The objective is to simulate the behavior of static approaches with probabilistic communication costs (that are either $d$ with a probability $p$ or $D$).

1 Tree construction

```r
tree_build <- function(n, d, c) {
    parent <- array(0, n)
    soonest <- array(0, n)
    soonest[n] <- c
    for (i in (n - 1):1) {
        minCost <- Inf
        for (j in (i + 1):n) {
            currentCost <- soonest[j] + d
            if (currentCost < minCost) {
                minCost <- currentCost
                bestSucc <- j
            }
        }
        soonest[i] <- minCost + c
        parent[i] <- bestSucc
    }
    return(parent)
}

Tests:

tree_build(4, 1, 0)
## [1] 4 3 4 0
tree_build(8, 1, 0)
## [1] 8 7 6 5 8 7 8 0
tree_build(8, 1, 1)
## [1] 8 7 6 8 7 8 8 0

This corresponds indeed to binomial and Fibonacci trees.
2 MC simulator for static approaches

Now, let’s define a function for evaluating trees:

```r
static_MC <- function(T_orig, rdist, iterations) {
  result <- NULL
  for (k in 1:iterations) {
    T <- T_orig
    L <- array(0, length(T))
    while (any(T != 1:length(T))) {
      # Select leaves
      leaves <- which(!(1:length(T) %in% T))
      # Find the next that is reay
      current <- leaves[which.min(L[leaves])]
      if (T[current] != 0)
        L[T[current]] <- max(L[T[current]], L[current]) + rdist(1)
      T[current] <- current
    }
    result <- c(result, max(L))
  }
  return(result)
}
```

First, let’s define a general way of specifying the communication costs (we choose \(d = 11\) and \(D = 29\), which are prime numbers):

```r
n <- 8
T_bino <- tree_build(n, 1, 0)
T_fibo <- tree_build(n, 1, 1)
iterations <- 1e+05
# Two prime numbers with a large lcm
rdist <- function(n, p = 0.5, d = 11, D = 29) {
  return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
```

```r
table(static_MC(T_bino, rdist, iterations))/iterations
##
## 33 44 51 62 69 80 87
## 0.00815 0.00830 0.18761 0.01529 0.47356 0.00775 0.29934
```

```r
table(static_MC(T_fibo, rdist, iterations))/iterations
##
## 44 51 62 69 80 87 98 116
## 0.01600 0.03180 0.13988 0.10983 0.31269 0.07648 0.25049 0.06283
```

# List of possible values (through linear combinations)
```r
d <- 11
D <- 29
d * log2(n):0 + D * 0:log2(n)
## [1] 33 51 69 87
```

```r
sort(colSums(c(d, D) * t(expand.grid(0:log2(n), 0:log2(n)))))
## [1] 0 11 22 29 33 40 51 58 62 69 80 87 91 98 109 120
```
3 MC simulator for dynamic approaches

```r
dyn_MC <- function(n, rdist, iterations) {
  result <- NULL
  for (k in 1:iterations) {
    comm <- rdist(n/2)
    available <- FALSE
    while (length(comm) > 1 || available) {
      current <- which.min(comm)
      if (available)
        comm[current] <- comm[current] + rdist(1) else comm <- comm[-current]
      available <- !available
    }
    result <- c(result, comm[1])
  }
  return(result)
}

dyn_com_MC <- function(n, rdist, iterations) {
  result <- NULL
  for (k in 1:iterations) {
    comm <- rdist(n/2)
    while (length(comm) > 1) {
      current <- which.min(comm)
      first <- TRUE
      if (current < length(comm) & current + 1 == Inf & current > 1 & current - 1 == Inf & rbinom(1, 1, 0.5) == 1)
        first <- FALSE
      if (current < length(comm) & current + 1 == Inf & first) {
        comm[current] <- comm[current] + rdist(1)
        comm <- comm[-(current + 1)]
      } else if (current > 1 & current - 1 == Inf) {
        comm[current] <- comm[current] + rdist(1)
        comm <- comm[-(current - 1)]
      } else {
        comm[current] <- Inf
      }
    }
    result <- c(result, comm[1])
  }
  return(result)
}

Some preliminary tests (checking that both methods are equivalent with two values):

iterations <- 1e+05
dist <- function(n, p = 0.5, d = 11, D = 29) {
  return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
table(dyn_MC(2, rdist, iterations))/iterations
```

#
Let’s study the possible values for a slightly more complex scenario \((n = 8)\):

\[
\begin{align*}
\text{n} & \leftarrow 8 \\
\text{iterations} & \leftarrow 1e+05 \\
\text{table(dyn_MC(n, rdist, iterations))}/\text{iterations} & \\
\end{align*}
\]

\[
\begin{align*}
\text{##} & \quad \text{33} & \quad \text{44} & \quad \text{51} & \quad \text{62} & \quad \text{69} & \quad \text{80} & \quad \text{87} \\
\text{## 0.00836} & \quad 0.03018 & \quad 0.19448 & \quad 0.06289 & \quad 0.42875 & \quad 0.03119 & \quad 0.24415 \\
\\text{table(dyn_com_MC(n, rdist, iterations))}/\text{iterations} & \\
\\end{align*}
\]

\[
\begin{align*}
\text{##} & \quad \text{33} & \quad \text{44} & \quad \text{51} & \quad \text{62} & \quad \text{69} & \quad \text{80} & \quad \text{87} & \quad \text{98} & \quad \text{116} \\
\text{## 0.00781} & \quad 0.01545 & \quad 0.16244 & \quad 0.05454 & \quad 0.39982 & \quad 0.07170 & \quad 0.24166 & \quad 0.03843 & \quad 0.00815 \\
\\end{align*}
\]

The most efficient execution is when costs are all \(d\) and the execution is equivalent to a binomial. Inversely, the less efficient execution is when costs are all \(D\). There is 7 possible total durations.

## Comparisons

\[
\begin{align*}
\text{compare_approaches <- function(n, iterations, rdist) \{} \\
\text{ \quad bino <- static_MC(tree_build(n, 1, 0), rdist, iterations)} \\
\text{ \quad fibo <- static_MC(tree_build(n, 1, 1), rdist, iterations)} \\
\text{ \quad dyn <- dyn_MC(n, rdist, iterations)} \\
\text{ \quad dyn_com <- dyn_com_MC(n, rdist, iterations)} \\
\text{ \quad plot.ecdf(bino, verticals = TRUE, cex = 0.3, xlim = range(c(bino, fibo,} \\
\text{ \quad \quad dyn, dyn_com)))} \\
\text{ \quad plot.ecdf(fibo, verticals = TRUE, cex = 0.3, add = TRUE, col = 2)} \\
\text{ \quad plot.ecdf(dyn, verticals = TRUE, cex = 0.3, add = TRUE, col = 3)} \\
\text{ \quad plot.ecdf(dyn_com, verticals = TRUE, cex = 0.3, add = TRUE, col = 4)} \\
\text{\}} \\
\end{align*}
\]

\[
\begin{align*}
\text{n} & \leftarrow 64 \\
\text{iterations} & \leftarrow 10000 \\
\text{rdist <- function(n, p = 0.5, d = 11, D = 29) \{} \\
\text{ \quad return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))} \\
\text{\}} \\
\text{compare_approaches(n, iterations, rdist)}
\end{align*}
\]
Now, if the variability is higher, both dynamic approaches should be better:

```r
n <- 64
iterations <- 10000
rdist <- function(n, p = 0.5, d = 11, D = 89) {
  return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
compare_approaches(n, iterations, rdist)
```
The dynamic approaches are both better and the Fibo one is the worse. If the variability is low:

\begin{verbatim}
n <- 64
iterations <- 10000
rdist <- function(n, p = 0.5, d = 11, D = 13) {
    return(sample(c(d, D), n, replace = TRUE, prob = c(p, 1 - p)))
}
compare_approaches(n, iterations, rdist)
\end{verbatim}
Bino and dynamic are equivalent. Fibo is still worse than dynamic with non-commutative operations.

5 Conclusion

- Study all previous notes on the Markov approach for checking the MC relevance (problem in the dyn code).
- Introduce computation costs.