The 3.8.0 release of simmer, the Discrete-Event Simulator for R, hit CRAN almost a week ago, and Windows binaries are already available. This version includes two highly requested new features that justify this second consecutive minor release.

**Attachment of precomputed data**

Until v3.7.0, the *generator* was the only means to attach data to trajectories, and it was primarily intended for dynamic generation of arrivals:

library(simmer)

set.seed(42)

hello\_sayer <- trajectory() %>%

log\_("hello!")

simmer() %>%

add\_generator("dummy", hello\_sayer, function() rexp(1, 1)) %>%

run(until=2)

## 0.198337: dummy0: hello!

## 0.859232: dummy1: hello!

## 1.14272: dummy2: hello!

## 1.18091: dummy3: hello!

## 1.65409: dummy4: hello!

## simmer environment: anonymous | now: 2 | next: 3.11771876826972

## { Monitor: in memory }

## { Source: dummy | monitored: 1 | n\_generated: 6 }

Although it may be used to attach precomputed data too, especially using the at() adaptor:

simmer() %>%

add\_generator("dummy", hello\_sayer, at(seq(0, 10, 0.5))) %>%

run(until=2)

## 0: dummy0: hello!

## 0.5: dummy1: hello!

## 1: dummy2: hello!

## 1.5: dummy3: hello!

## simmer environment: anonymous | now: 2 | next: 2

## { Monitor: in memory }

## { Source: dummy | monitored: 1 | n\_generated: 21 }

Now, let’s say that we want to attach some empirical data, and our observations not only include arrival times, but also priorities and some attributes (e.g., measured service times) :

Using Simmer to Build Specific Model

I've got data on actual events and I need to model what might have happened if different resources were available. The next stage will be to build a "proper" simulation where events and times are created more randomly. My problem is that I can't work out how to ensure a specific activity gets assigned the start time, priority and timeout which it had in real life.

library(simmer)

set.seed(654)

env <- simmer()

workerCount <- 2

actualData <- data.frame(arrTime = c(1:10,1:5),

priority = 1:3, duration = rnorm(15, 50, 5))

activityTraj <- trajectory() %>%

seize('worker') %>%

timeout(5) %>%

release('worker')

env %>%

add\_resource('worker', workerCount, Inf, preemptive = TRUE) %>%

add\_generator('worker', activityTraj, at(actualData$arrTime),

mon = 2, priority = 2)

env %>% run(50)

What I need to do in the above is to make the priority in the generator read from the data frame (currently hard coded at 2) and the timeout (currently hard coded at 5) in the trajectory also read from the data frame. I can't see how I can ensure that the row that specifies the priority and time of the activity will also be used to specify the duration (or "timeout").

First of all, you must ensure that your actualData frame is sorted by arrTime:

actualData <- data.frame(arrTime = c(1:10,1:5),

priority = 1:3,

duration = rnorm(15, 50, 5)) %>%

dplyr::arrange(arrTime)

Then, let's build a helper function to consume the columns of your actualData:

consume <- function(x, prio=FALSE) {

i <- 0

function() {

i <<- i + 1

if (prio) c(x[[i]], x[[i]], FALSE)

else x[[i]]

}

}

which can be applied to your trajectory as follows:

activityTraj <- trajectory() %>%

set\_prioritization(consume(actualData$priority, TRUE)) %>%

set\_attribute("duration", consume(actualData$duration)) %>%

seize('worker') %>%

timeout(function(attr) attr["duration"]) %>%

release('worker')

because your arrivals are sorted. Finally, let's run the simulation:

env %>%

add\_resource('worker', workerCount, Inf, preemptive = TRUE) %>%

add\_generator('worker\_', activityTraj, at(actualData$arrTime)) %>%

run()

and check that the actual durations were ok:

activity\_time <- get\_mon\_arrivals(env) %>%

tidyr::separate(name, c("prefix", "n"), convert=TRUE) %>%

dplyr::arrange(n) %>%

dplyr::pull(activity\_time)

all(activity\_time == actualData$duration)

#> TRUE

**UPDATE**: Since simmer v3.8.0, the new data source add\_dataframe greatly simplifies this kind of pattern:

library(simmer)

workerCount <- 2

actualData <- data.frame(

time = c(1:10,1:5), priority = 1:3, service = rnorm(15, 50, 5)) %>%

dplyr::arrange(time)

activityTraj <- trajectory() %>%

seize('worker') %>%

timeout\_from\_attribute("service") %>%

release('worker')

env <- simmer() %>%

add\_resource('worker', workerCount, Inf, preemptive = TRUE) %>%

add\_dataframe('worker\_', activityTraj, actualData, time="absolute") %>%

run()

activity\_time <- get\_mon\_arrivals(env) %>%

tidyr::separate(name, c("prefix", "n"), convert=TRUE) %>%

dplyr::arrange(n) %>%

dplyr::pull(activity\_time)

all(activity\_time == actualData$duration)

#> TRUE

myData <- data.frame(

time = c(1:10,1:5),

priority = 1:3,

duration = rnorm(15, 50, 5)) %>%

dplyr::arrange(time)

This is indeed possible using generators, but it requires some trickery; more specifically, the clever usage of a *consumer* function as follows:

consume <- function(x, prio=FALSE) {

i <- 0

function() {

i <<- i + 1

if (prio) c(x[[i]], x[[i]], FALSE)

else x[[i]]

}

}

activityTraj <- trajectory() %>%

seize("worker") %>%

timeout\_from\_attribute("duration") %>%

release("worker")

initialization <- trajectory() %>%

set\_prioritization(consume(myData$priority, TRUE)) %>%

set\_attribute("duration", consume(myData$duration)) %>%

join(activityTraj)

arrivals\_gen <- simmer() %>%

add\_resource("worker", 2, preemptive=TRUE) %>%

add\_generator("dummy\_", initialization, at(myData$time)) %>%

run() %>%

get\_mon\_arrivals()

# check the resulting duration times

activity\_time <- arrivals\_gen %>%

tidyr::separate(name, c("prefix", "n"), convert=TRUE) %>%

dplyr::arrange(n) %>%

dplyr::pull(activity\_time)

all(activity\_time == myData$duration)

## [1] TRUE

Since this v3.8.0, the new data source add\_dataframe greatly simplifies this process:

arrivals\_df <- simmer() %>%

add\_resource("worker", 2, preemptive=TRUE) %>%

add\_dataframe("dummy\_", activityTraj, myData, time="absolute") %>%

run() %>%

get\_mon\_arrivals()

identical(arrivals\_gen, arrivals\_df)

## [1] TRUE

**On-disk monitoring**

As some users noted , the default in-memory monitoring capabilities can turn problematic for very long simulations. To address this issue, the simmer() constructor gains a new argument, mon, to provide different types of monitors. Monitoring is still performed in-memory by default, but as of v3.8.0, it can be offloaded to disk through monitor\_delim() and monitor\_csv(), which produce flat delimited files.

mon <- monitor\_csv()

mon

## simmer monitor: to disk (delimited files)

## { arrivals: /tmp/RtmpAlQH2g/file6933ce99281\_arrivals.csv }

## { releases: /tmp/RtmpAlQH2g/file6933ce99281\_releases.csv }

## { attributes: /tmp/RtmpAlQH2g/file6933ce99281\_attributes.csv }

## { resources: /tmp/RtmpAlQH2g/file6933ce99281\_resources.csv }

env <- simmer(mon=mon) %>%

add\_generator("dummy", hello\_sayer, function() rexp(1, 1)) %>%

run(until=2)

## 0.26309: dummy0: hello!

## 0.982183: dummy1: hello!

env

## simmer environment: anonymous | now: 2 | next: 2.29067480322535

## { Monitor: to disk (delimited files) }

## { arrivals: /tmp/RtmpAlQH2g/file6933ce99281\_arrivals.csv }

## { releases: /tmp/RtmpAlQH2g/file6933ce99281\_releases.csv }

## { attributes: /tmp/RtmpAlQH2g/file6933ce99281\_attributes.csv }

## { resources: /tmp/RtmpAlQH2g/file6933ce99281\_resources.csv }

## { Source: dummy | monitored: 1 | n\_generated: 3 }

read.csv(mon$handlers["arrivals"]) # direct access

## name start\_time end\_time activity\_time finished

## 1 dummy0 0.2630904 0.2630904 0 1

## 2 dummy1 0.9821828 0.9821828 0 1

get\_mon\_arrivals(env) # adds the "replication" column

## name start\_time end\_time activity\_time finished replication

## 1 dummy0 0.2630904 0.2630904 0 1 1

## 2 dummy1 0.9821828 0.9821828 0 1 1

See below for a comprehensive list of changes.

**New features:**

* New data source add\_dataframe enables the attachment of precomputed data, in the form of a data frame, to a trajectory. It can be used instead of (or along with) add\_generator. The most notable advantage over the latter is that add\_dataframe is able to automatically set attributes and prioritisation values per arrival based on columns of the provided data frame (#140 closing #123).
* New set\_source activity deprecates set\_distribution(). It works both for generators and data sources (275a09c, as part of #140).
* New monitoring interface allows for disk offloading. The simmer() constructor gains a new argument mon to provide different types of monitors. By default, monitoring is performed in-memory, as usual. Additionally, monitoring can be offloaded to disk through monitor\_delim and monitor\_csv, which produce flat delimited files. But more importantly, the C++ interface has been refactorised to enable the development of new monitoring backends (#146 closing #119).

**Minor changes and fixes:**

* Some documentation improvements (1e14ed7, 194ed05).
* New default until=Inf for the run method (3e6aae9, as part of #140).
* branch and clone now accept lists of trajectories, in the same way as join, so that there is no need to use do.call (#142).
* The argument continue (present in seize and branch) is recycled if only one value is provided but several sub-trajectories are defined (#143).
* Fix process reset: sources are reset in strict order of creation (e7d909b).
* Fix infinite timeouts (#144).