Optimizing Network Delays with Bandit Algorithms

Our goal is to minimize the delay, so we consider the negative delay as the "reward." Since bandit algorithms maximize reward, minimizing negative delay achieves the same.

Functions

`simmer_mg1` function: Simulates an M/G/1 queuing system using the simmer package.

Parameters:

- `g`: Graph object representing the network topology.
- `Capacity_Gbps`: Capacity of the links in Gbps.
- `Load`: Load of the system.
- `PS_size`: Packet sizes.
- `PS_weights`: Weights of packet sizes.
- `k_num`: Number of packets to simulate.
- `path_v`: Vector representing the path in the network.

Returns:

- Vector of spending times for the specified number of packets.

```
simmer_mg1 <- function(g, Capacity_Gbps, Load, PS_size, PS_weights, k_num, path_v){

  # Calculate total number of packets in the system
  N = sum(PS_size * PS_weights)

  # Calculate node capacity in Bps
  nodes_capacity_Bps = Capacity_Gbps * 1e9

  # Get edge IDs for the given path
  path_e <- sapply(2:length(path_v), function(i) {
    get.edge.ids(g, as.vector(c(path_v[i-1], path_v[i])))
  })

  # Calculate capacity per packet in Bps
  Capacity_ps = Capacity_Gbps * 1e9 / (8 * N)

  # Calculate traffic per packet in Bps
  traffic_ps = Capacity_ps * Load

  # Calculate service rate
  mu = 1 / Capacity_ps

  # Calculate variance of the number of packets in the system
```

```
  var_N <- sum(PS_size^2 * PS_weights) - N^2
  Cs2 <- var_N / (N^2)

  # Calculate theoretical queue delay for each link in the path
  theor_qeueu_delay_link_mg1 <- mu * Load / (1 - Load) * (1 + Cs2) / 2 + mu
  theor_qeueu_delay_mg1 <- theor_qeueu_delay_link_mg1*length(path_e)
  # Calculate theoretical propagation delay for each link in the path
  theor_prop_delay_link_mg1 <- 5e-6 * E(g)$Distance[path_e]

  # Print total average delay
  cat("Total theoretical average delay", theor_qeueu_delay_mg1 + sum(theor_prop_delay_link_mg1), "s \n"

  # Initialize simulation environment
  env <- simmer()

  # Add resources for each node in the graph
  for (i in 1:length(V(g))) {
    env %>% add_resource(paste0("node_", i), 1)
  }

  # Create trajectory for each link in the path
  trajectory_name <- lapply(1:length(path_e), function(i) {
    trajectory() %>%
      seize(paste0("node_", path_v[i])) %>%
      timeout(function() 8 * sample(PS_size, size = 1, replace = TRUE, prob = PS_weights) / nodes_capac
      release(paste0("node_", path_v[i])) %>%
      timeout(function() theor_prop_delay_link_mg1[i])
  }) %>% join()

  # Define arrival process
  env %>% add_generator("trajectory_name", trajectory_name, function() rexp(1, traffic_ps))

  # Run simulation
  env %>% run(until = (k_num+1e3) / traffic_ps)

  # Get spending times for all arrivals up to k_num
  all_arrivals_res <- data.frame(env %>%
                                   get_mon_arrivals(per_resource = FALSE) %>%
                                   transform(waiting_time_in_queue = round(end_time - start_time - acti
                                   transform(spending_time = end_time - start_time))

  # Return spending times for k_num arrivals
  return(all_arrivals_res$spending_time[1:k_num])

}
```

`get_n_trial_convergence` function: Checks convergence based on average rewards.

The convergence criterion is defined as follows:

$$\text{Err} < \text{threshold}$$

where Err is calculated as:

$$\text{Err} = \frac{\sum_{i=1}^{n} |\overline{\text{Reward\_curr}}_i - \overline{\text{Reward\_prev\_n}}_i|}{|\overline{\text{Reward\_prev\_n}}_i|}$$

Here, $\overline{\text{Reward\_curr}}_i$ represents the average rewards for the current trial, and $\overline{\text{Reward\_prev\_n}}_i$ represents the mean of the previous n average rewards.

```r
get_n_trial_convergence <- function(all_average_rewards, i_trial, conv_num, convergence_threshold) {

  # Check if the current trial is less than or equal to the convergence window
  if (i_trial < conv_num) {
    return(FALSE)  # Not converged yet
  }

  # Retrieve the average rewards for the current trial
  current_average_rewards <-  ifelse(is.na(all_average_rewards[[i_trial]]), 0, all_average_rewards[[i_t

  # Retrieve the previous n average rewards (convergence window)
  prev_n_average_rewards <- all_average_rewards[(i_trial - conv_num):(i_trial - 1)]

  # Calculate the total sum of the previous n average rewards for each iteration
  total_sum <- colSums(do.call(rbind, lapply(prev_n_average_rewards, function(x) ifelse(is.na(x), 0, x))

  # Calculate the mean of the previous n average rewards
  prev_n_mean <- total_sum / conv_num

  # Calculate the error of the current average rewards according the previous n mean
  error_perc <- sum(abs(current_average_rewards - prev_n_mean))/(abs(sum(prev_n_mean)))*1e2

  # Check if the error is less than the convergence threshold
  if (error_perc < convergence_threshold) {
    cat("Error =", error_perc, "% \n")
    cat("Converged at trial", i_trial, "\n")  # Print convergence message
    converged = TRUE
    cat("Converged:",converged)
    return(TRUE)  # Converged
  } else {
    return(FALSE)  # Not converged yet
  }
}
```

Definition of basic parameters

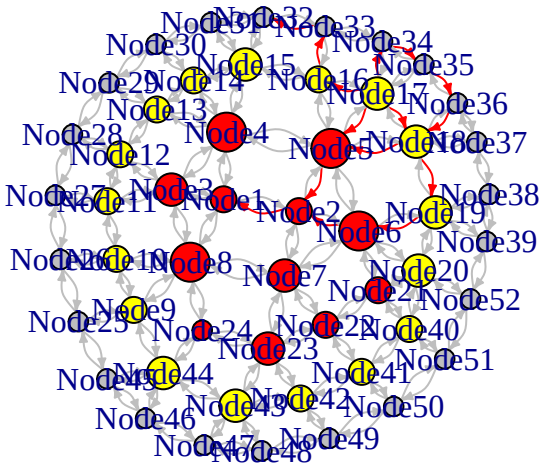
```r
# Path configuration
path1_v = c(17, 5, 2, 1) # From Node17 to Node1
path2_v = c(17, 16, 33, 32) # From Node17 to Node32
path3_v = c(17, 18, 19, 6, 2) # From Node17 to Node2
path4_v = c(17, 34, 35, 36, 18, 5) # From Node17 to Node5
arms = 4 # 4 paths
```

```r
# Parameter definition
n_trials <- 10000 # Number of trials
epsilon <- 0.7 # Exploration rate
PS_size=c((64+127)/2,(128+255)/2,(256+511)/2, (512+1023)/2, (1024+1513)/2, 1514, (1515+9100)/2)
PS_weights=c(33.2/100, 5.4/100, 3.3/100, 3.7/100, 34.6/100, 14.6/100, 5.2/100)
N = sum(PS_size*PS_weights)
```

igraph calculations

```
## National nodes: 1 2 3 4 5 6 7 8 21 22 23 24
```

```
## Regional nodes: 9 10 11 12 13 14 15 16 17 18 19 20 40 41 42 43 44
```




```
## Distance for path1: 2.6 km
```

```
## Distance for path2: 1.4 km
```

```
## Distance for path3: 2.8 km
```

```
## Distance for path4: 2.2 km
```

```r
k_num = n_trials
mg1_packets_path1 <- simmer_mg1(g, Capacity_Gbps = 10, Load = 0.8, PS_size, PS_weights, k_num, path1_v)
```

```
## Total theoretical average delay 2.669341e-05 s
```

```r
mg1_packets_path2 <- simmer_mg1(g, Capacity_Gbps = 10, Load = 0.5, PS_size, PS_weights, k_num, path2_v)
```

```
## Total theoretical average delay 1.225762e-05 s
```

```r
mg1_packets_path3 <- simmer_mg1(g, Capacity_Gbps = 10, Load = 0.4, PS_size, PS_weights, k_num, path3_v)
```

```
## Total theoretical average delay 1.976041e-05 s
```

```r
mg1_packets_path4 <- simmer_mg1(g, Capacity_Gbps = 10, Load = 0.1, PS_size, PS_weights, k_num, path4_v)
```

```
## Total theoretical average delay 1.559687e-05 s
```

```r
cat("Total simulated average delay path1", mean(mg1_packets_path1), "s \n")
```

```
## Total simulated average delay path1 2.704599e-05 s
```

```r
cat("Total simulated average delay path2", mean(mg1_packets_path2), "s \n")
```

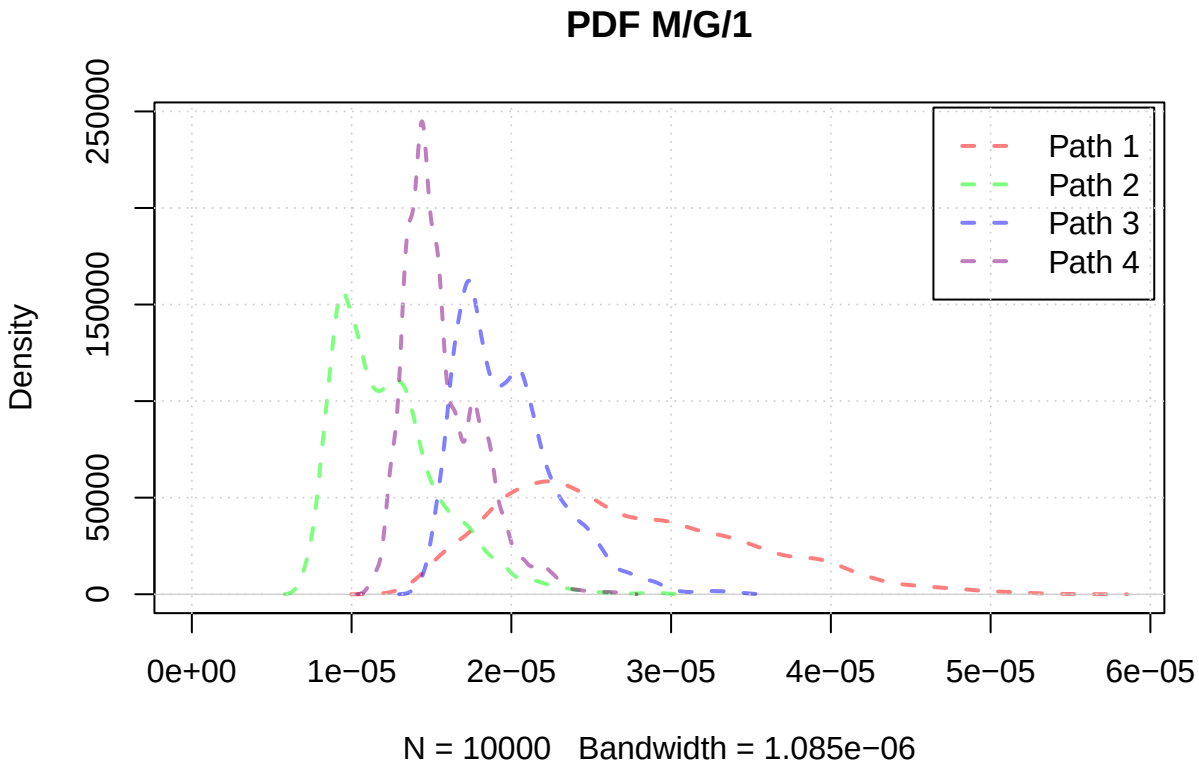
```
## Total simulated average delay path2 1.23116e-05 s
```

```r
cat("Total simulated average delay path3", mean(mg1_packets_path3), "s \n")
```

```
## Total simulated average delay path3 1.974541e-05 s
```

```r
cat("Total simulated average delay path4", mean(mg1_packets_path4), "s \n")
```

```
## Total simulated average delay path4 1.567548e-05 s
```



Simulation for the average delay

Initialization

```
#arms <- length(mu)
counts <- numeric(arms) # A vector initialized with zeros to track how many times each path has been se
rewards <- numeric(arms) # A vector initialized with zeros to accumulate the total rewards (negative de
```

- `conv_num`: This variable represents the number of previous trials used for convergence checking. It specifies the number of previous average rewards to consider when checking for convergence.
- `convergence_threshold`: This variable defines the threshold for convergence, maximum allowed variance between the current average rewards and the mean of the previous `conv_num` average rewards. If the variance falls below this threshold, the system is considered converged.
- `converged`: This variable is a boolean flag that indicates whether the system has converged or not.

```
converged_at_trial <- 0
conv_num <- 5
convergence_threshold <- 1 #%
all_average_rewards <- list()
converged <- FALSE
```

```r
for(i in 1:n_trials){
  # Decide to explore or exploit
  if(runif(1) < epsilon){
    # Exploration: choose a random path
    chosen_arm <- sample(arms, 1)
  } else {
    # Exploitation: choose the best path based on average reward
    #average_rewards <- rewards/pmax(counts, 1)
    average_rewards <- rewards/pmax(counts, 1)
    chosen_arm <- which.max(average_rewards)
  }

  # Simulate the delay (reward) from the chosen path
  reward <- -switch(chosen_arm,
                    sample(mg1_packets_path1,1),
                    sample(mg1_packets_path2,1),
                    sample(mg1_packets_path3,1),
                    sample(mg1_packets_path4,1)) #-rnorm(1, mu[chosen_arm], sigma[chosen_arm])

  # Update counts and rewards
  counts[chosen_arm] <- counts[chosen_arm] + 1
  rewards[chosen_arm] <- rewards[chosen_arm] + reward

  all_average_rewards[[i]] <- rewards/counts

  if (get_n_trial_convergence(all_average_rewards, i, conv_num, convergence_threshold)) {
    break
  }
}
```

```
## Error = 0.725512 %
## Converged at trial 22
## Converged: TRUE
```

Results

```r
average_rewards <- rewards/counts
cat("Counts of selections for each path:", counts, "\n")
```

```
## Counts of selections for each path: 5 8 2 7
```

```r
cat("Average rewards (negative delay) for each path:", average_rewards, "\n")
```

```
## Average rewards (negative delay) for each path: -3.326779e-05 -1.036389e-05 -2.028132e-05 -1.571269e
```

```r
cat("The best path is: Path", which.max(average_rewards), "\n")
```

```
## The best path is: Path 2
```

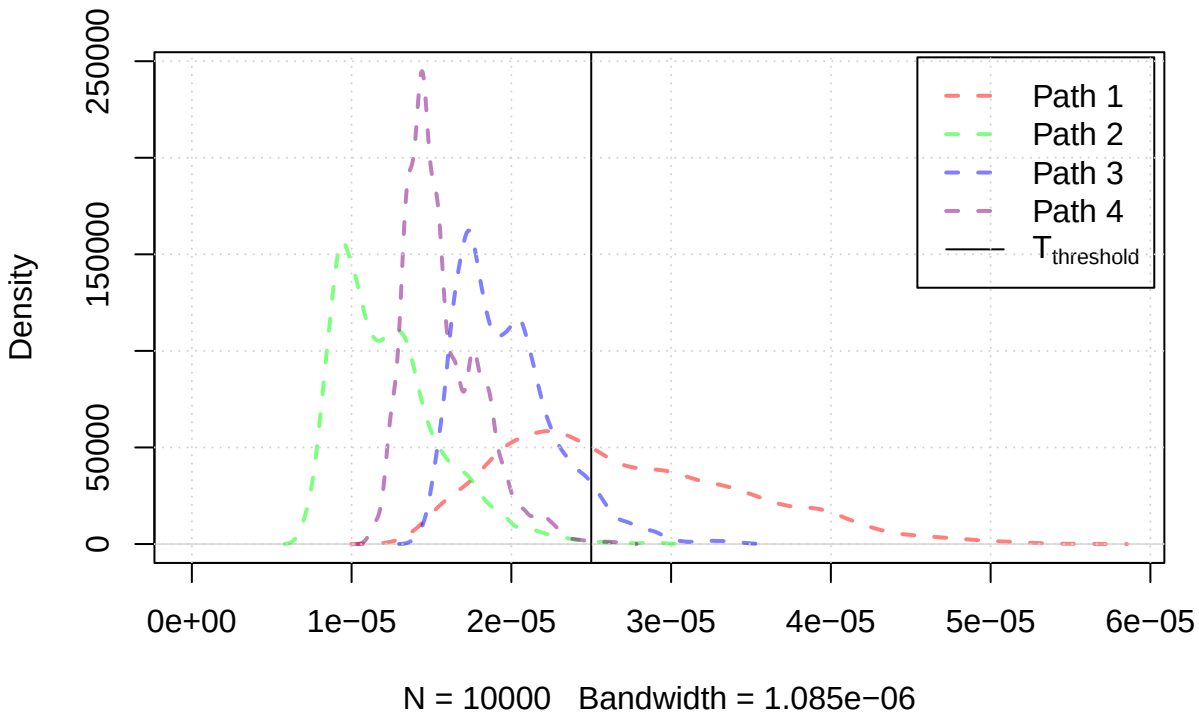
Simulation for the percentiles

Checking if e2e delay is below the given threshold

$T_{\text{threshold}} = 25$ ms

$$\begin{cases} Delay > T_{\text{threshold}}, \text{reward} = -1 \\ Delay < T_{\text{threshold}}, \text{reward} = 0 \end{cases}$$

```
threshold = 2.5e-5 #s
```



Percent above threshold for entire population:

```
print(paste("Percentage above threshold theoretical:", theor_perc(mg1_packets_path1, threshold), "%"))
```

```
## [1] "Percentage above threshold theoretical: 53.13 %"
```

```
print(paste("Percentage above threshold theoretical:", theor_perc(mg1_packets_path2, threshold), "%"))
```

```
## [1] "Percentage above threshold theoretical: 0.31 %"
```

```
print(paste("Percentage above threshold theoretical:", theor_perc(mg1_packets_path3, threshold), "%"))
```

```
## [1] "Percentage above threshold theoretical: 6.93 %"
```

```
print(paste("Percentage above threshold theoretical:", theor_perc(mg1_packets_path4, threshold), "%"))
```

```
## [1] "Percentage above threshold theoretical: 0.21 %"
```

```
counts <- numeric(arms) # A vector initialized with zeros to track how many times each path has been se
penalties <- numeric(arms) # A vector initialized with zeros to accumulate the total penalties (negativ

converged_at_trial <- 0
conv_num <- 25
convergence_threshold <- 1 #%
all_average_penalties <- list()
converged <- FALSE

for(i in 1:n_trials){
  # Decide to explore or exploit
  if(runif(1) < epsilon){
    # Exploration: choose a random path
    chosen_arm <- sample(arms, 1)
  } else {
    # Exploitation: choose the best path based on average penalty
    average_penalties <- penalties/pmax(counts, 1)
    chosen_arm <- which.max(average_penalties)
  }

  # Simulate the delay (penalty) from the chosen path
  delay <- switch(chosen_arm,
                  sample(mg1_packets_path1,1),
                  sample(mg1_packets_path2,1),
                  sample(mg1_packets_path3,1),
                  sample(mg1_packets_path4,1))

  # Update counts and penalties
  penalty = -ifelse(delay > threshold, 1, 0)
  counts[chosen_arm] <- counts[chosen_arm] + 1
  penalties[chosen_arm] <- penalties[chosen_arm] + penalty

  all_average_penalties[[i]] <- penalties/counts

  if (get_n_trial_convergence(all_average_penalties, i, conv_num, convergence_threshold)) {
    break
  }
}
```

```
## Error = 0.7092199 %
## Converged at trial 54
## Converged: TRUE
```

Results

```r
average_rewards <- penalties/counts
cat("Counts of selections for each path:", counts, "\n")
```

```
## Counts of selections for each path: 9 20 12 13
```

```r
cat("Total rewards (negative delay) for each path:", penalties, "\n")
```

```
## Total rewards (negative delay) for each path: -4 0 0 0
```

```r
cat("Average rewards (negative delay) for each path:", average_penalties, "\n")
```

```
## Average rewards (negative delay) for each path: -0.4444444 0 0 0
```

```r
cat("Percent of packets above threshold simulated:", -average_penalties*1e2, "% \n")
```

```
## Percent of packets above threshold simulated: 44.44444 0 0 0 %
```

```r
cat("Percent of packets above threshold theoretical:", theor_perc(mg1_packets_path1, threshold),theor_p
```

```
## Percent of packets above threshold theoretical: 53.13 0.31 6.93 0.21 %
```

```r
cat("The best path is: Path", which.max(average_penalties), "\n")
```

```
## The best path is: Path 2
```