRM(2)
##  "#ED813E"  "#FF64B4"  "#00C0B6"
```

Spatial transcriptomics example

We demonstrate Palo on an example spatial transcriptomics data of mouse brain described in the manuscript.

Load the data, which is a Seurat object.

```
d <- readRDS(paste0(system.file('data',package = 'Palo'),'/spatial.rds'))
```

Get the spatial coordinates.

```
u <- d$anterior1@coordinates[,c(2,3)]
```

Get the spot clusters.

```
cl <- Idents(d)
```

Generate a color palette which is used by ggplot2.

```
gg_color_hue <- function(n) {
  hues = seq(15, 375, length = n + 1)
  hcl(h = hues, l = 65, c = 100)[1:n]
}
pal <- gg_color_hue(length(unique(cl)))
```

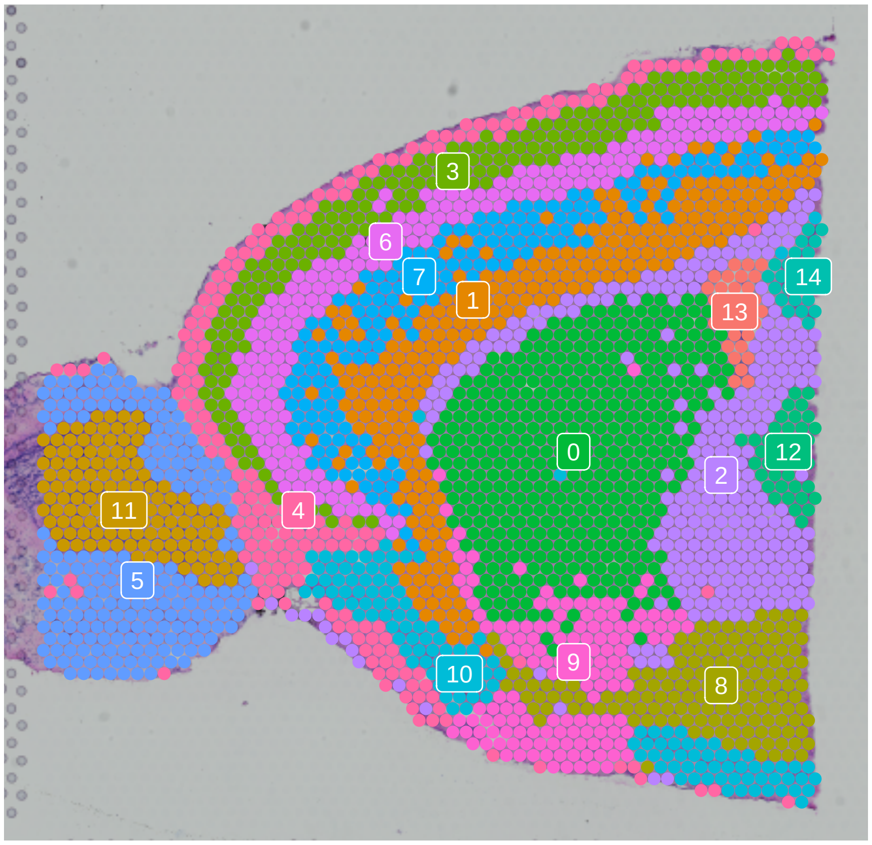
Run Palo to get the optimized palette. Here we increased the number of optimization iterations (by increasing refine_iter and early_stop) to improve the performance.

```
palopal <- Palo(u,cl,pal,refine_iter=2000,early_stop = 1000)
palopal
```

```
##         2         5         3        11         8         4         9         0
## "#B983FF" "#619CFF" "#6BB100" "#C99800" "#A3A500" "#FF67A4" "#FD61D1" "#00BA38"
##        10         1         6        14         7        12        13
## "#00BCD8" "#E58700" "#E76BF3" "#00C0AF" "#00B0F6" "#00BF7D" "#F8766D"
```

Generate the spatial plot with Palo palette using the SpatialDimPlot function by Seurat.

```
SpatialDimPlot(d, label = TRUE, label.size = 3,stroke=NA) +
  scale_fill_manual(values=palopal) + theme(legend.position='none')
```



Session Info

```
sessionInfo()
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS  10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] SeuratData_0.2.1   SeuratObject_4.0.0 Seurat_4.0.0       ggplot2_3.3.3
## [5] Palo_1.1
##
## loaded via a namespace (and not attached):
##   [1] nlme_3.1-148          matrixStats_0.57.0    RcppAnnoy_0.0.18
##   [4] RColorBrewer_1.1-2    httr_1.4.2            sctransform_0.3.2
##   [7] tools_4.0.2           utf8_1.2.1            R6_2.5.0
##  [10] irlba_2.3.3          rpart_4.1-15         KernSmooth_2.23-17
##  [13] uwot_0.1.10          mgcv_1.8-33          DBI_1.1.0
##  [16] lazyeval_0.2.2       colorspace_2.0-1     withr_2.4.2
##  [19] tidyselect_1.1.0     gridExtra_2.3        compiler_4.0.2
##  [22] cli_2.5.0            plotly_4.9.2.1       labeling_0.4.2
##  [25] scales_1.1.1         spatstat.data_2.0-0  lmtest_0.9-37
##  [28] ggridges_0.5.2       pbapply_1.4-2        rappdirs_0.3.1
##  [31] goftest_1.2-2        spatstat_1.64-1      stringr_1.4.0
##  [34] digest_0.6.27        spatstat.utils_2.0-0 rmarkdown_2.7
##  [37] pkgconfig_2.0.3      htmltools_0.5.1.1    highr_0.8
##  [40] fastmap_1.0.1        htmlwidgets_1.5.1    rlang_0.4.11
##  [43] shiny_1.5.0          farver_2.1.0         generics_0.1.0
##  [46] zoo_1.8-8            jsonlite_1.7.1       ica_1.0-2
##  [49] dplyr_1.0.2          magrittr_2.0.1       patchwork_1.0.1
##  [52] Matrix_1.2-18        Rcpp_1.0.5           munsell_0.5.0
##  [55] fansi_0.4.2          abind_1.4-5          reticulate_1.16
##  [58] lifecycle_1.0.0      stringi_1.5.3        yaml_2.2.1
##  [61] MASS_7.3-51.6        Rtsne_0.15           plyr_1.8.6
##  [64] grid_4.0.2           parallel_4.0.2       listenv_0.8.0
##  [67] promises_1.1.1       ggrepel_0.8.2        crayon_1.4.1
##  [70] deldir_0.2-10        miniUI_0.1.1.1       lattice_0.20-41
##  [73] cowplot_1.1.0        splines_4.0.2        tensor_1.5
##  [76] knitr_1.33           pillar_1.6.1         igraph_1.2.5
##  [79] future.apply_1.6.0   reshape2_1.4.4       codetools_0.2-16
##  [82] leiden_0.3.3         glue_1.4.2           evaluate_0.14
##  [85] data.table_1.13.2    vctrs_0.3.8          png_0.1-7
##  [88] httpuv_1.5.4         polyclip_1.10-0      gtable_0.3.0
##  [91] RANN_2.6.1           purrr_0.3.4          tidyr_1.1.2
##  [94] scattermore_0.7      future_1.17.0        xfun_0.22
##  [97] mime_0.9             xtable_1.8-4         later_1.1.0.1
## [100] survival_3.1-12      viridisLite_0.4.0    tibble_3.1.2
## [103] cluster_2.1.0        globals_0.12.5       fitdistrplus_1.1-1
## [106] ellipsis_0.3.2       ROCR_1.0-11
```