performance of community detection methods

Fabio Morea

2023-05-19

Abstract

This notebook compares the performance of several community detection methods, and explores the potential improvements that can be obtained by the following techniques: pruning, consensus and pre-cut.

introduction

Network analysis is a powerful tool for understanding complex systems in both the hard and social sciences. By representing data as a network, it is possible to identify communities of similar members, or nodes, that are connected to each other by various relationships. This allows to gain insights into the structure of the system, as well as the behavior of the nodes within it.

For example, in social networks, nodes can represent people, and the relationships between them can represent various types of interactions, such as friendships or collaborations. By analyzing the network, researchers can identify clusters of people who are more likely to interact with each other, or who share similar characteristics.

As the size of a network increases, the complexity of the network also increases, making it difficult to represent the network in a simple visual format. As a result, a simple visualization may not provide much clarity in understanding the structure of the network. To gain a better understanding of a large network, more sophisticated visualizations and analysis techniques are needed.

This notebook compares the performance of several community detection methods, and explores the potential improvements that can be obtained by the following techniques: pruning, consensus and pre-cut."

The analysis is developed as follows: 1- Indicators and benchmark networks to assess performance of CD methods 2- community definitions and community detection methods 3- performance of 6 CD methods on LFR benchmark networks (repeated trials) 4- potential improvements by p

- about different methods (6)
- about LFR benchmark networks

About LFR benchmark networks

LFR (Lancichinetti-Fortunato-Radicchi) benchmark graphs are synthetic networks used to evaluate the performance of network analysis algorithms. They are generated using a stochastic block model and are designed to have realistic community structure, degree distributions, and other properties of real-world networks.

They are used to test the accuracy of algorithms for network analysis tasks such as community detection, link prediction, and node classification.

Benchmark graphs for testing community detection algorithms, Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi, Phys. Rev. E 78, 046110 2008

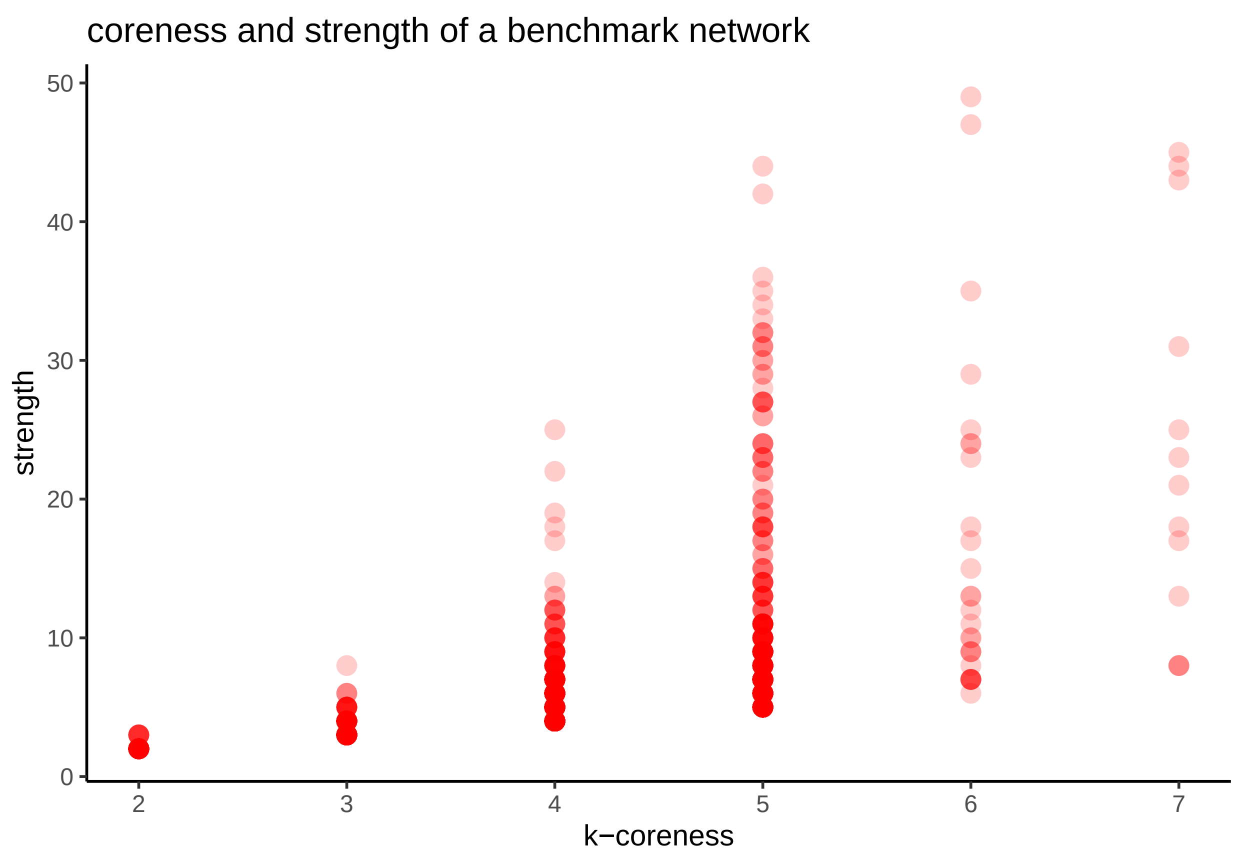
LFR_benchmark_graph — NetworkX 3.1 documentation

sample network (mu = 0.50): coreness and strength

```r
load_benchmark_network <- function(mui, path = path, verbose = FALSE) {
    #upload graph
    filename = paste0(path, mui, ".gml")
    g <- read_graph(filename, format = "gml")
    # extract giant component
    components <- igraph::clusters(g, mode = "weak")
    g <-induced_subgraph(g, V(g)[components$membership == which.max(components$csize)])
    # set names and weight (ww <- 1.0)
    V(g)$core <- coreness(g)
    V(g)$str <- strength(g)
    V(g)$name <- paste0("V" , V(g)$label)
    E(g)$ww <- 1.0
    # print
    if (verbose == TRUE) {
        print(paste0("Loaded benchmark network ", path, mui, ".gml"))
        print(paste("Giant component size :", length(V(g)$community)))
        print(paste("Built-in communities: ", max(V(g)$community)))
        mod <- round( modularity(g, array(V(g)$community) ), digits = 3)
        print(paste("Modularity of built-in communities: ", mod))
    }
    return(g)
}
```

```r
path <- "./LFR_graphs/LFR/LFR_benchmark_"
g <- load_benchmark_network(mui = 20, path = path, verbose = TRUE)
```

```
## [1] "Loaded benchmark network ./LFR_graphs/LFR/LFR_benchmark_20.gml"
## [1] "Giant component size : 999"
## [1] "Built-in communities:  37"
## [1] "Modularity of built-in communities:  0.723"
```



exploration of variability of results using different methods

```r
find_communities <- function(g, method, verbose = FALSE) {
    if (method == "LV"){
        comms <- cluster_louvain(g, resolution = 1.0)
    } else if (method == "FG"){
        comms <- fastgreedy.community(g)
    } else if (method == "IM"){
        comms <- infomap.community(g)
    } else if (method == "LP"){
        comms <- label.propagation.community(g)
    } else if (method == "ML"){
        comms <- multilevel.community(g)
    } else if (method == "WT"){
        comms <- walktrap.community(g)
    } else {
        print("No valid method")
        stop
    }
    comms$algorithm = method
    if (verbose == TRUE) {print(paste("Community detection with ", method, "completed."))}
    return(comms)
}
```

```r
analyse_communities <- function(communities, mui, verbose = FALSE){

    method <- communities$algorithm
    c_membership <- communities$membership
    mod <- round(modularity (g,  c_membership+1), digits = 3)
    # need to use c_membership + 1 to handle community label 0
    #
    nc <- length(table(c_membership))

    true_labels <- data.frame(V(g)$name, V(g)$community)
    louvain_labels <- data.frame(V(g)$name,c_membership)
    n_m_i = round( NMI(true_labels,louvain_labels)$value, 3)

    if(verbose == TRUE){
        print(paste("Communities found: ",nc))
        print(paste("Modularity: ", mod))
        print(paste("Normalized Mutual Information", n_m_i))
        #print(table(list(c_membership)))
    }
    return(data.frame(mui, method , mod, nc, n_m_i))
}
```

Community detection: single trial

Method LV Louvain

```r
mui = 20
g <- load_benchmark_network(mui = mui, path = path)
tmp_comms <- find_communities(g, method = "LV", verbose = TRUE)
```

```
## [1] "Community detection with  LV completed."
```

```r
results <- analyse_communities(tmp_comms, mui, verbose = TRUE)
```

```
## [1] "Communities found:  27"
## [1] "Modularity:  0.725"
## [1] "Normalized Mutual Information 0.923"
```

Method FG Fast Greedy

```r
g <- load_benchmark_network(mui = mui, path = path)
tmp_comms <- find_communities(g, method = "FG", verbose = TRUE)
```

```
## [1] "Community detection with  FG completed."
```

```r
results <- analyse_communities(tmp_comms, mui, verbose = TRUE)
```

```
## [1] "Communities found:  23"
## [1] "Modularity:  0.703"
## [1] "Normalized Mutual Information 0.858"
```

Method IM infomap

```r
g <- load_benchmark_network(mui = mui, path = path)
tmp_comms <- find_communities(g, method = "IM", verbose = TRUE)
```

```
## [1] "Community detection with  IM completed."
```

```r
results <- analyse_communities(tmp_comms, mui, verbose = TRUE)
```

```
## [1] "Communities found:  44"
## [1] "Modularity:  0.722"
## [1] "Normalized Mutual Information 0.977"
```

method LP Label Propagation

```
g <- load_benchmark_network(mui = mui, path = path)
tmp_comms <- find_communities(g, method = "IM", verbose = TRUE)
```

```
## [1] "Community detection with  IM completed."
```

```
results <- analyse_communities(tmp_comms, mui, verbose = TRUE)
```

```
## [1] "Communities found:  44"
## [1] "Modularity:  0.722"
## [1] "Normalized Mutual Information 0.977"
```

method ML Multi Label

```
g <- load_benchmark_network(mui = mui, path = path)
tmp_comms <- find_communities(g, method = "ML", verbose = TRUE)
```

```
## [1] "Community detection with  ML completed."
```

```
results <- analyse_communities(tmp_comms, mui, verbose = TRUE)
```

```
## [1] "Communities found:  27"
## [1] "Modularity:  0.724"
## [1] "Normalized Mutual Information 0.922"
```

method WT walktrap

```
g <- load_benchmark_network(mui = mui, path = path)
tmp_comms <- find_communities(g, method = "WT", verbose = TRUE)
```

```
## [1] "Community detection with  WT completed."
```

```
results <- analyse_communities(tmp_comms, mui, verbose = TRUE)
```

```
## [1] "Communities found:  39"
## [1] "Modularity:  0.72"
## [1] "Normalized Mutual Information 0.955"
```

Community detection: repeated trials

repeted trials generate *may* different results (depending on the algorithm)

```
mui = 60
g <- load_benchmark_network(mui = mui, path = path)
for (i in 1:10){
    tmp_comms <- find_communities(g, method = "ML")
    results <- analyse_communities(tmp_comms, mui)
    print(paste("trial", i,  "communities found", results[4]))
}
```

```
## [1] "trial 1 communities found 16"
## [1] "trial 2 communities found 17"
## [1] "trial 3 communities found 18"
## [1] "trial 4 communities found 16"
## [1] "trial 5 communities found 15"
## [1] "trial 6 communities found 15"
## [1] "trial 7 communities found 15"
## [1] "trial 8 communities found 15"
## [1] "trial 9 communities found 13"
## [1] "trial 10 communities found 16"
```

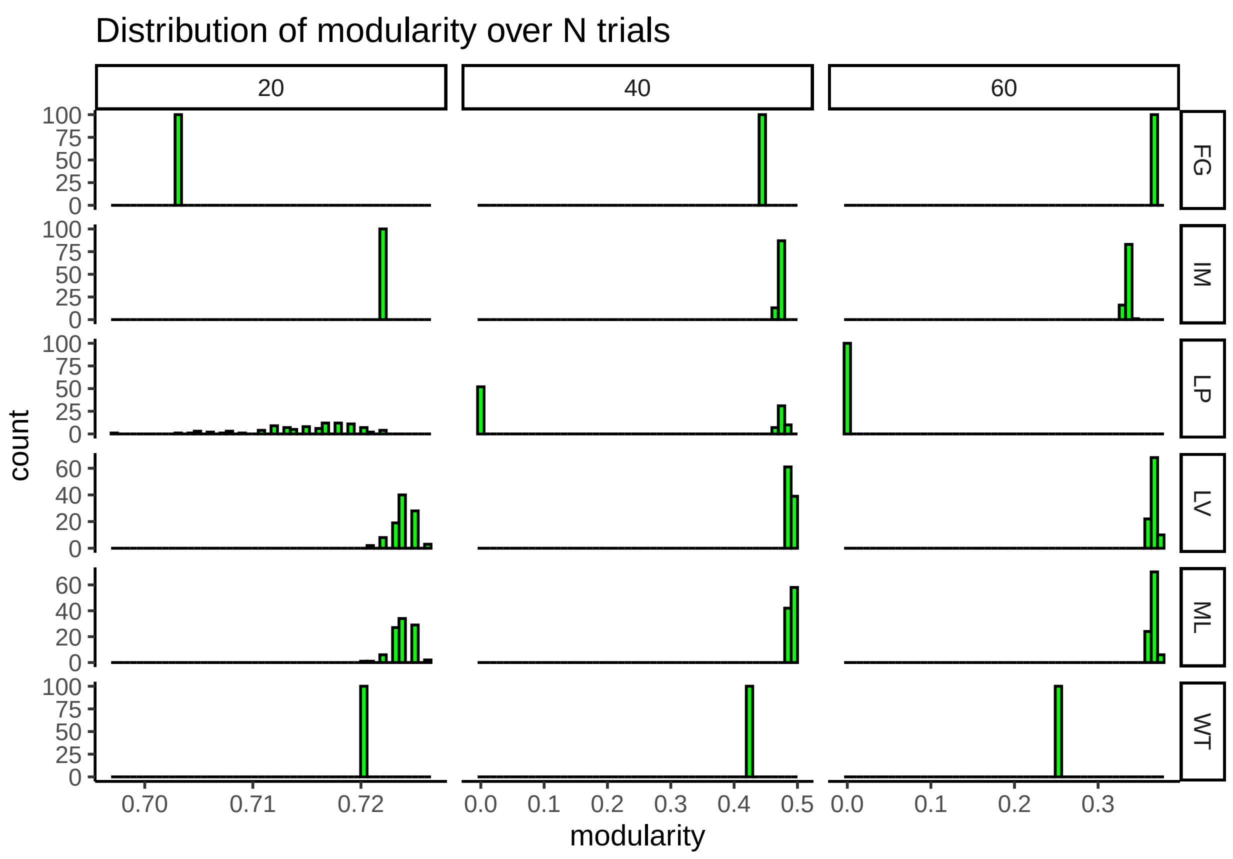
Community detection: N trials

We generate 200 trials for a range of MU and save results.

```
data_to_plot<- read_csv('results_n_trials.csv')
```

results

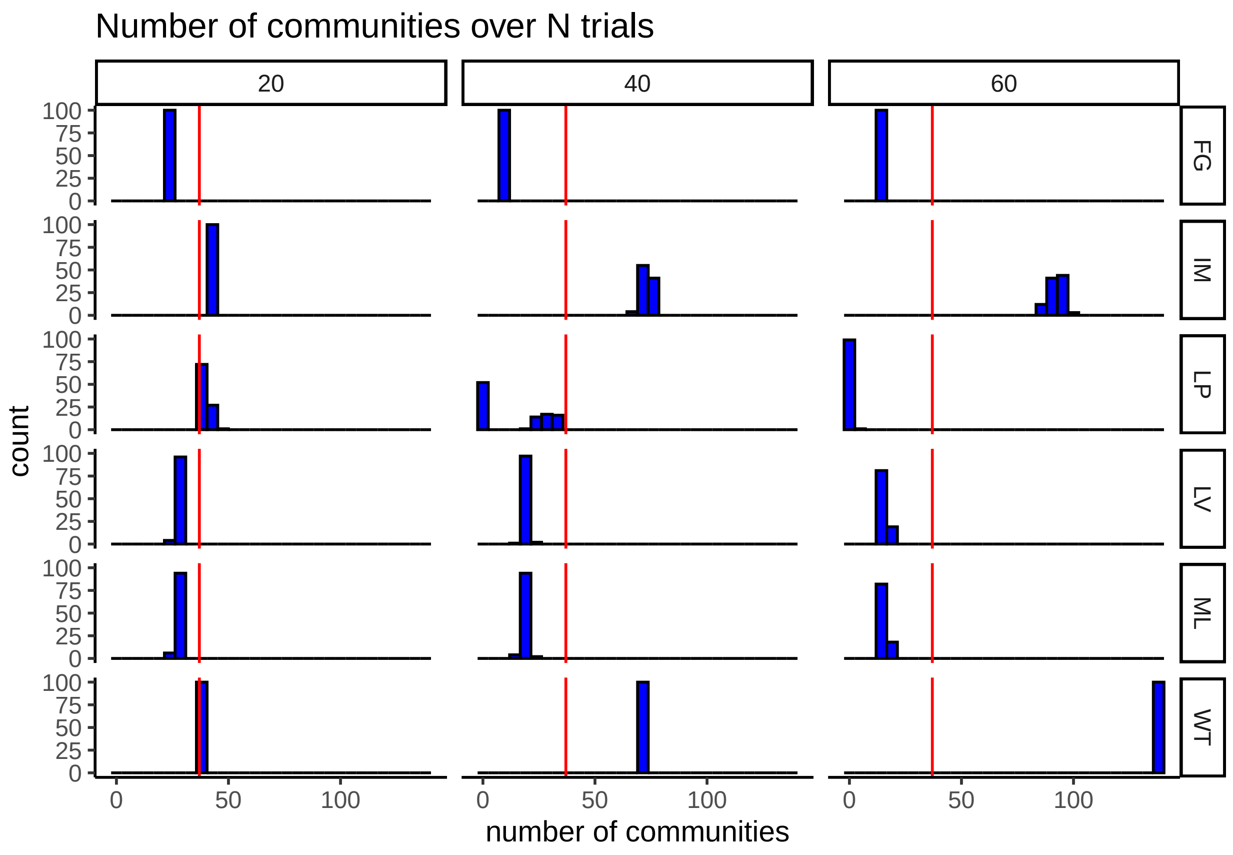
```
data_to_plot %>% filter(mui %in% c(20,40,60)) %>%
    ggplot(aes(x = mod)) +
    geom_histogram(color = "black", fill = "green", bins = 50) +
    labs(title = "Distribution of modularity over N trials", x = "modularity", y = "count") +
    theme_classic() +
    facet_grid(rows = vars(method), cols = vars(mui),  scales = "free" )
```



results (2)

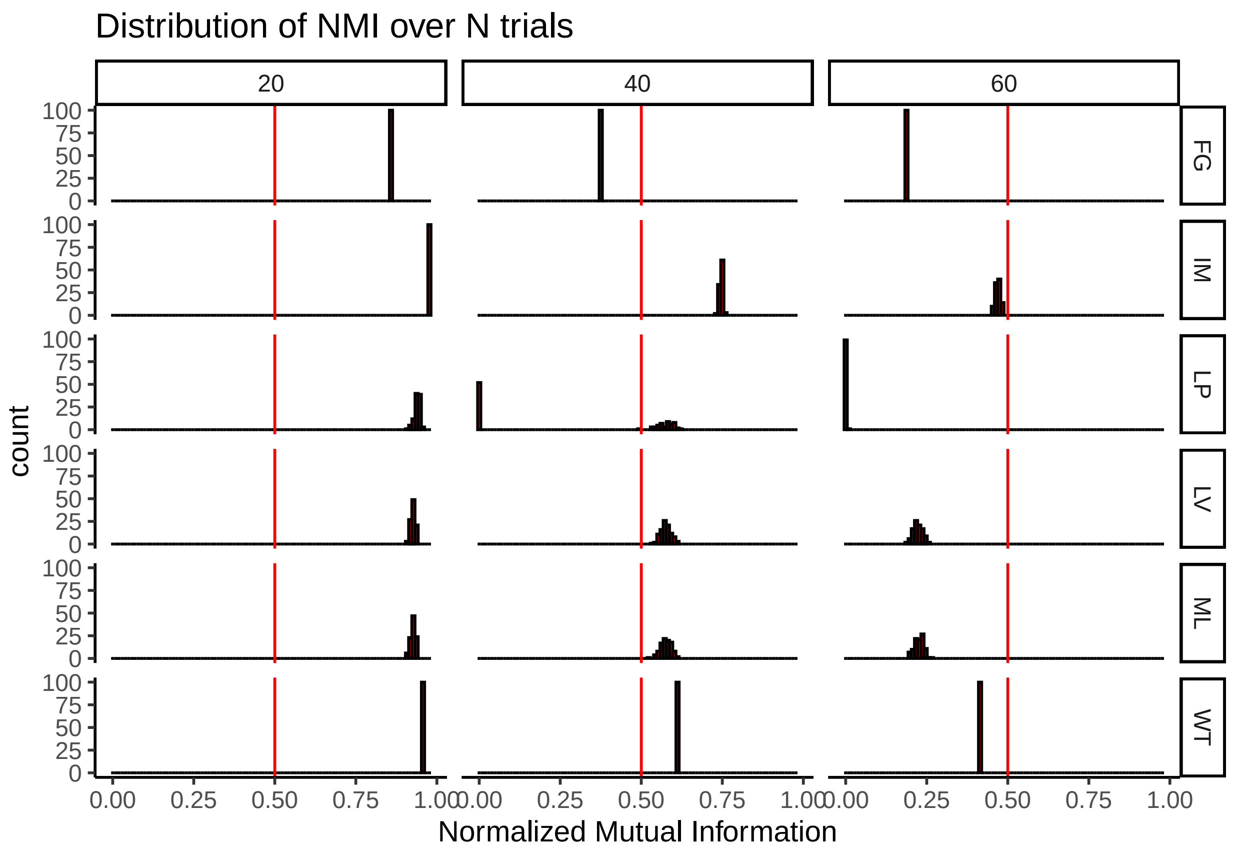
```
data_to_plot %>% filter(mui %in% c(20,40,60)) %>%
    ggplot(aes(x = nc)) +
    geom_histogram(color = "black", fill = "blue" ) +
    labs(title = "Number of communities over N trials", x = "number of communities", y = "count") +
    geom_vline(xintercept = 37, color = 'red')+
    theme_classic() +
    facet_grid(rows = vars(method), cols = vars(mui))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



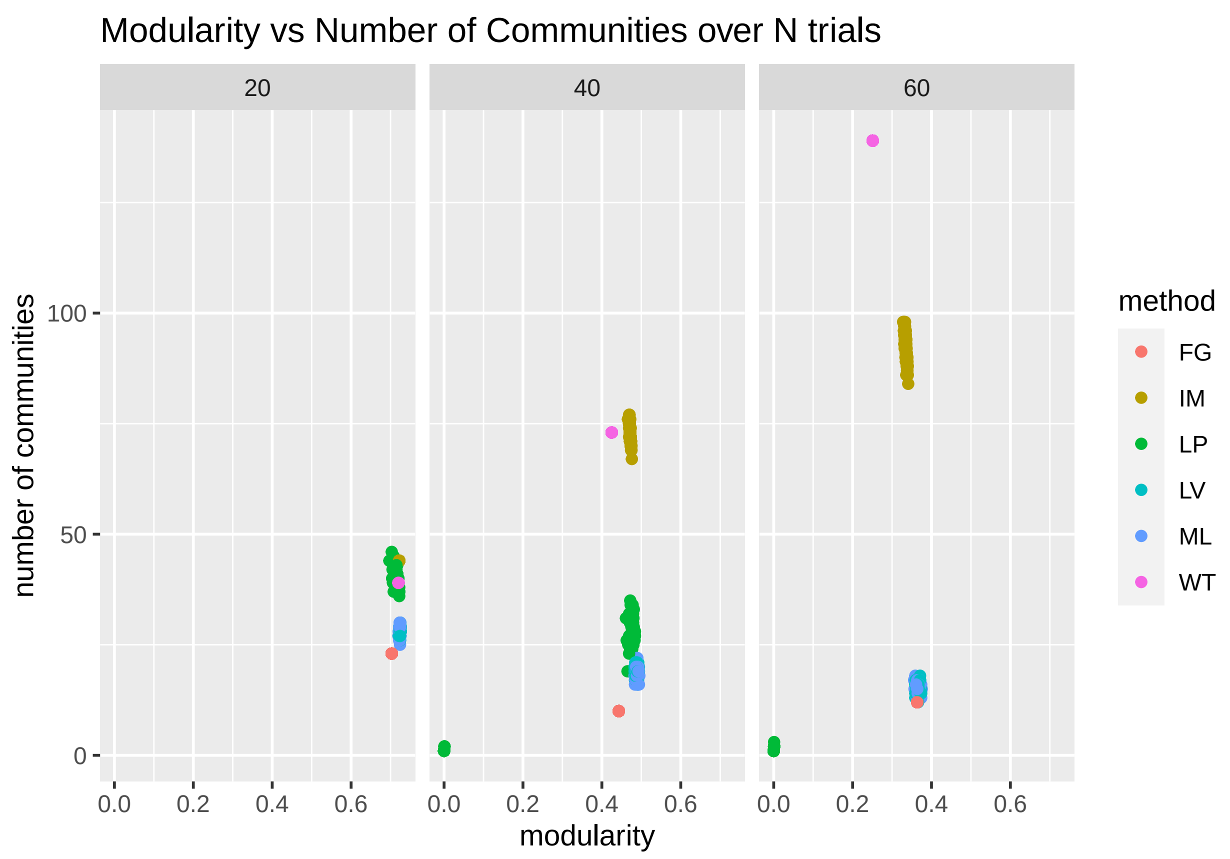
results (3)

```
data_to_plot %>% filter(mui %in% c(20,40,60)) %>%
    ggplot(aes(x = n_m_i)) +
    geom_histogram(color = "black", fill = "red", bins = 100) +
    labs(title = "Distribution of NMI over N trials", x = "Normalized Mutual Information", y = "count")
    geom_vline(xintercept = 0.5, color = 'red')+
    theme_classic() +
    facet_grid( method ~ mui )
```



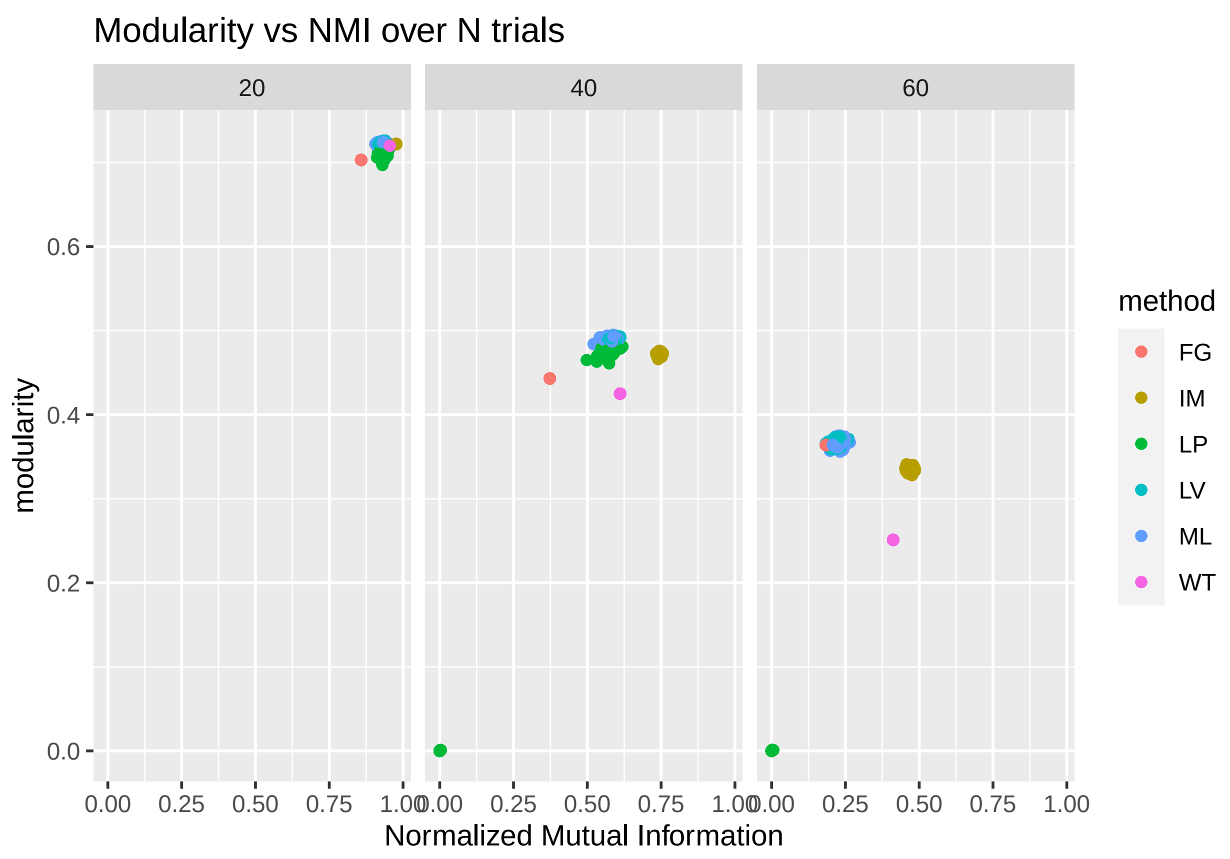
results (4)

```
data_to_plot %>% filter(mui %in% c(20,40,60)) %>%
    ggplot(aes(x = mod, y = nc)) +
    geom_point(aes(color = method)) +
    labs(title = "Modularity vs Number of Communities over N trials", y = "number of communities", x =
    theme_gray() +
    facet_wrap(mui ~ .)
```

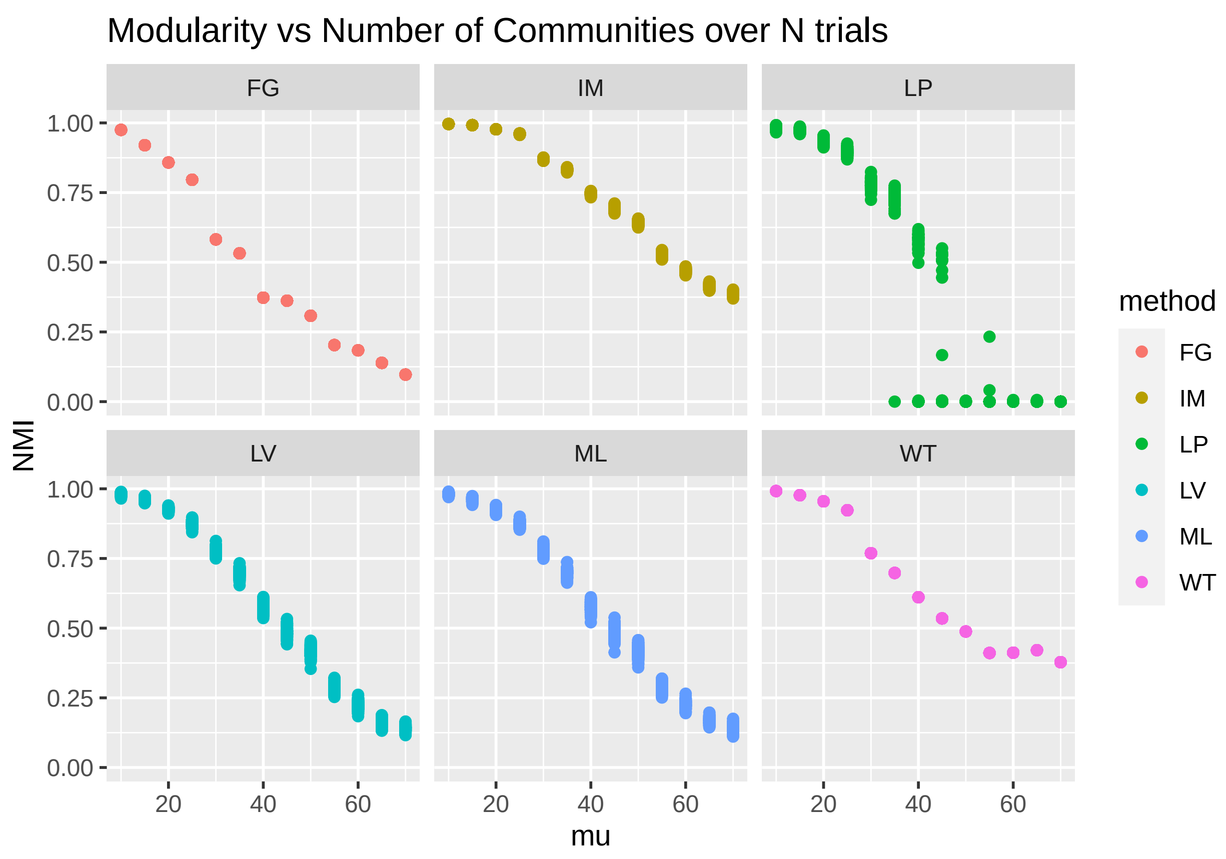


results (5)

```
data_to_plot %>% filter(mui %in% c(20,40,60)) %>%
    ggplot(aes(x = n_m_i, y = mod)) +
    geom_point(aes(color = method)) +
    labs(title = "Modularity vs NMI over N trials", x = "Normalized Mutual Information", y = "modularit
    theme_gray() +
    facet_wrap(mui ~ .)#, scales ="free")
```

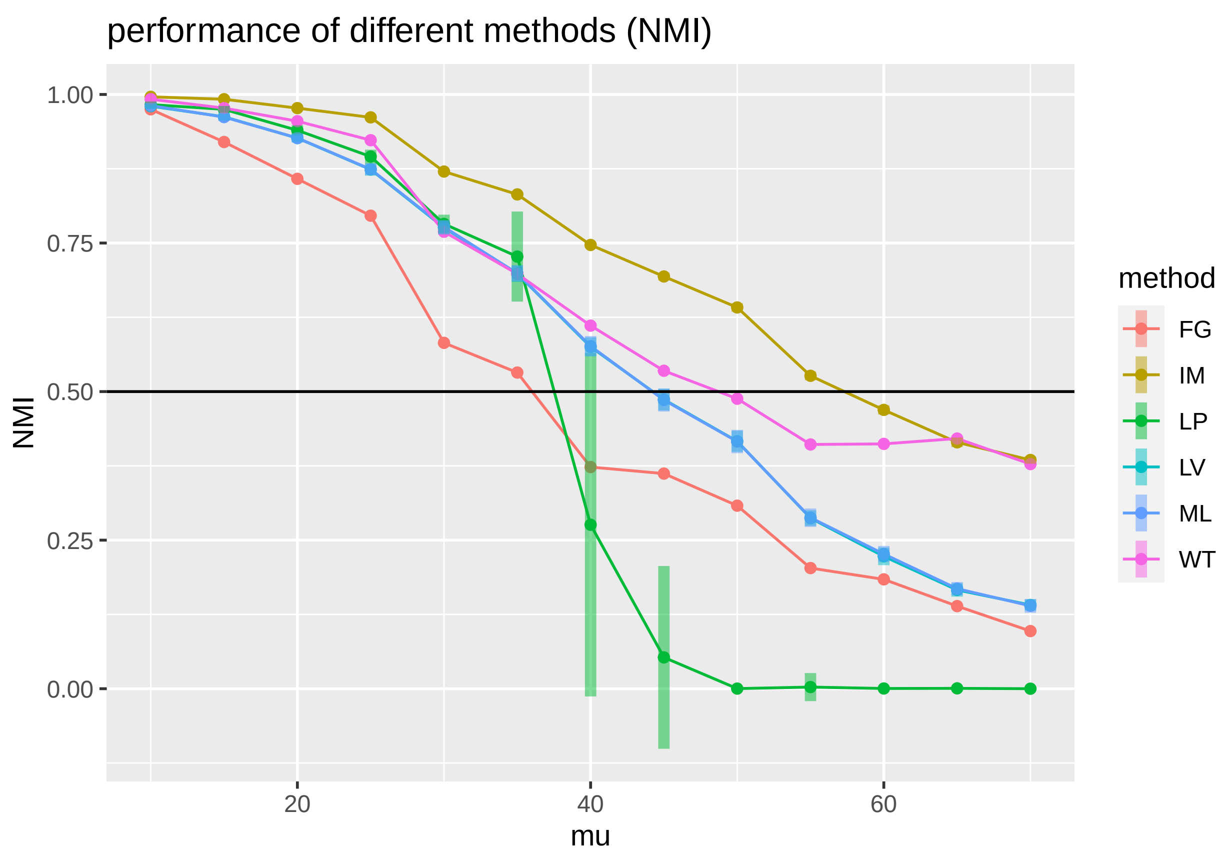


```
data_to_plot %>% ggplot(aes(x = mui, y = n_m_i)) +
    geom_point(aes(color = method)) +
    labs(title = "Modularity vs Number of Communities over N trials", y = "NMI", x = "mu") +
    theme_gray() +
    facet_wrap(method ~ .) #,  scales = "free_x"
```

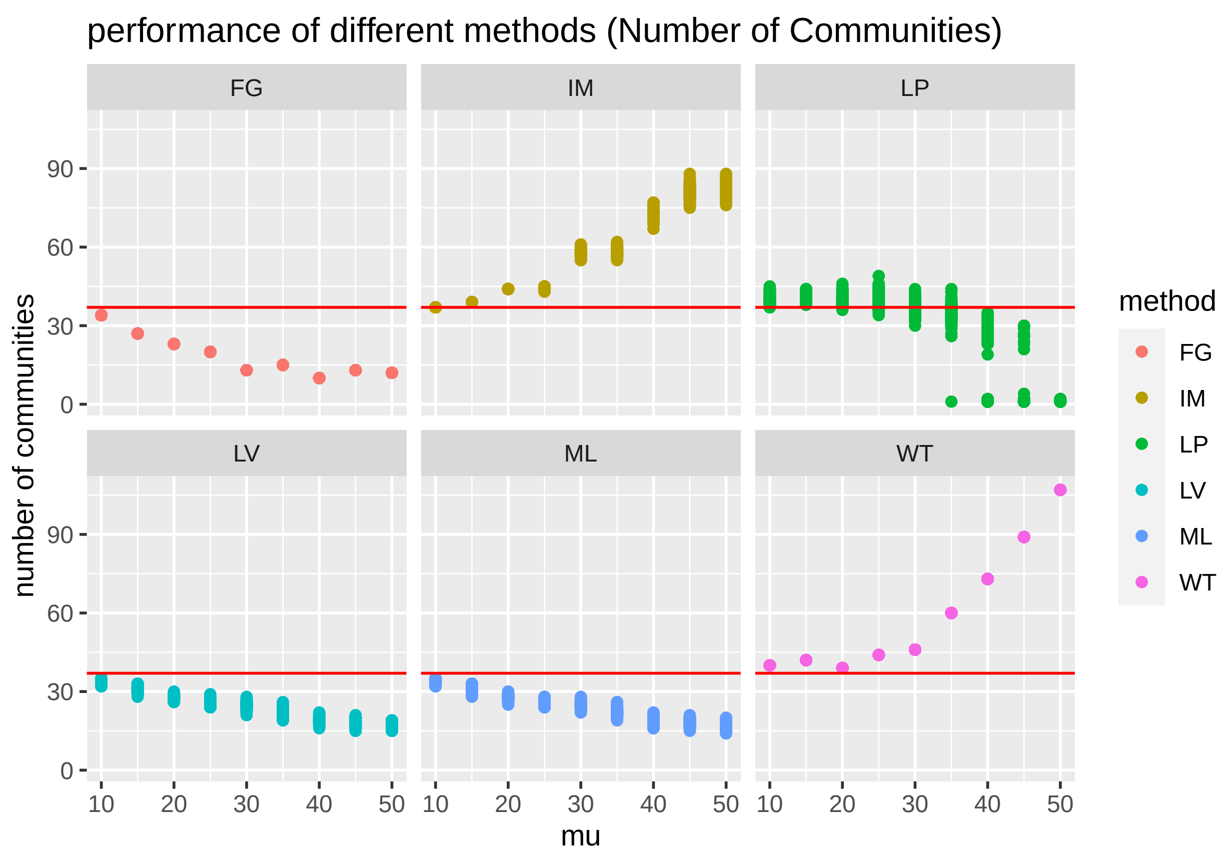


```
data_to_plot %>% group_by(method, mui) %>%
    summarise(mean_nmi = mean(n_m_i), sd_nmi = sd(n_m_i) ) %>%
    ggplot(aes(x = mui, y = mean_nmi)) +
    geom_line(aes(color = method)) +
    geom_point(aes(color = method)) +
    geom_linerange(aes(ymin =mean_nmi-sd_nmi, ymax = mean_nmi + sd_nmi, color = method), linewidth = 2,
    geom_hline(yintercept = 0.5)+
    labs(title = "performance of different methods (NMI)", y = "NMI", x = "mu") +
    theme_gray()
```

```
## 'summarise()' has grouped output by 'method'. You can override using the
## '.groups' argument.
```



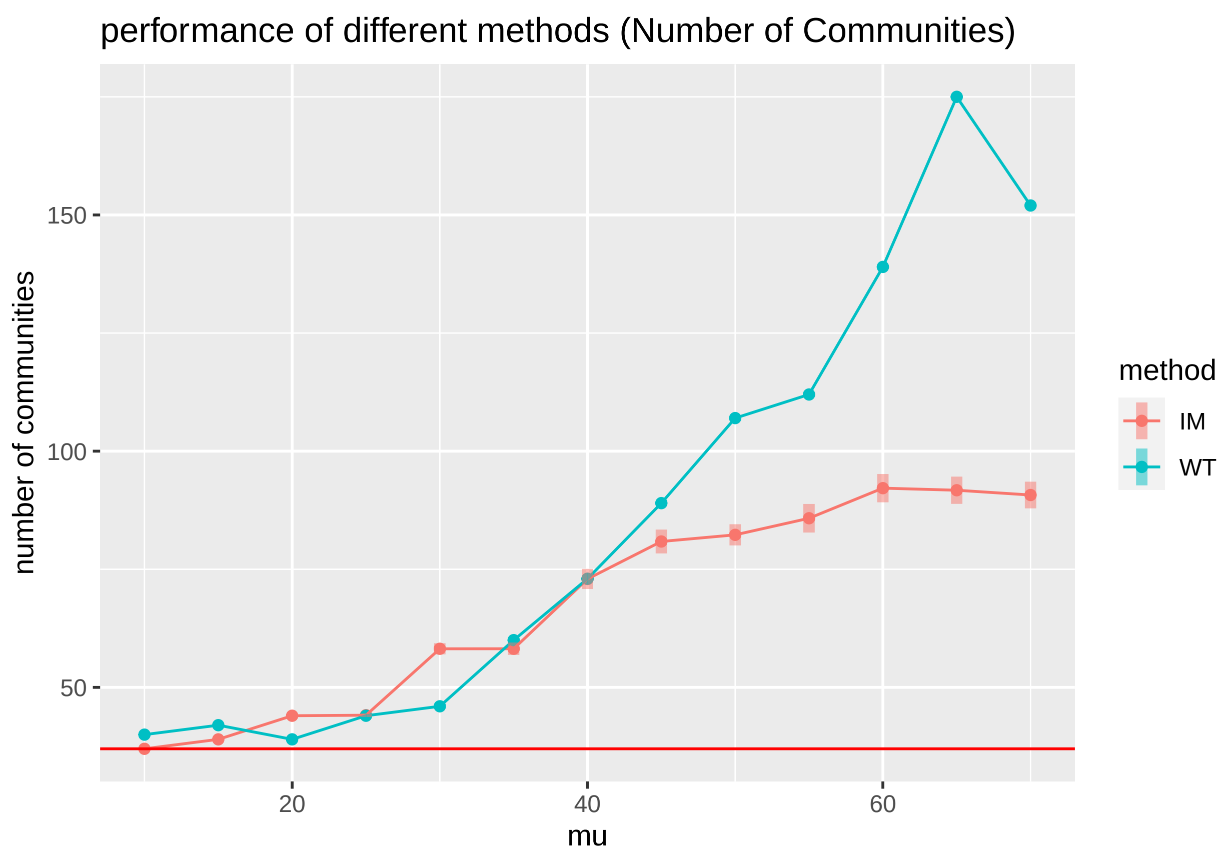
```
data_to_plot %>% filter(mui <=50) %>%
    ggplot(aes(x = mui, y = nc)) +
    geom_point(aes(color = method)) +
    labs(title = "performance of different methods (Number of Communities)", y = "number of communities
    geom_hline(yintercept = 37, color = 'red')+
    theme_gray() +
    facet_wrap(method ~ .)
```



methods that overestimate NC

```
data_to_plot %>% filter(method %in% c("WT", "IM")) %>%
    group_by(method, mui) %>%
    summarise(mean_nc = mean(nc), sd_nc = sd(nc)) %>%
    ggplot(aes(x = mui, y = mean_nc)) +
    geom_point(aes(color = method)) +
    geom_line(aes(color = method)) +
    geom_linerange(aes(ymin =mean_nc-sd_nc, ymax = mean_nc + sd_nc, color = method), linewidth = 2, alph
    geom_hline(yintercept = 37, color = 'red')+

    labs(title = "performance of different methods (Number of Communities)", y = "number of communities
    theme_gray()
```

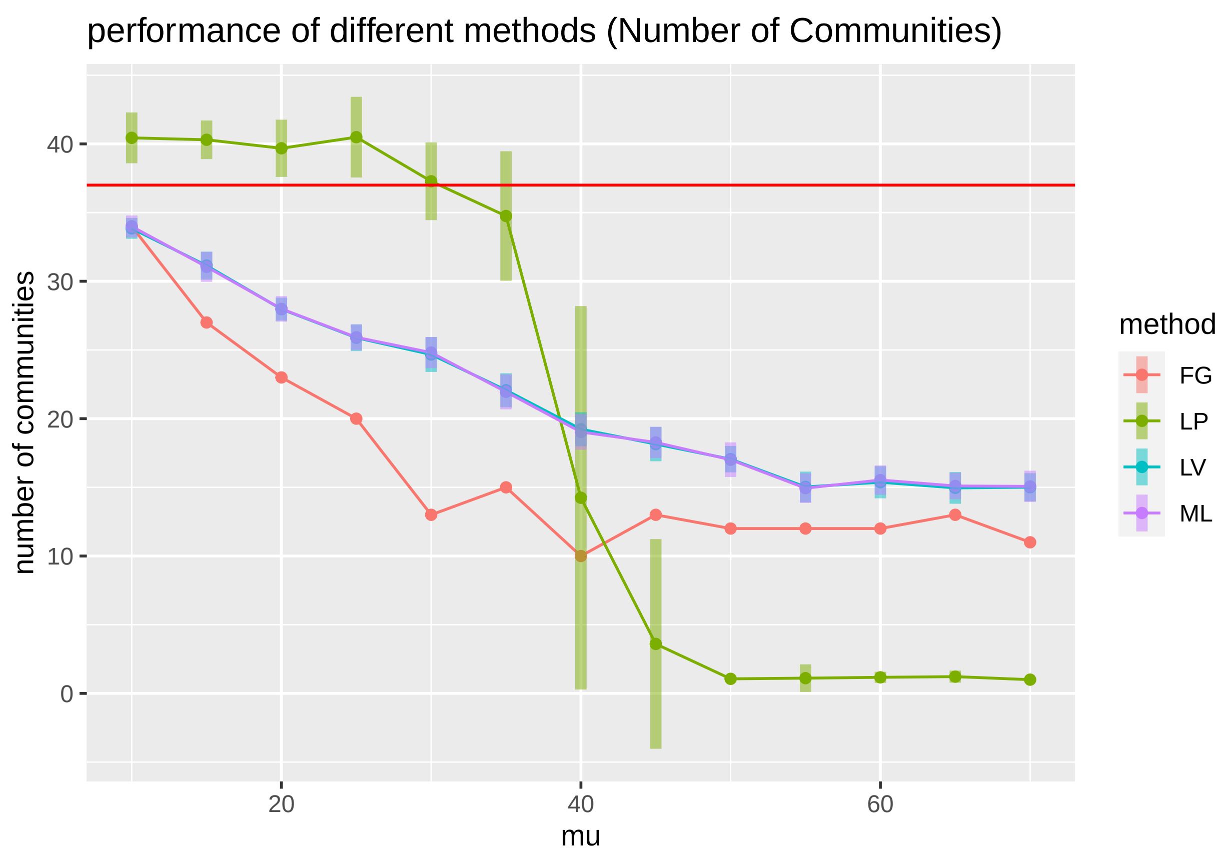
```
## 'summarise()' has grouped output by 'method'. You can override using the
## '.groups' argument.
```



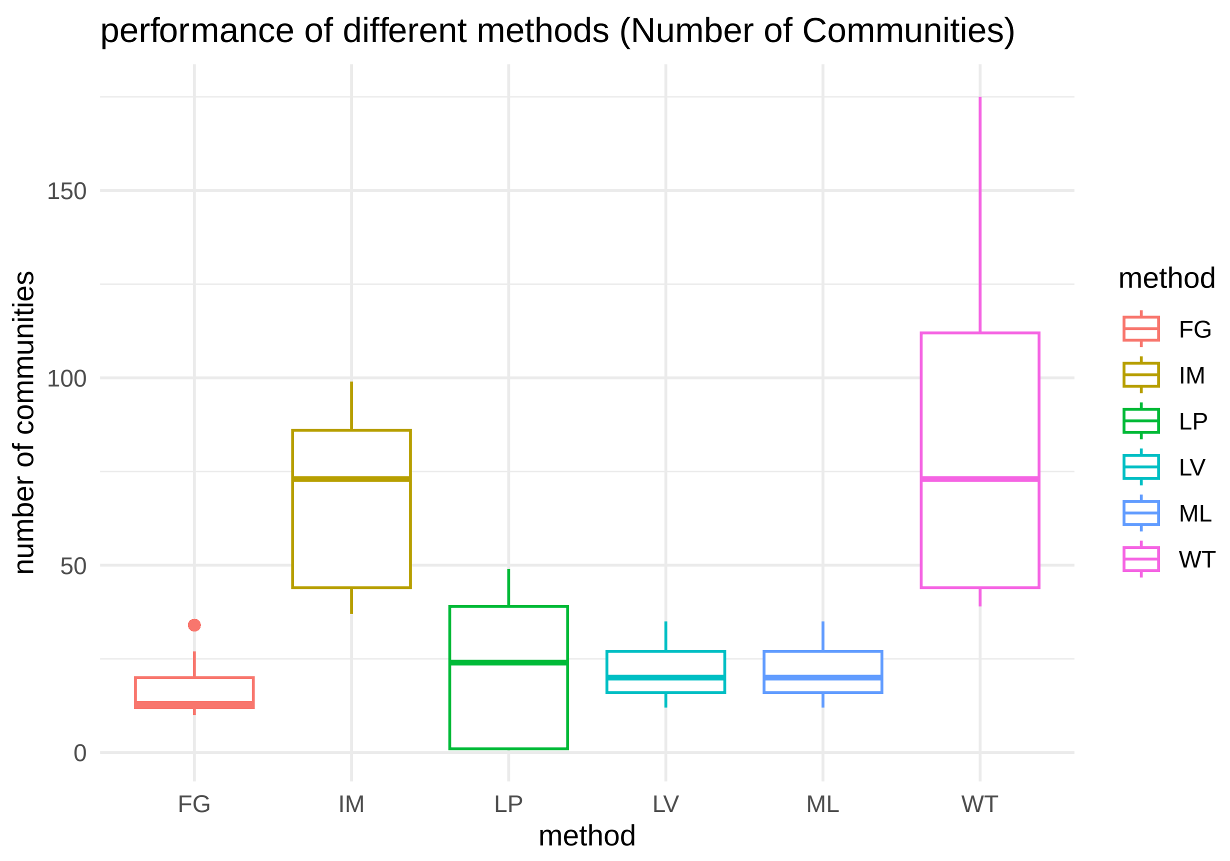
methods that underestimate NC

```
data_to_plot %>% filter(method %in% c("LV", "FG", "ML", "LP")) %>%
    group_by(method, mui) %>%
    summarise(mean_nc = mean(nc), sd_nc = sd(nc)) %>%
    ggplot(aes(x = mui, y = mean_nc)) +
    geom_point(aes(color = method)) +
    geom_line(aes(color = method)) +
    geom_linerange(aes(ymin =mean_nc-sd_nc, ymax = mean_nc + sd_nc, color = method), linewidth = 2, alpl
    geom_hline(yintercept = 37, color = 'red')+

    labs(title = "performance of different methods (Number of Communities)", y = "number of communities
    theme_gray()
```

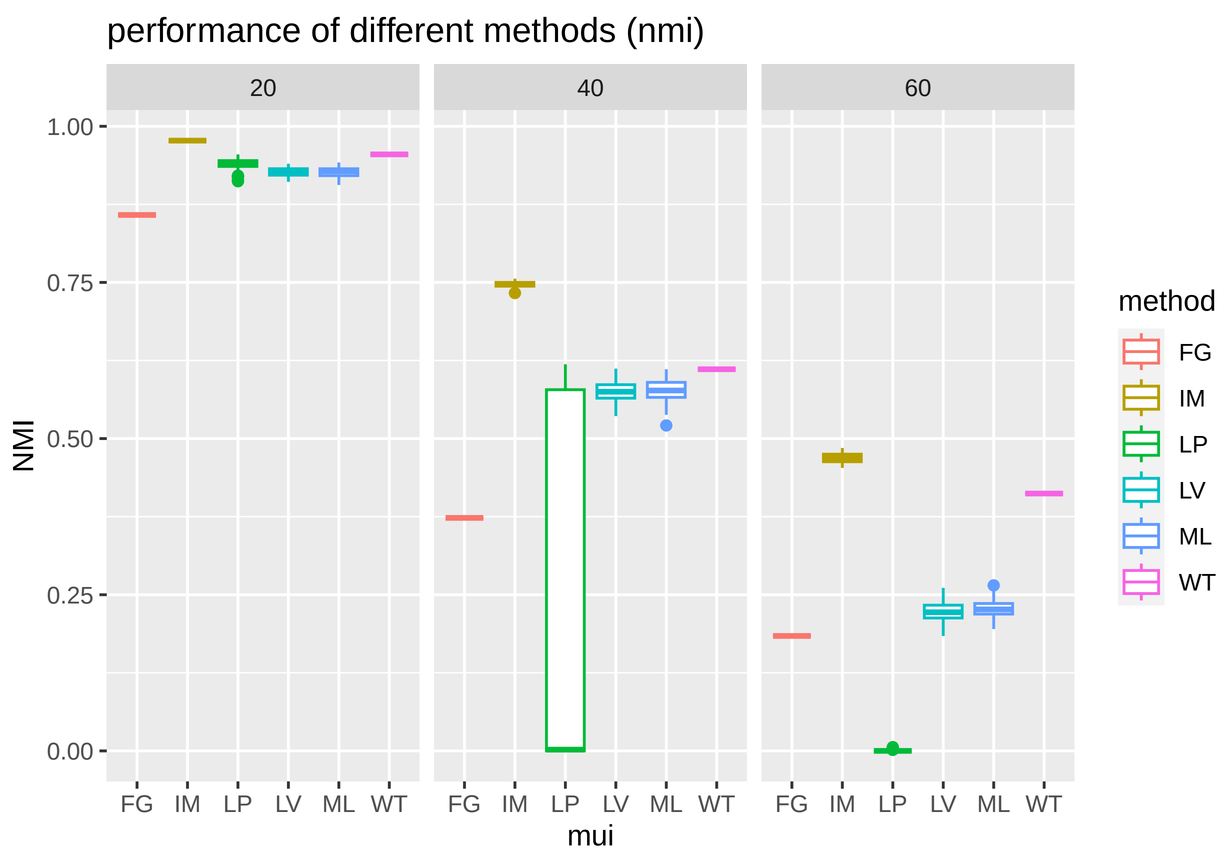
```
## ‘summarise()‘ has grouped output by ’method’. You can override using the
## ‘.groups‘ argument.
```



```
data_to_plot %>% ggplot(aes( x = method, y = nc)) +
    geom_boxplot(aes(color = method)) +
     labs(title = "performance of different methods (Number of Communities)", y = "number of communitie
    theme_minimal()
```

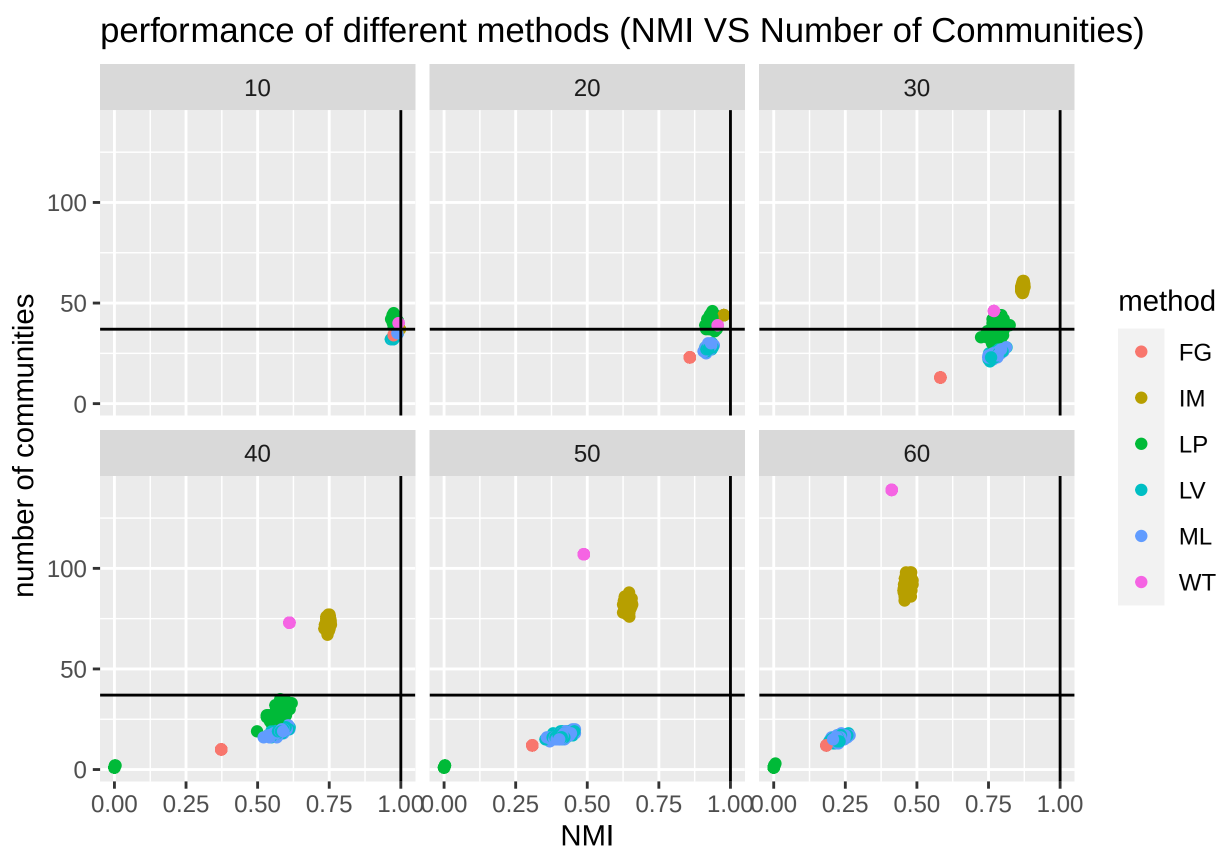


```
data_to_plot %>% filter(mui %in%c(20,40,60)) %>%
    ggplot(aes( x = method, y = n_m_i)) +
    geom_boxplot(aes(color = method)) +
     labs(title = "performance of different methods (nmi)", y = "NMI", x = "mui") +
    theme_gray()  + facet_wrap((mui ~ .))
```



Performance measured as NMI and NC (improvement: normalize NC/37)

```
data_to_plot %>% filter(mui %in%c(10,20,30,40,50,60)) %>%
    ggplot(aes(x = n_m_i, y = nc)) +
    geom_point(aes(color = method))+
    labs(title = "performance of different methods (NMI VS Number of Communities)",
         y = "number of communities", x = "NMI") +
    geom_hline(yintercept = 37)+
    geom_vline(xintercept = 1.0)+
    theme_gray()  + facet_wrap(mui ~ .)
```



improving performance by pruning to "community zero"

pruning small communities to a community 0 can improve NMI and NC of course only for the methods that overestimate NC

```
prune_to_community0 <- function(comms, min_vids, verbose = FALSE) {
    comms_membership <- comms$membership
    cs <- table(comms_membership)
    for (i in 1:max(comms_membership)) {
        if (cs[i] < min_vids) {
            comms_membership[comms_membership == i] <- 0
        }
    }
    comms$membership <- comms_membership
    if (verbose == TRUE) {
        print(paste("Pruning communities below ", min_vids))}
    return(comms)
}
```

```
mui = 40
g <- load_benchmark_network(mui = mui, path = path, verbose = TRUE)
```

```
## [1] "Loaded benchmark network ./LFR_graphs/LFR/LFR_benchmark_40.gml"
## [1] "Giant component size : 1000"
## [1] "Built-in communities:  37"
## [1] "Modularity of built-in communities:  0.464"
```

```
comms_sample <- find_communities(g, method = "WT", verbose = TRUE)
```

```
## [1] "Community detection with  WT completed."
```

```
results <- analyse_communities(comms_sample, mui, verbose = TRUE)
```

```
## [1] "Communities found:  73"
## [1] "Modularity:  0.425"
## [1] "Normalized Mutual Information 0.611"
```

```
comms_pruned <- prune_to_community0(comms_sample, min_vids = 10, verbose = TRUE)
```

```
## [1] "Pruning communities below  10"
```

```
results_pruned <- analyse_communities(comms_pruned, mui, verbose = TRUE)
```

```
## [1] "Communities found:  28"
## [1] "Modularity:  0.422"
## [1] "Normalized Mutual Information 0.55"
```

```
mui = 30
g <- load_benchmark_network(mui = mui, path = path, verbose = TRUE)
```

```
## [1] "Loaded benchmark network ./LFR_graphs/LFR/LFR_benchmark_30.gml"
## [1] "Giant component size : 1000"
## [1] "Built-in communities:  37"
## [1] "Modularity of built-in communities:  0.578"
```

```
comms_sample <- find_communities(g, method = "IM", verbose = TRUE)
```

```
## [1] "Community detection with  IM completed."
```

```
results <- analyse_communities(comms_sample, mui, verbose = TRUE)
```

```
## [1] "Communities found:  56"
## [1] "Modularity:  0.585"
## [1] "Normalized Mutual Information 0.864"
```

```
for (mv in 1:10){
    comms_pruned <- prune_to_community0(comms_sample, min_vids = mv, verbose = FALSE)
    results_pruned <- analyse_communities(comms_pruned, mui, verbose = FALSE)
    print(paste(results_pruned$n_m_i,results_pruned$nc ))
}
```

```
## [1] "0.864 56"
## [1] "0.864 56"
## [1] "0.864 56"
## [1] "0.859 50"
## [1] "0.855 47"
## [1] "0.853 46"
## [1] "0.851 45"
## [1] "0.845 41"
## [1] "0.842 39"
## [1] "0.842 38"
```

NMI è invariata NC migliora

```
g <- load_benchmark_network(mui = mui, path = path, verbose = TRUE)
```

```
## [1] "Loaded benchmark network ./LFR_graphs/LFR/LFR_benchmark_30.gml"
## [1] "Giant component size : 1000"
## [1] "Built-in communities:  37"
## [1] "Modularity of built-in communities:  0.578"
```

```
comms_sample <- find_communities(g, method = "WT", verbose = TRUE)
```

```
## [1] "Community detection with  WT completed."
```

```
results <- analyse_communities(comms_sample, mui, verbose = TRUE)
```

```
## [1] "Communities found:  46"
## [1] "Modularity:  0.569"
## [1] "Normalized Mutual Information 0.769"
```

```
for (mv in 1:10){
    comms_pruned <- prune_to_community0(comms_sample, min_vids = mv, verbose = FALSE)
    results_pruned <- analyse_communities(comms_pruned, mui , verbose = FALSE)
    print(paste(results_pruned$n_m_i,results_pruned$nc ))
}
```

```
## [1] "0.769 46"
## [1] "0.769 45"
## [1] "0.766 39"
## [1] "0.763 36"
## [1] "0.763 35"
## [1] "0.763 35"
## [1] "0.76 33"
## [1] "0.76 33"
## [1] "0.76 33"
## [1] "0.757 32"
```

improving performance by consensus (on N repetitions of the same algorithm)

Consensus helps improving performance (applicable to methods that underestimate NV, as it generates a number of small communities, AND have some intrinsic variability of results)

```
normalized_co_occurrence <- function(all_clusters, n_trials)
{
    x <- matrix(0,
                nrow = nrow(all_clusters),
                ncol = nrow(all_clusters))
    colnames(x) <- V(g)$name
    rownames(x) <- V(g)$name

    for (i in (1:n_trials)) {
        nclusters <- max(all_clusters[, i])
        for (k in 1:nclusters) {
            samecluster <- (which(all_clusters[, i] == k))
            nc <- length(samecluster)
            for (t in 1:nc) {
                for (j in 1:nc) {
                    x[samecluster[j], samecluster[t]] <-
                        x[samecluster[j], samecluster[t]] + 1
                }
            }
        }
    }

    return (x/ ncol(all_clusters))
}
```

```
consensus <-
    function(all_clusters,
             min_p = 0.01,
             min_vids = 1,
             verbose = FALSE) {

        remaining <-
            normalized_co_occurrence(all_clusters, n_trials = n_trials)

        v.processed <- 0
        current.cluster = 0
        ccs <- data.frame(name = V(g)$name)
        ccs$mbshp = rep(0, nrow(ccs))
        ccs$prob = apply(remaining, 1, max)

        more_clusters_to_process = TRUE

        while (more_clusters_to_process) {
            cluster_ii_members <- which(remaining[1,] > min_p)
            v.processed <- v.processed + length(cluster_ii_members)
            if (verbose == TRUE) {
                print(paste(
                    "start While loop with v to process = ",
                    dim(remaining)[1],
                    v.processed
                ))
            }
            selected <-
                remaining[cluster_ii_members, cluster_ii_members]

            if (is.matrix(selected)) {
                diag(selected) <- 0 # diagonal elements are not relevant
                enough_vids     <- length(cluster_ii_members) > min_vids
                if (enough_vids == TRUE) {
                    current.cluster <- current.cluster + 1
                    if (verbose == TRUE) {
                        print(paste(
                            "community",
                            current.cluster,
                            max(selected),
                            length(cluster_ii_members)
                        ))
                    }

                    for (j in 1:nrow(selected)) {
                        nn <- names(selected[1,])[j]
                        if (verbose == TRUE)  {
                            print(paste(
                                "Adding",
                                nn,
                                "to comm",
                                current.cluster
                            ))
                        }
                        ccs$mbshp[ccs$name == nn] <-  current.cluster
                        ccs$prob[ccs$name == nn] <-  max(selected[j,])

                    }
                } else {
                    if (verbose == TRUE)  {
                        print(paste(
                            "community zero",
                            max(selected),
                            length(cluster_ii_members)
                        ))
                    }
                    for (j in 1:nrow(selected)) {
                        nn <- names(selected[1,])[j]
                        ccs$mbshp[ccs$name == nn] <-  0
                        ccs$prob[ccs$name == nn] <-  max(selected)
                    }
                }
            }
            tmp <- remaining[-cluster_ii_members, -cluster_ii_members]
            if (is.matrix(tmp)) {
                if (dim(tmp)[1] <= 1) {
                    more_clusters_to_process <- FALSE
                }
                remaining <- tmp
            } else {
                more_clusters_to_process <- FALSE
            }
        }

        return(ccs)
    }
```

```
n_trials <- 100
method <- 'LV'
all_clusters <- c()
mui = 20
g <- load_benchmark_network(mui = mui, path = path, verbose = TRUE)
```

```
## [1] "Loaded benchmark network ./LFR_graphs/LFR/LFR_benchmark_20.gml"
## [1] "Giant component size : 999"
## [1] "Built-in communities:  37"
## [1] "Modularity of built-in communities:  0.723"
```

```
for (i in 1:n_trials){
    comms_i <- find_communities(g, method = method)
    all_clusters <- cbind(all_clusters, comms_i$membership)
}
print("")
```

```
## [1] ""
```

```
print("Results of last single trial")
```

```
## [1] "Results of last single trial"
```

```
results <- analyse_communities(comms_i, mui, verbose = TRUE)
```

```
## [1] "Communities found:  28"
## [1] "Modularity:  0.724"
## [1] "Normalized Mutual Information 0.928"
```

```
print("")
```

```
## [1] ""
```

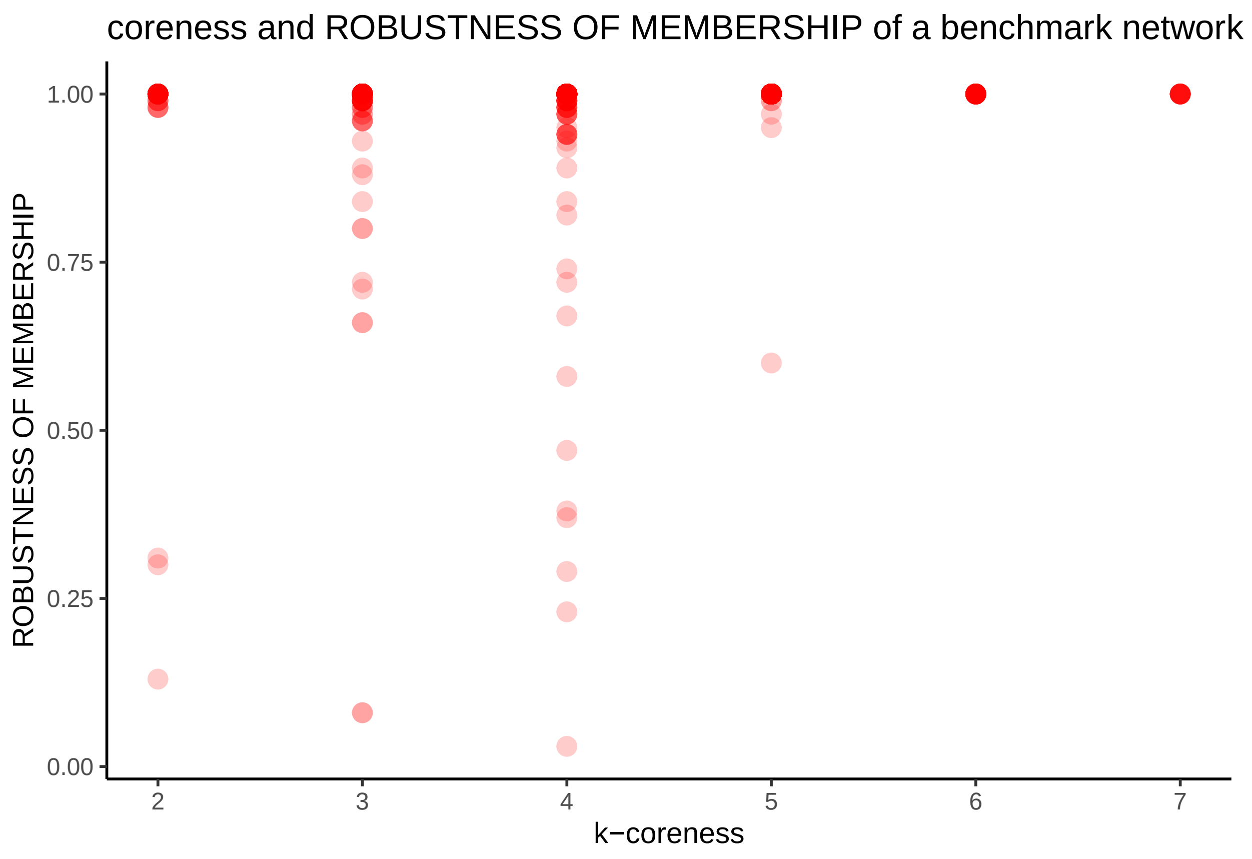
```
print("Results of consensus")
```

```
## [1] "Results of consensus"
```

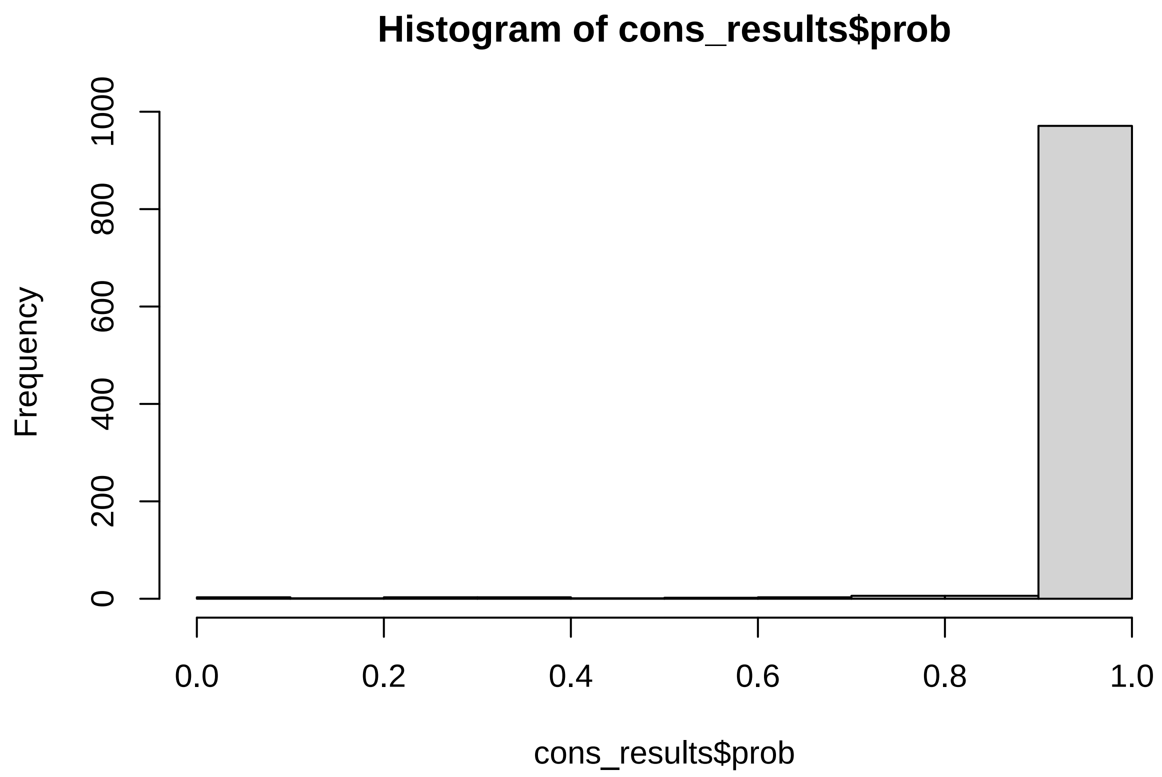
```r
cons_results <- consensus(all_clusters)
cons_communities <- make_clusters(g, array(cons_results$mbshp))
cons_communities$algorithm <- paste0(method,"_cons")
cons_communities$rm <-cons_results$prob

results_cons <- analyse_communities(cons_communities, mui, verbose = TRUE)
```

```
## [1] "Communities found:  20"
## [1] "Modularity:  0.695"
## [1] "Normalized Mutual Information 0.828"
```

```r
V(g)$rm <- cons_results$prob
```



```r
hist(cons_results$prob)
```



## improve consensus by pruning p<0.5 TODO

improving performance by pre-cut on methods that underestimate NC

```
pre_cut_network <- function(g,
                            alpha = 0.05,
                            epsilon = 1 / 1000) {
  g_pre_cut <- g
  n_items <- length(E(g_pre_cut))
  n_null <- as.integer(alpha * n_items)
  applied_weights <- E(g_pre_cut)$ww
  applied_weights[sample(n_items, n_null)] <- epsilon
  E(g_pre_cut)$weight <- applied_weights
  return(g_pre_cut)
}
```

```
g <- load_benchmark_network(mui = 30, path = path, verbose = TRUE)
```

```
## [1] "Loaded benchmark network ./LFR_graphs/LFR/LFR_benchmark_30.gml"
## [1] "Giant component size : 1000"
## [1] "Built-in communities:  37"
## [1] "Modularity of built-in communities:  0.578"
```

```
gp <- pre_cut_network(g,  alpha = 0.05, epsilon = 1/1000)
```

‘ “{r eval=FALSE, include=FALSE} print(“Results of consensus and precut”) cons_results <- consensus(all_clusters_precut) cons_communities <- make_clusters(g, array(cons_results$mbshp))cons_communities$algorithm <- paste0(method,“_cons”) cons_communities$rm <- cons_results$prob

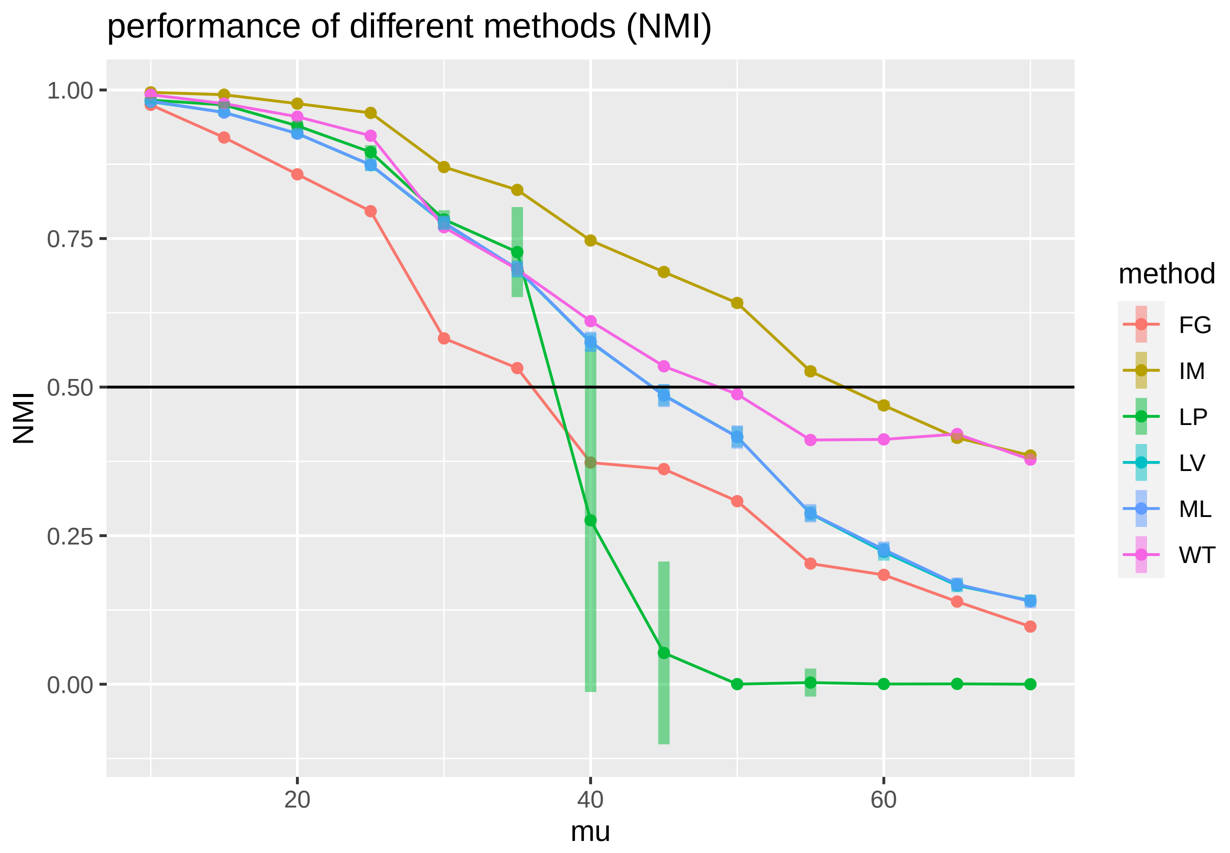
results_cons <- analyse_communities(cons_communities, verbose = TRUE) V(g)$rm <- cons_results$prob

```
‘ ‘‘{r include=FALSE}
data_to_plot_pc<- read_csv(’results_n_trials_precut.csv’)
```

{r} data_to_plot_pc %>% group_by(method, mui) %>% summarise(mean_nmi = mean(n_m_i), sd_nmi = sd(n_m_i)) %>% ggplot(aes(x = mui, y = mean_nmi)) + geom_line(aes(color = method)) + geom_point(aes(color = method)) + geom_linerange(aes(ymin =mean_nmi-sd_nmi, ymax = mean_nmi + sd_nmi, color = method), linewidth = 2, alpha = 0.5)+ geom_hline(yintercept = 0.5)+ labs(title = “performance of different methods + PRE CUT (NMI)”, y = “NMI”, x = “mu”) + theme_gray()

```
‘‘‘r
data_to_plot %>% group_by(method, mui) %>%
    summarise(mean_nmi = mean(n_m_i), sd_nmi = sd(n_m_i) ) %>%
    ggplot(aes(x = mui, y = mean_nmi)) +
    geom_line(aes(color = method)) +
    geom_point(aes(color = method)) +
    geom_linerange(aes(ymin =mean_nmi-sd_nmi, ymax = mean_nmi + sd_nmi, color = method), linewidth = 2,
    geom_hline(yintercept = 0.5)+
    labs(title = "performance of different methods (NMI)", y = "NMI", x = "mu") +
    theme_gray()
```

```
## `summarise()` has grouped output by 'method'. You can override using the
## `.groups` argument.
```



# estimating "robustness of membership" associated with each node

can we exploit the stocasticity to generate a "robustness of membership" associated qith each node? identify node ROLES in a plot x = coreness y = robstness identify Leader / follower / picot / fringe

improving performance by "pruning to"community zero "pre-cuts" inspired by Random Forrest

generating each trial by only a part of the network, with "pre-cuts" as in Random forrest?