Scaling up reproducible research for single cell transcriptomics using MetaNeighbor (Protocol 1)

Protocol 1: assessment of cell type replicability with unsupervised MetaNeighbor

Protocol 1 demonstrates how to compute and visualize cell type replicability across 4 human pancreas datasets. We will show steps detailing how to install MetaNeighbor, how to download and reformat the datasets with the SingleCellExperiment package, how to compute and interpret MetaNeighbor AUROCs. All code blocks can be run in R command line, Rstudio, RMarkdown notebooks or a jupyter notebook with an R kernel.

Step 0: Installation of MetaNeighbor and packages used in the protocol (1-10 minutes)

Important: the installation process may create conflicts in the notebook environment. We recommend running 1. and 2. in a separate R shell or restarting the R session associated with the notebook after 2.

1. We start by installing the latest MetaNeighbor package from the Gillis lab GitHub page.

```r
if (!require("devtools")) {
  install.packages("devtools", quiet=TRUE)
}
devtools::install_github("gillislab/MetaNeighbor")
```

Note that the latest stable version of MetaNeighbor is also available through Bioconductor by running *BiocManager::install(“MetaNeighbor”)*. As some of the functionalities illustrated here have not yet been pushed to Bioconductor (they will be available in the next release, November 2020), we installed the latest development version from GitHub.

2. Next, we install the following packages, which are not necessary to run MetaNeighbor itself, but are needed to run the protocol.

```r
to_install = c("scRNAseq", "tidyverse", "org.Hs.eg.db", "UpSetR")
installed = sapply(to_install, requireNamespace)
if (sum(!installed) > 0) {
    if (!requireNamespace("BiocManager", quietly = TRUE)) {
        install.packages("BiocManager")
        BiocManager::install()
    }
    BiocManager::install(to_install[!installed])
}
```

Reminder: don’t forget to restart the R session at this stage.

Step 1: creation of a merged SingleCellExperiment dataset (1-2 minutes)

3. We consider 4 pancreatic datasets along with their independent annotation (from the original publications). MetaNeighbor expects a gene-by-cell matrix encapsulated in a SummarizedExperiment format. We recommend the SingleCellExperiment (SCE) package, because it is able to handle sparse matrix formats. We load the pancreas datasets using the scRNAseq package, which provide annotated datasets that are already in the SingleCellExperiment format.

```
library(scRNAseq)
my_data <- list(
    baron = BaronPancreasData(),
    lawlor = LawlorPancreasData(),
    seger = SegerstolpePancreasData(),
    muraro = MuraroPancreasData()
)
```

Note that Seurat objects can easily be converted into SingleCellExperiment objects by using Seurat's *Convert* function, e.g. *Convert(from = seurat_object, to = "sce")*.

4. MetaNeighbor's *mergeSCE* function can be used to merge multiple SingleCellExperiment objects. Importantly, the output object will be restricted to genes, metadata columns and assays that are common to all datasets. Before we use *mergeSCE*, we need to make sure that gene and metadata information aligns across datasets.

We start by checking if gene information aligns (stored in the *rownames* slot of the SCE object).

```
lapply(my_data, function(x) head(rownames(x), 3))
```

```
## $baron
## [1] "A1BG" "A1CF" "A2M"
##
## $lawlor
## [1] "ENSG00000229483" "ENSG00000232849" "ENSG00000229558"
##
## $seger
## [1] "SGIP1" "AZIN2" "CLIC4"
##
## $muraro
## [1] "A1BG-AS1__chr19" "A1BG__chr19"     "A1CF__chr10"
```

Two datasets (Baron, Segerstolpe) use gene symbols, one dataset (Muraro) combines symbols with chromosome information (to avoid duplicate gene names) and the last dataset (Lawlor) uses Ensembl identifiers. Here we will convert all gene names to unique gene symbols. We start by converting gene names in the Muraro dataset by using the symbols stored in the *rowData* slot of the SCE object, and remove all duplicated gene symbols.

```
rownames(my_data$muraro) <- rowData(my_data$muraro)$symbol
my_data$muraro <- my_data$muraro[!duplicated(rownames(my_data$muraro)),]
```

Next, we convert Ensembl IDs to gene symbols in the Lawlor dataset, removing all IDs with no match and all duplicated symbols.

```
library(org.Hs.eg.db)

symbols <- mapIds(org.Hs.eg.db, keys=rownames(my_data$lawlor),
                  keytype="ENSEMBL", column="SYMBOL")
keep <- !is.na(symbols) & !duplicated(symbols)
my_data$lawlor <- my_data$lawlor[keep,]
rownames(my_data$lawlor) <- symbols[keep]
```

5. We now turn our attention to metadata, which are stored in the *colData* slot of the SCE objects. Here we need to make sure that the column that contains cell type information is labeled identically in all datasets.

```
lapply(my_data, function(x) colnames(colData(x)))
```

```
## $baron
## [1] "donor" "label"
##
## $lawlor
## [1] "title"         "age"           "bmi"           "cell type"
## [5] "disease"       "islet unos id" "race"          "Sex"
##
## $seger
## [1] "Source Name"               "individual"
## [3] "single cell well quality" "cell type"
## [5] "disease"                   "sex"
## [7] "age"                       "body mass index"
##
## $muraro
## [1] "label" "donor" "plate"
```

Two datasets have the cell type information in the "cell type" column, the other two in the "label" column. We add a "cell type" column in the latter two datasets.

```
my_data$baron$"cell type" <- my_data$baron$label
my_data$muraro$"cell type" <- my_data$muraro$label
```

6. Last, we check that count matrices, stored in the *assays* slot, have identical names.

```
lapply(my_data, function(x) names(assays(x)))
```

```
## $baron
## [1] "counts"
##
## $lawlor
## [1] "counts"
##
## $seger
## [1] "counts"
##
## $muraro
## [1] "counts"
```

The count matrices are all stored in an assay named "counts", no change is needed here.

7. Now that gene, cell type and count matrix information is aligned across datasets, we can create a merged dataset. *mergeSCE* takes a list of SCE objects as an input and outputs a single SCE object.

```
library(MetaNeighbor)

fused_data = mergeSCE(my_data)
dim(fused_data)
```

```
## [1] 15295 15793
```

```
head(colData(fused_data))
```

```
## DataFrame with 6 rows and 2 columns
##                                 cell type    study_id
##                               <character> <character>
## human1_lib1.final_cell_0001      acinar       baron
## human1_lib1.final_cell_0002      acinar       baron
## human1_lib1.final_cell_0003      acinar       baron
## human1_lib1.final_cell_0004      acinar       baron
## human1_lib1.final_cell_0005      acinar       baron
## human1_lib1.final_cell_0006      acinar       baron
```

The new dataset contains 15,295 common genes, 15,793 cells and two metadata columns: a concatenated "cell type" column, and "study_id", a column created by *mergeSCE* containing the name of the original studies (corresponding to the names provided in the "my_data" list).

8. To avoid having to recreate the merged object, we recommend saving it as an RDS file.

```r
saveRDS(fused_data, "merged_pancreas.rds")
```

Step 2: Hierarchical cell type replicability analysis (1 minute)

9. We start by loading the MetaNeighbor (analysis) and the SingleCellExperiment (data handling) libraries, as well as the previously created pancreas dataset.

```r
library(MetaNeighbor)
library(SingleCellExperiment)

pancreas_data = readRDS("merged_pancreas.rds")
```

10. To perform neighbor voting and identify replicating cell types, MetaNeighbor builds a cell-cell similarity network, which we defined as the Spearman correlation over a user-defined set of genes. We found that we obtained best results by picking genes that are highly variable across datasets, which can be picked using the *variableGenes* function.

```r
global_hvgs = variableGenes(dat = pancreas_data, exp_labels = pancreas_data$study_id)
length(global_hvgs)
```

```
## [1] 600
```

The function returns a list of 600 genes that were detected as highly variable in each of the 4 datasets. In our experience, we obtained best performance for gene sets ranging from 200 to 1,000 variable genes. If the *variableGenes* returns a gene set that is too small (in particular when you are comparing a large number of datasets), the number of genes can be increased by setting the "min_recurrence" parameter. For example, by setting "min_recurrence=2", we would keep all genes that are highly variable in at least 2 ot of the 4 datasets.

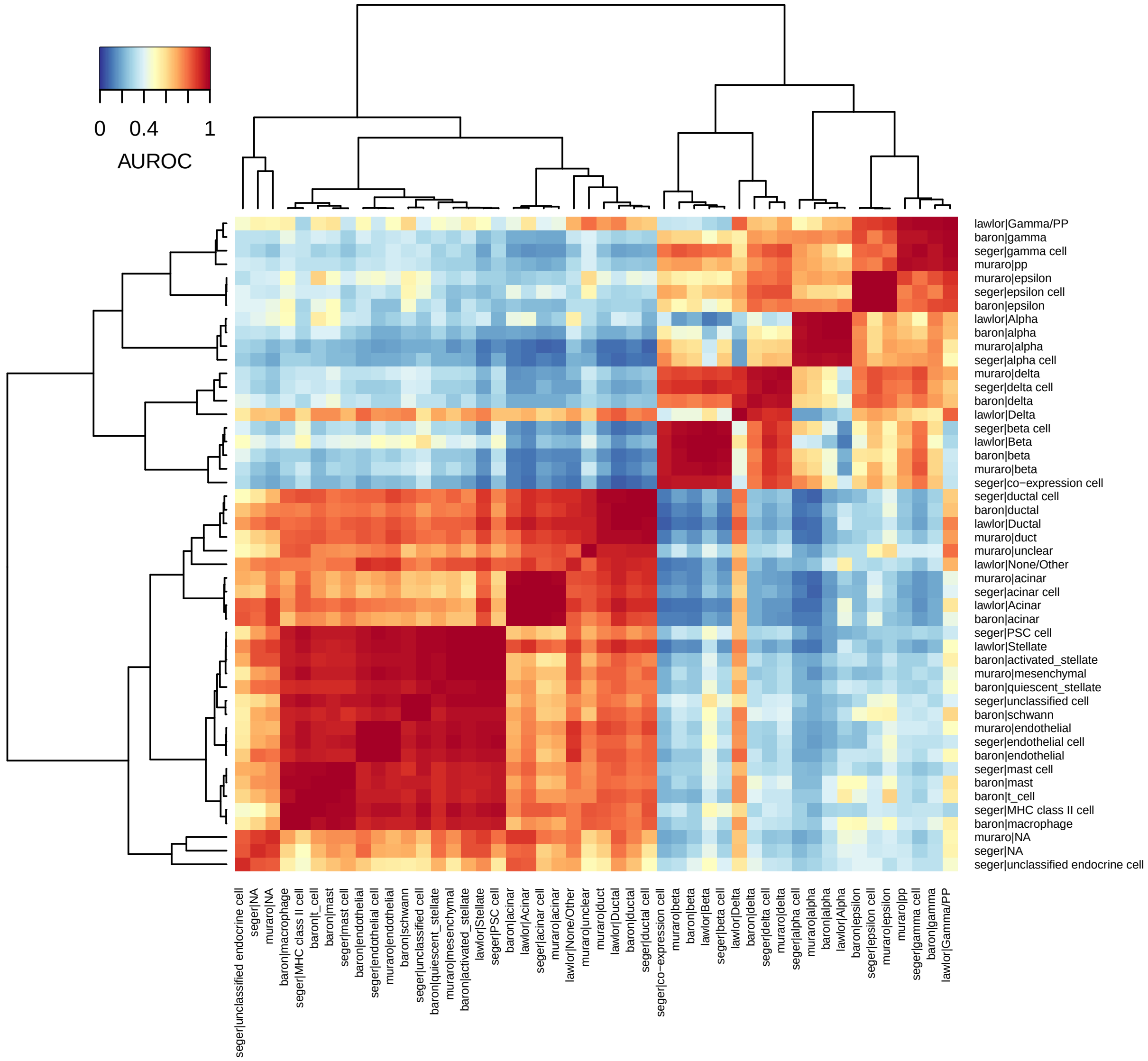
11. The merged dataset and a set of biological meaningful genes is all we need to run MetaNeighbor and obtain cell type similarities. Because the dataset is large (> 10k cells), we use the fast implementation of MetaNeighbor ("fast_version=TRUE").

```r
aurocs = MetaNeighborUS(var_genes = global_hvgs,
                        dat = pancreas_data,
                        study_id = pancreas_data$study_id,
                        cell_type = pancreas_data$"cell type",
                        fast_version = TRUE)
```

MetaNeighborUS returns a cell-type-by-cell-type matrix containing cell type similarities. Cell type similarities are defined as an Area Under the ROC curve (AUROC), which range between 0 and 1, where 0 indicates low similarity and 1 high similarity.

12. For ease of interpretation, we visualize AUROCs as a heatmap, where rows and columns are cell types from all the datasets.

```
plotHeatmap(aurocs, cex = 0.5)
```



In the heatmap, the color of each square indicates the proximity of a pair of cell types, ranging from blue (low similarity) to red (high similarity). For example, "baron|gamma" (2nd row) is highly similar to "seger|gamma" (3rd column from the right) but very dissimilar from "muraro|duct" (middle column). To group similar cell types together, *plotHeatmap* applies hierarchical clustering on the AUROC matrix. On the heatmap, we see two large red blocks that indicate hierarchical structure in the data, with endocrine cell types clustering together (e.g., alpha, beta, gamma) and non-endocrine cells on the other side (e.g., amacrine, ductal, endothelial). Note that each red block is composed of smaller red blocks, indicating that cell types can be matched at an even higher resolution. The presence of off-diagonal patterns (e.g., "lawlor|Gamma/PP", "lawlor|Delta") suggests the presence of doublets or contamination, but the heatmap is dominated by the clear presence of red blocks, which is a strong indicator of replicability.

13. To identify pairs of replicable cell types, we rely on a simple heuristic: a pair of cell type is replicable if they are reciprocal top hits (they preferentially vote for each other) and the AUROC exceeds a given

threshold value (in our experience, 0.9 is a good heuristic value).

```r
topHits(aurocs, dat = pancreas_data, study_id = pancreas_data$study_id,
        cell_type = pancreas_data$"cell type", threshold = 0.9)
```

Study_ID|Celltype_1	Study_ID|Celltype_2	Mean_AUROC	Match_type
seger|epsilon cell	muraro|epsilon	1.00	Reciprocal_top_hit
seger|epsilon cell	baron|epsilon	1.00	Above_0.9
baron|mast	seger|mast cell	1.00	Reciprocal_top_hit
seger|endothelial cell	muraro|endothelial	1.00	Reciprocal_top_hit
lawlor|Stellate	seger|PSC cell	1.00	Reciprocal_top_hit
baron|macrophage	seger|MHC class II cell	1.00	Reciprocal_top_hit
muraro|endothelial	baron|endothelial	1.00	Above_0.9
lawlor|Stellate	baron|activated_stellate	1.00	Above_0.9
baron|acinar	lawlor|Acinar	1.00	Reciprocal_top_hit
seger|PSC cell	muraro|mesenchymal	1.00	Above_0.9
baron|alpha	lawlor|Alpha	1.00	Reciprocal_top_hit
lawlor|Acinar	seger|acinar cell	1.00	Above_0.9
baron|schwann	seger|unclassified cell	1.00	Reciprocal_top_hit
seger|acinar cell	muraro|acinar	0.99	Above_0.9
lawlor|Beta	seger|beta cell	0.99	Reciprocal_top_hit
baron|ductal	seger|ductal cell	0.99	Reciprocal_top_hit
lawlor|Beta	baron|beta	0.99	Above_0.9
baron|ductal	lawlor|Ductal	0.99	Above_0.9
seger|MHC class II cell	baron|t_cell	0.99	Above_0.9
baron|gamma	lawlor|Gamma/PP	0.99	Reciprocal_top_hit
lawlor|Beta	muraro|beta	0.98	Above_0.9
seger|ductal cell	muraro|duct	0.98	Above_0.9
lawlor|Alpha	muraro|alpha	0.98	Above_0.9
seger|PSC cell	baron|quiescent_stellate	0.98	Above_0.9
lawlor|Gamma/PP	seger|gamma cell	0.98	Above_0.9
seger|delta cell	muraro|delta	0.98	Reciprocal_top_hit
lawlor|Gamma/PP	muraro|pp	0.98	Above_0.9
muraro|alpha	seger|alpha cell	0.98	Above_0.9
muraro|delta	baron|delta	0.96	Above_0.9
baron|beta	seger|co-expression cell	0.95	Above_0.9
seger|ductal cell	muraro|unclear	0.93	Above_0.9
baron|delta	lawlor|Delta	0.92	Above_0.9
baron|ductal	lawlor|None/Other	0.91	Above_0.9

We find a long list of replicable endocrine cell types (e.g., epsilon, alpha and beta cells) and non-endocrine cell types (e.g. mast, endothelial or acinar cells). This list provides strong evidence that these cell types are robust, as they are identified across all datasets with high AUROC.

14. In the case where there is a clear structure in the data (endocrine vs non-endocrine here), we can refine AUROCs by splitting the data. AUROCs have a simple interpretation: an AUROC of 0.6 indicates that cells from a given cell type are ranked in front of 60% of other test cells. However, this interpretation is outgroup dependent: because endocrine cells represent ~65% of cells, even an unrelated pair of non-endocrine cell types will have an AUROC > 0.65, because non-endocrine cells will always be ranked in front of endocrine cells.

By starting with the full datasets, we uncovered the global structure in the data (endocrine vs non-endocrine). However, to evaluate replicability of endocrine cell types and reduce dataset composition effects, we can make the assessment more stringent by restricting the outgroup to close cell types, i.e. by keeping only endocrine subtypes. We split cell types in two by using the *splitClusters* function and retain only endocrine cell types.

```
level1_split = splitClusters(aurocs, k = 2)
level1_split
```

```
## $`1`
##  [1] "baron|acinar"                    "baron|activated_stellate"
##  [3] "baron|ductal"                    "baron|endothelial"
##  [5] "baron|macrophage"                "baron|mast"
##  [7] "baron|quiescent_stellate"        "baron|schwann"
##  [9] "baron|t_cell"                    "lawlor|Acinar"
## [11] "lawlor|Ductal"                   "lawlor|None/Other"
## [13] "lawlor|Stellate"                 "seger|acinar cell"
## [15] "seger|ductal cell"               "seger|endothelial cell"
## [17] "seger|mast cell"                 "seger|MHC class II cell"
## [19] "seger|NA"                        "seger|PSC cell"
## [21] "seger|unclassified cell"         "seger|unclassified endocrine cell"
## [23] "muraro|acinar"                   "muraro|duct"
## [25] "muraro|endothelial"              "muraro|mesenchymal"
## [27] "muraro|NA"                       "muraro|unclear"
##
## $`2`
##  [1] "baron|alpha"              "baron|beta"
##  [3] "baron|delta"              "baron|epsilon"
##  [5] "baron|gamma"              "lawlor|Alpha"
##  [7] "lawlor|Beta"              "lawlor|Delta"
##  [9] "lawlor|Gamma/PP"          "seger|alpha cell"
## [11] "seger|beta cell"          "seger|co-expression cell"
## [13] "seger|delta cell"         "seger|epsilon cell"
## [15] "seger|gamma cell"         "muraro|alpha"
## [17] "muraro|beta"              "muraro|delta"
## [19] "muraro|epsilon"           "muraro|pp"
```

```
first_split = level1_split[[2]]
```

By outputting “level1_split”, we found that the cell types were nicely split between non-endocrine and endocrine, and that endocrine cell types where in the second element of the list. Note that *splitClusters* applies a simple hierarchical clustering algorithm to separate cell types, cell types can be selected manually in more complex scenarios.

15. We repeat the MetaNeighbor analysis on endocrine cells only. First, we subset the data to the endocrine cell types that we previously stored in “first_split”.

```
full_labels = makeClusterName(pancreas_data$study_id, pancreas_data$"cell type")
subdata = pancreas_data[, full_labels %in% first_split]
dim(subdata)
```

```
## [1] 15295  9341
```

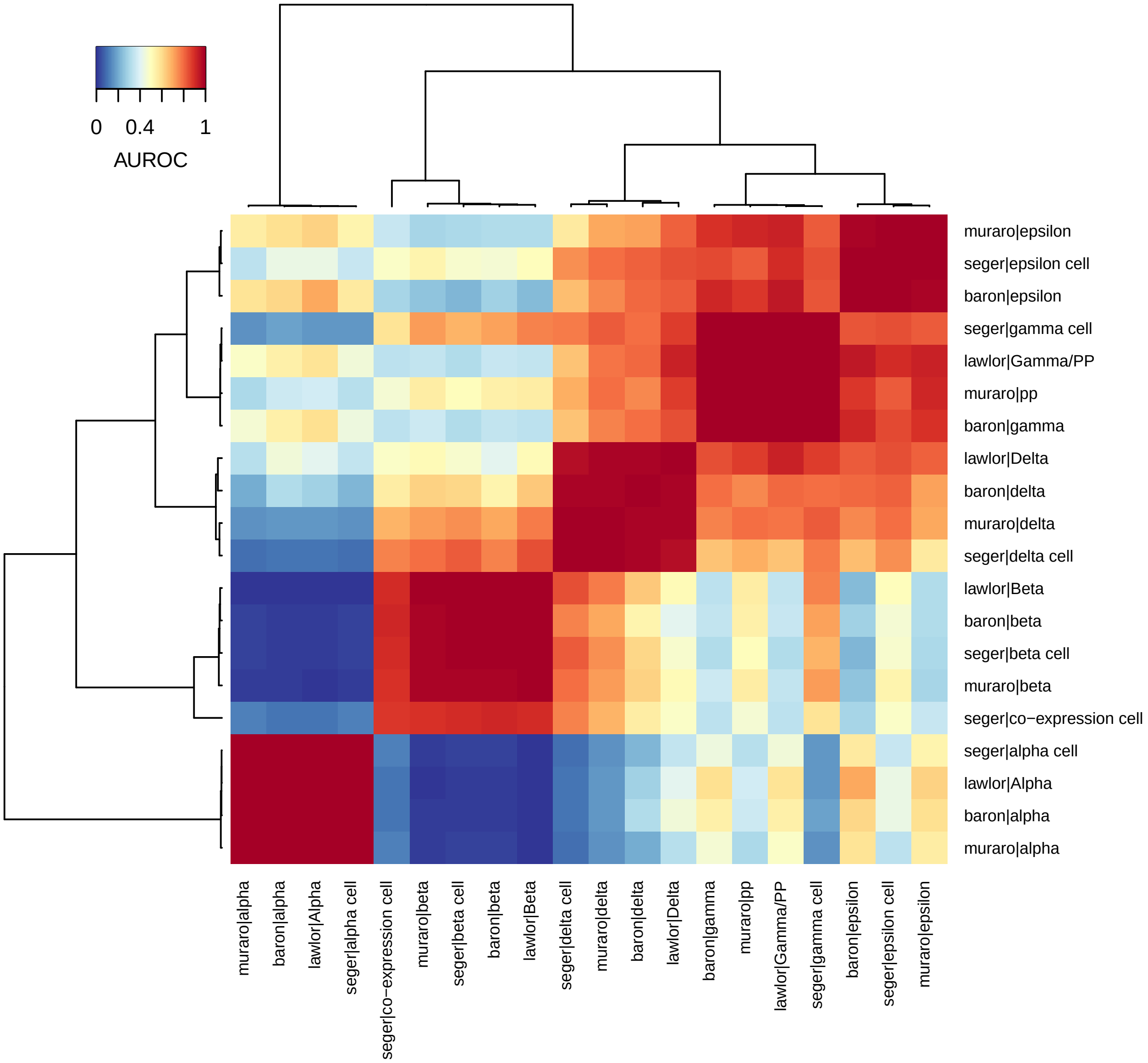
The new dataset contains the 9341 putative endocrine cells.

16. To focus on variability that is specific to endocrine cells, we re-pick highly variable genes.

```
var_genes = variableGenes(dat = subdata, exp_labels = subdata$study_id)
```

17. Finally, we recompute cell type similarities and visualize AUROCs.

```
aurocs = MetaNeighborUS(var_genes = var_genes,
                        dat = subdata, fast_version = TRUE,
                        study_id = subdata$study_id,
                        cell_type = subdata$"cell type")
plotHeatmap(aurocs, cex = 0.7)
```



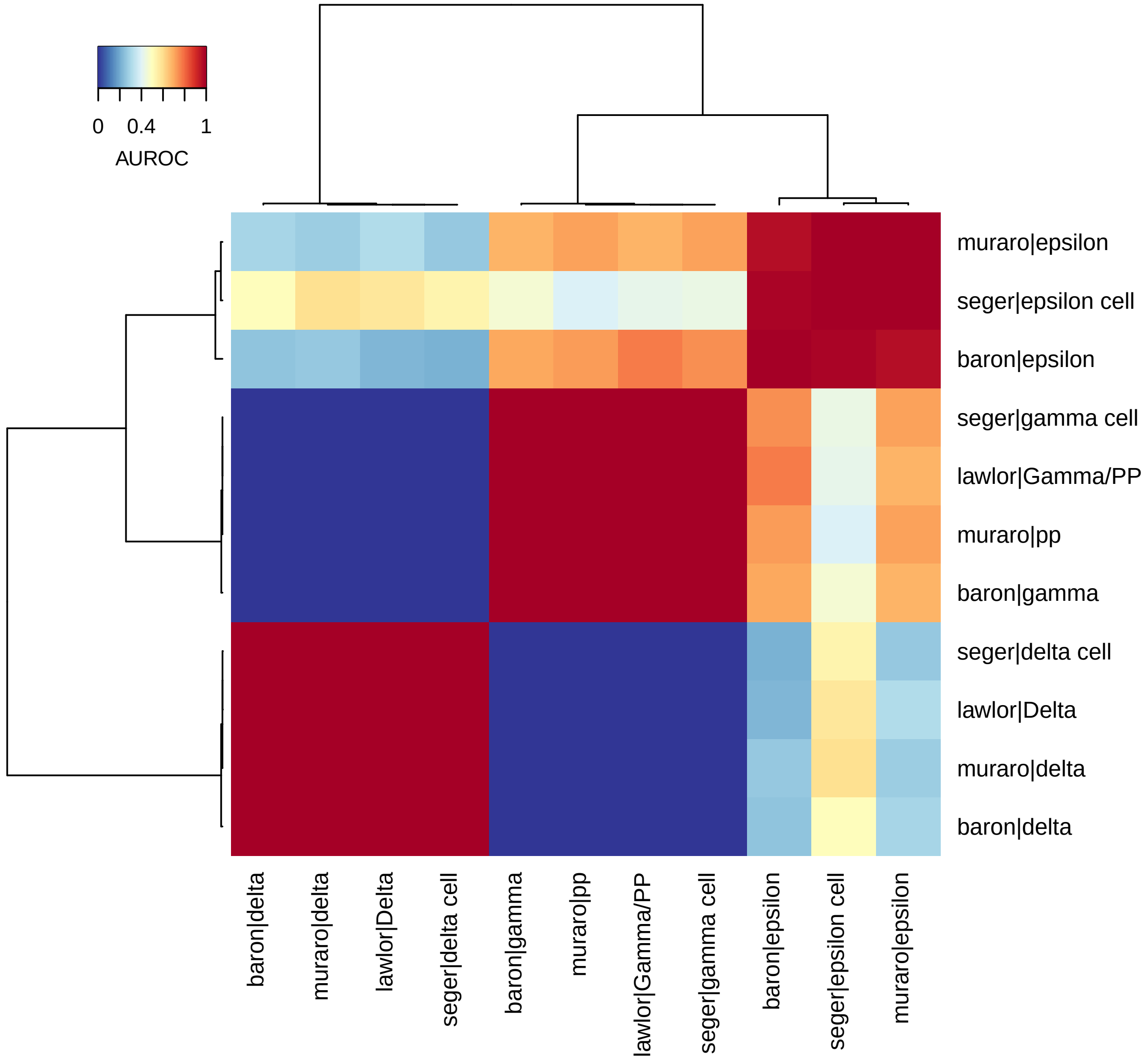
The resulting heatmap illustrates an example of a strong set of replicating cell types: when the assessment becomes more stringent (restriction to closely related cell types), the similarity of replicating cell types remains strong (AUROC~1 for alpha, beta, gamma, delta and epsilon cells) while the cross-cell-type similarity decreases (shift from red to blue, e.g. similarity of alpha and beta cell types has shifted from orange/red in the global heatmap to dark blue in the endocrine heatmap) by virtue of zooming in on a subpart of the dataset.

18. We can continue to zoom in as long as there are at least two cell types per dataset.

```
level2_split = splitClusters(aurocs, k = 3)
my_split = level2_split[[3]]
subdata = pancreas_data[, full_labels %in% my_split]
var_genes = variableGenes(dat = subdata, exp_labels = subdata$study_id)
length(var_genes)
```

```
## [1] 274
```

```
aurocs = MetaNeighborUS(var_genes = var_genes,
                        dat = subdata, fast_version = TRUE,
                        study_id = subdata$study_id,
                        cell_type = subdata$"cell type")
plotHeatmap(aurocs, cex = 1)
```



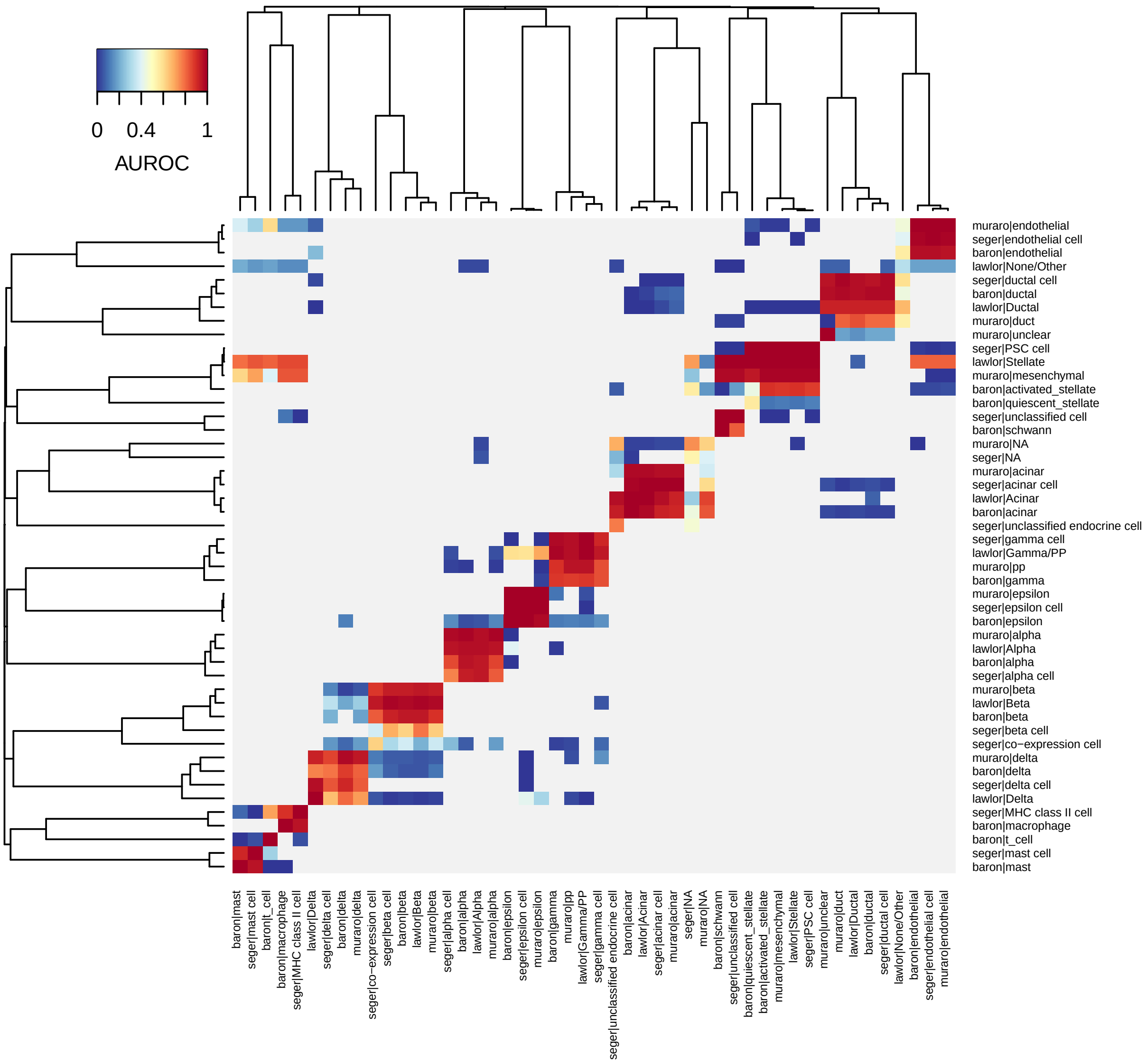
Here we remove the alpha and beta cells (representing close to 85% of endocrine cells) and validate that, even when restricting to neighboring cell types, there is still a clear distinction between delta, gamma and epsilon cells (AUROC ~ 1).

Step 3: stringent assessment of replicability with one-vs-best AUROCs (1 minute)

In the previous section, we created progressively more stringent replicability assessments by selecting more and more specific subsets of related cell types. As an alternative, we provide the “one_vs_best” parameter, which offers similar results without having to restrict the dataset manually. In this scoring mode, MetaNeighbor will automatically identify the two closest matching cell types in each test dataset and compute an AUROC based on the voting result for cells from the closest match against cells from the second closest match. Essentially, we are asking how easily a cell type can be distinguished from its closest neighbor.

19. To obtain one-vs-best AUROCs, we run the same command as before with two additional parameters: “one_vs_best = TRUE” and “symmetric_output = FALSE”.

```r
best_hits = MetaNeighborUS(var_genes = global_hvgs,
                           dat = pancreas_data,
                           study_id = pancreas_data$study_id,
                           cell_type = pancreas_data$"cell type",
                           fast_version = TRUE,
                           one_vs_best = TRUE, symmetric_output = FALSE)
plotHeatmap(best_hits, cex = 0.5)
```



The interpretation of the heatmap is slightly different compared to one-vs-all AUROCs. First, since we only compare the two closest cell types, most cell type combinations are not tested (NAs, shown in gray on the heatmap). Second, by setting "symmetric_output=FALSE", we broke the symmetry of the heatmap: train cell types are shown as columns and test cell types are shown as rows. Since each cell type is only tested against two cell types in each test dataset (closest and second closest match), we have 8 values per column (2 per dataset).

This representation helps to rapidly identify a cell type's closest hits as well as their closest outgroup. For example, ductal cells (2nd red square from the top right) strongly match with each other (one-vs-best AUROC>0.8) and acinar cells are their closest outgroup (blue segments in the same column). The non-symmetric view makes it clear when best hits are not reciprocal. For example, mast cells (first two columns) heavily vote for "lawlor|Stellate" and "muraro|mesenchymal", but this vote is not reciprocal. This pattern indicates that the mast cell type is missing in the Lawlor and Muraro datasets: because mast cells have no natural match in these datasets, they vote for the next closest cell type (stellate cells). The lack of reciprocity in voting is an important tool to detect imbalances in dataset composition.

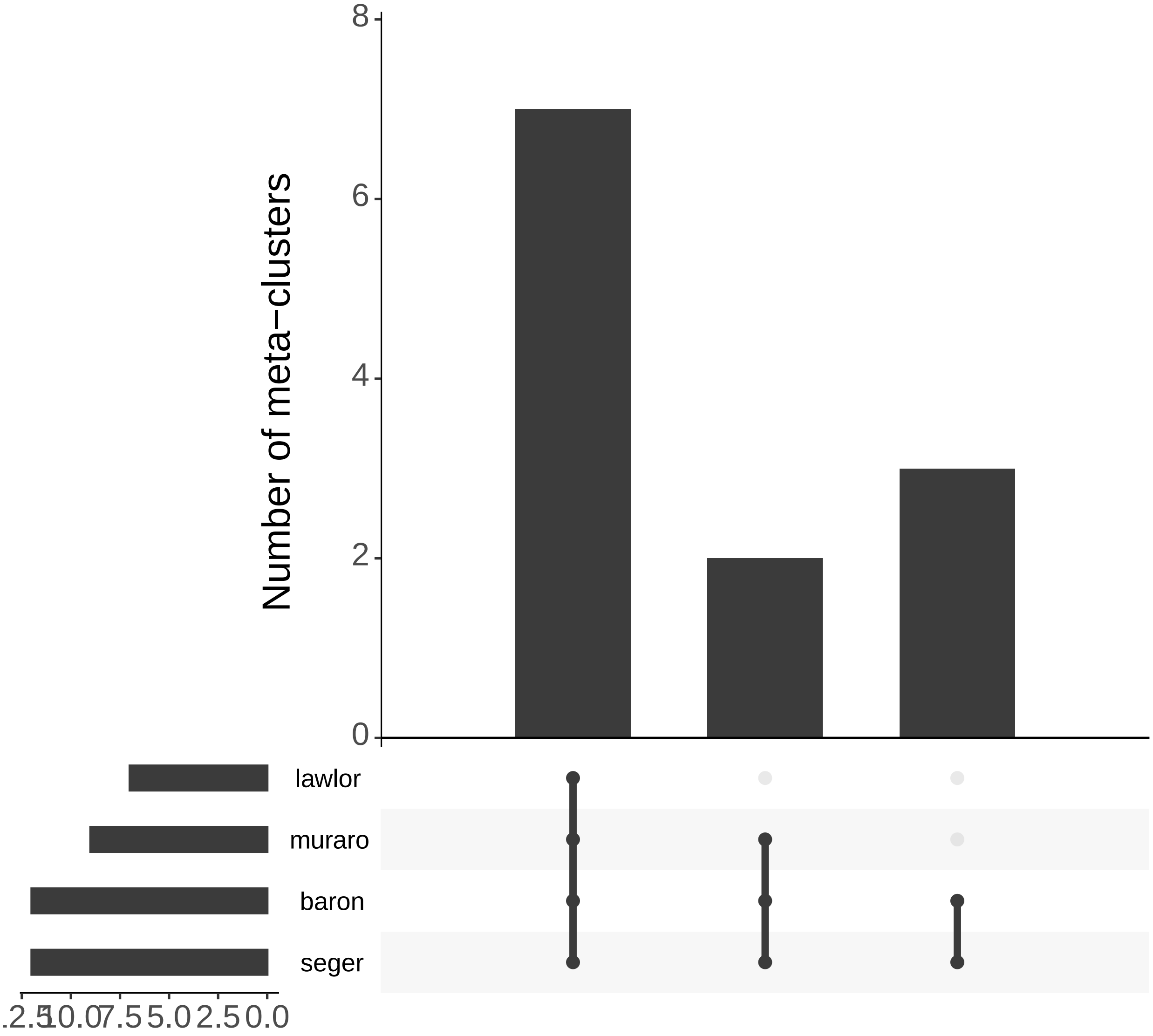
20. When using one-vs-best AUROCs, we recommend extracting replicating cell types as meta-clusters. Cell types are part of the same meta-cluster if they are reciprocal best hits. Note that if cell type A

is the reciprocal best hit of B and C, all three cell types are part of the same meta-cluster, even if B and C are not reciprocal best hits. To further filter for strongly replicating cell types, we specify an AUROC threshold (in our experience, 0.7 is a strong one-vs-best AUROC threshold).

```
mclusters = extractMetaClusters(best_hits, threshold = 0.7)
mcsummary = scoreMetaClusters(mclusters, best_hits)
```

The *scoreMetaClusters* provides a good summary of meta-clusters, ordering cell types by the number of datasets in which they replicate, then by average AUROC. We find 12 cell types that have strong support across at least 2 datasets, with 7 cell types replicating across all 4 datasets. 8 cell types are tagged as “outlier”, indicating they had no strong match in any other dataset. These cell types usually contain doublets, low quality cells or contaminated cell types. To rapidly visualize the number of robust cell types, the replicability structure can be summarized as an Upset plot with the *plotUpset* function.

```
plotUpset(mclusters)
```



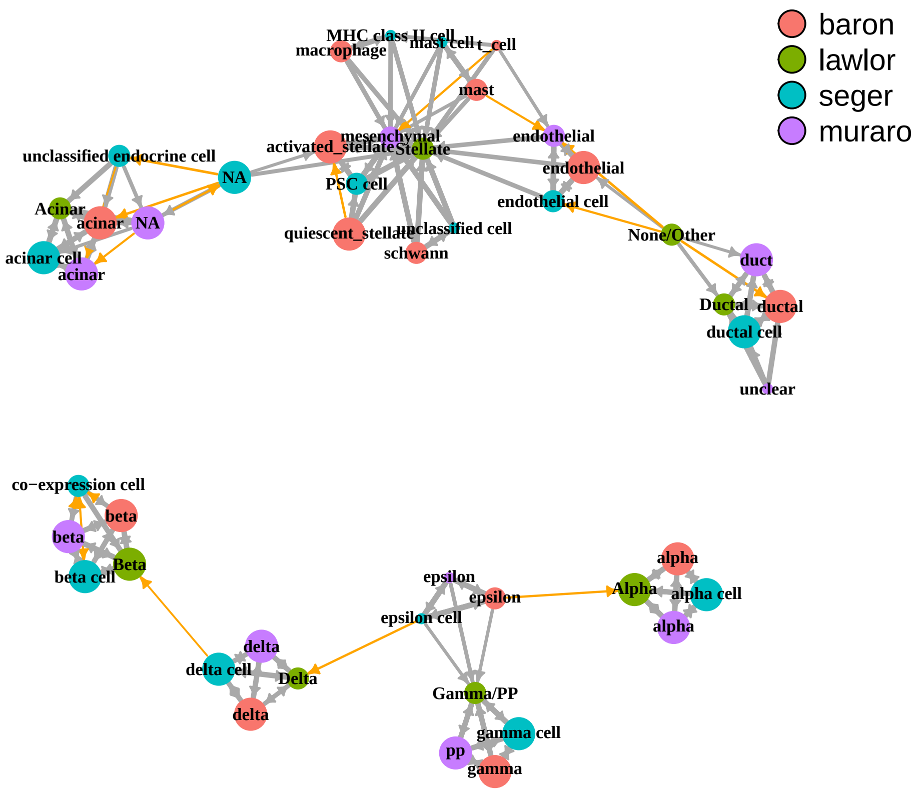
To further investigate the robustness of meta-clusters, they can be visualized as heatmaps (called “cell-type badges”) with the *plotMetaClusters* function. Because the function generates one heatmap per meta-cluster, we save the output to a PDF file to facilitate investigation. Each badge shows an AUROC heatmap restricted to one specific meta-cluster. These badges help diagnose cases where AUROCs are lower in a specific train or test dataset. For example, the "muraro|duct" cell type has systematically lower AUROCs, suggesting the presence of contaminating cells in another cell type (probably in the "muraro|unclear" cell type).

```
pdf("meta_clusters.pdf")
plotMetaClusters(mclusters, best_hits)
dev.off()
```

```
## pdf
##   2
```

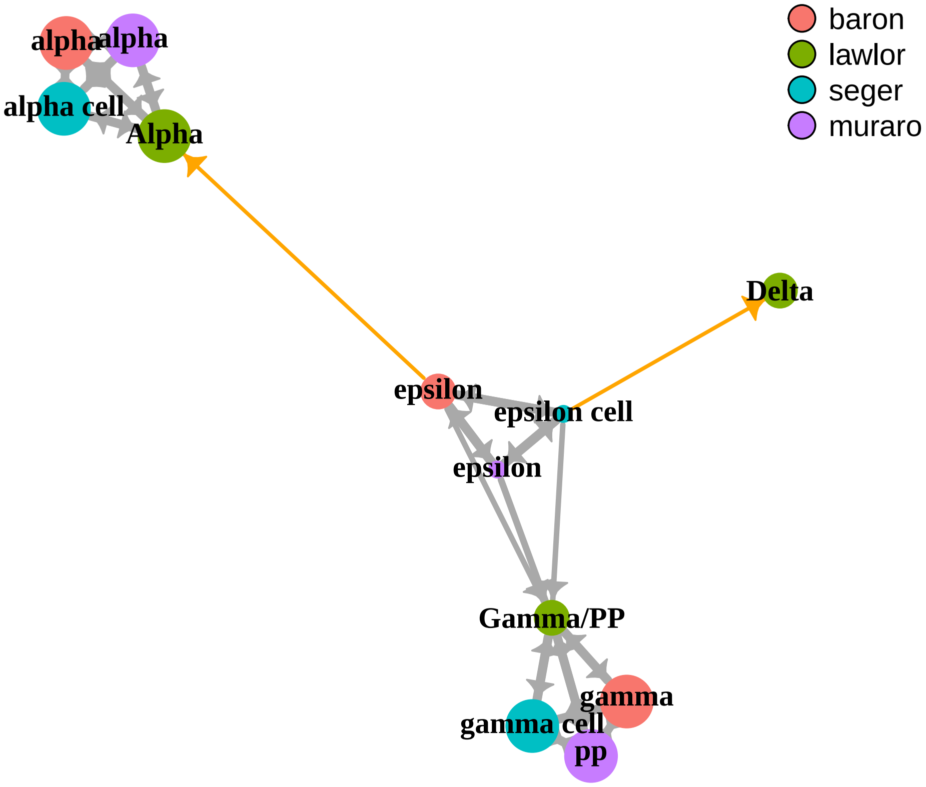
21. The last visualization is an alternative representation of the AUROC heatmap as a graph, which is particularly useful for large datasets. In this graph, top votes (AUROC > 0.5) are shown in gray, while outgroup votes (AUROC < 0.5) are shown in orange. To highlight close calls, we recommend keeping only strong outgroup votes, here with AUROC >= 0.3.

```
cluster_graph = makeClusterGraph(best_hits, low_threshold = 0.3)
plotClusterGraph(cluster_graph, pancreas_data$study_id,
                 pancreas_data$"cell type", size_factor=3)
```



We note that there are several orange edges, indicating that some cell types had two close matches. To investigate the origin of these close calls, we can focus on a cluster of interest (coi). Here we take a closer look at "baron|epsilon", query its closest neighbors in the graph with *extendClusterSet*, then zoom in on its subgraph with *subsetClusterGraph*.

```
coi = "baron|epsilon"
coi = extendClusterSet(cluster_graph, initial_set=coi, max_neighbor_distance=2)
subgraph = subsetClusterGraph(cluster_graph, coi)
plotClusterGraph(subgraph, pancreas_data$study_id,
                 pancreas_data$"cell type", size_factor=5)
```



In the "baron|epsilon" case, we find that the epsilon cell type is missing in the Lawlor dataset, so there is no natural match for the Baron epsilon cell type. In such cases, votes are frequently non-reciprocal and equally split between two unrelated cell types, here "Lawlor|Gamma/PP" and "Lawlor|Alpha". In general, the cluster graph can be used to understand how meta-clusters are extracted, why some clusters are tagged as outliers and diagnose problems where the resolution of cell types differs across datasets.