Hospital Capacity Planning Using Discrete Event Simulation: Numerical Methods

2021-03-03

Abstract

Resource planning for hospitals under special consideration of the COVID-19 pandemic.

Introduction

- Resource and capacity planning for hospitals.
- Paper: A novel modeling technique to predict resource-requirements in critical care: a case study (Lawton, McCooe) see (Lawt19a).

The challenge

A sector as critical as health sector experiencing unplanned demand for resources can lead to devastating loss. This challenge was experienced heavily around the world since the COVID19 pandamic started. Though, the challenge has always been there in the health sector, the pandemic amplified it.

Lack of suitable scientific tool and for modeling, predicting and planning for resources usage in the hospital sector has made the challenge more difficult to deal with. Hence, the need for a suitable scientic/data analysis tool to deal with this. This is the reason and idea of the **Babsim Resource Planning tool**.

The tool

The Babsim R package is tool for Simulation of resource allocation in hospitals.According to Lawton and McCooe paper on the same subject, *"There are a variety of ways in which ICU capacity, and other hospital processes, can be modelled. These can be categorised into two main types: top-down (often 'system dynamics' or 'time series'), and bottom-up (usually 'discrete event' or 'agent-based') approaches"* The tool uses the second method, which is Discrete event approach to do the simulation.

The method

A discrete-event simulation (DES) which is used in this simulator models the operation of a system as a (discrete) sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur; thus the simulation time can directly jump to the occurrence time of the next event, which is called next-event time progression.

Packages

SPOT: A set of tools for model-based optimization and tuning of algorithms.

Simmer: Simmer is a process-oriented and trajectory-based Discrete-Event Simulation (DES) package for R.

The required packages are installed if not already exists.As shown below, the packages are devtools for working with git, spot for working with spot and the babsim package.

```
#install.packages("devtools")
#devtools::install_github("r-lib/devtools")
```

```
#url <- "http://owos.gm.fh-koeln.de:8055/bartz/spot.git"
#devtools::install_git(url = url)
```

```
#url <- "http://owos.gm.fh-koeln.de:8055/bartz/babsim.hospital"
#devtools::install_git(url = url, subdir = "babsim.hospital")
```

- The required packages are loaded into the current workspace/environment.

```
rm(list = ls())
suppressPackageStartupMessages({
library("SPOT")
library("babsim.hospital")
library("simmer")
library("simmer.plot")
library("plotly")
})
```

- The package version of SPOT is checked since not all version could be used in the program. Package version of `SPOT` must be larger than `2.2.4`:
- Package version of `SPOT` must be larger than `2.2.4`:

```
packageVer
# Motivation

* Crises like the COVID-19 pandemic pose a serious challenge to health care institutions.
* They need to determine and plan the resources required for handling the increased load,
for instance in terms of hospital bedssion("SPOT")
```

and ventilators. After consultation with the local authorities of the Oberbergisches Land District, the babsimhospital tool was created to address the resource planning challenges the public authorities are facing. * This is a tool for capacity planning based on discrete event simulation that aims to address this challenge. * The predictive quality of the simulation is based on a set of 29 parameters. * The default values of these parameters were established in cooperation with medical professionals. * We aim to investigate and optimize these parameters to improve babsimhospital. * To that end, we use model-based optimization via SPOT and an in-depth sensitivity analysis. * The sensitivity analysis is crucial for the optimization process, since it allows to focus the search on the most important parameters of the simulation. * We illustrate that this approach allows for a reduction in the dimensions of the simulation without compromising the resulting accuracy.

Data used by babsim.hospital

Data Sources

- We combine data from two different sources:
    1. `simData`: simulation data, i.e., input data for the simulation. Here, we will use data from UK.
    2. `fieldData`: real data, i.e., data from the DIVI-Intensivregister. The field data will be used for validating the simulation output.
- Statistically speaking, the `babsim.hospital` simulator models resources usage in hospitals, e.g., number of ICU beds (y), as a function of the number of infected individuals (x).
- In addition to the number of infections, information about age and gender can be used as simulation input.
- You will take a closer look at these data in the following sections. Therefore, we have provided som real data from the UK.

Simulation Data: UK Data

- Read data from Excel file.
- WARNING:
  - Please change the path information with respect to your local setting!
- The CovidData is loaded or read into dataframe/tibble using read_excel function of readxl library.

```
library(readxl)
### X20201111UKdata <- read_excel("/Users/bartz/workspace/Lehre.d/IDEA-Master-AIT-WS2020-2021/Numerical
X20201111UKdata <- X20201111UKdata <- read_excel("CovidDataGroup3.xlsx")
ukdataRawFull1 <- X20201111UKdata
```

- The Str function of R is used to compactly display the structure of the uk data.

```
str(ukdataRawFull1)
%> tibble [240 x 5] (S3: tbl_df/tbl/data.frame)
%>  $ Date                  : POSIXct[1:240], format: "2020-03-16" "2020-03-17" ...
%>  $ TotalCOVID19Inpatient : num [1:240] 0 2 3 0 3 10 13 19 22 34 ...
%>  $ COVID19NonInvasiveCPAP: num [1:240] 0 0 0 0 0 0 0 0 0 0 ...
%>  $ COVID19VentilatedICU  : num [1:240] 0 0 0 0 0 0 0 0 0 0 ...
%>  $ NewCases              : num [1:240] 0 5 3 0 3 5 5 10 14 11 ...
```

- Date
- `Infected`: New cases = new cases either admitted to hospital or diagnosed in the hospital
- Deaths = deaths in hospital
- Discharges = successful discharge to home or other location from hospital
- Confirmed cases currently in hospital – by age range (nb excludes suspected cases I think)
- `bed`: Total COVID inpatient = total in hospital (includes ICU)
- `intensiveBed`: COVID-19 Non-invasive = patients on non-invasive ventilators (CPAP). In theory these should be on ICU but we don't have space for them all. The model should probably consider them to be ICU but not intubated (level 2)
- `intensiveBedVentilated`: COVID-19 Ventilated = patients ventilated and intubated on ICU
- `NewCasesUK`: New cases in UK = new cases diagnosed across the city. In March/April the government testing was very poor so there were a *lot* of undiagnosed cases in the UK. Currently we think we're diagnosing 30-50% of cases.
- Cumulative cases in UK = cumulative version of above

Consider Second Wave

- The UkdataRawFull1 is filtered to get the those whose dates is after September 1st 2020, which is the period assumed second wave started. The data structure is displayed with str function.

```
ukdataRaw <- ukdataRawFull1[as.Date(ukdataRawFull1$Date) > "2020-09-01",]
str(ukdataRaw)
%> tibble [70 x 5] (S3: tbl_df/tbl/data.frame)
%>  $ Date                  : POSIXct[1:70], format: "2020-09-02" "2020-09-03" ...
%>  $ TotalCOVID19Inpatient : num [1:70] 18 21 21 19 22 19 18 19 21 22 ...
%>  $ COVID19NonInvasiveCPAP: num [1:70] 3 5 5 5 5 5 6 5 5 5 ...
%>  $ COVID19VentilatedICU  : num [1:70] 2 2 2 2 2 2 2 2 2 0 ...
%>  $ NewCases              : num [1:70] 112 93 91 98 114 182 82 122 152 126 ...
```

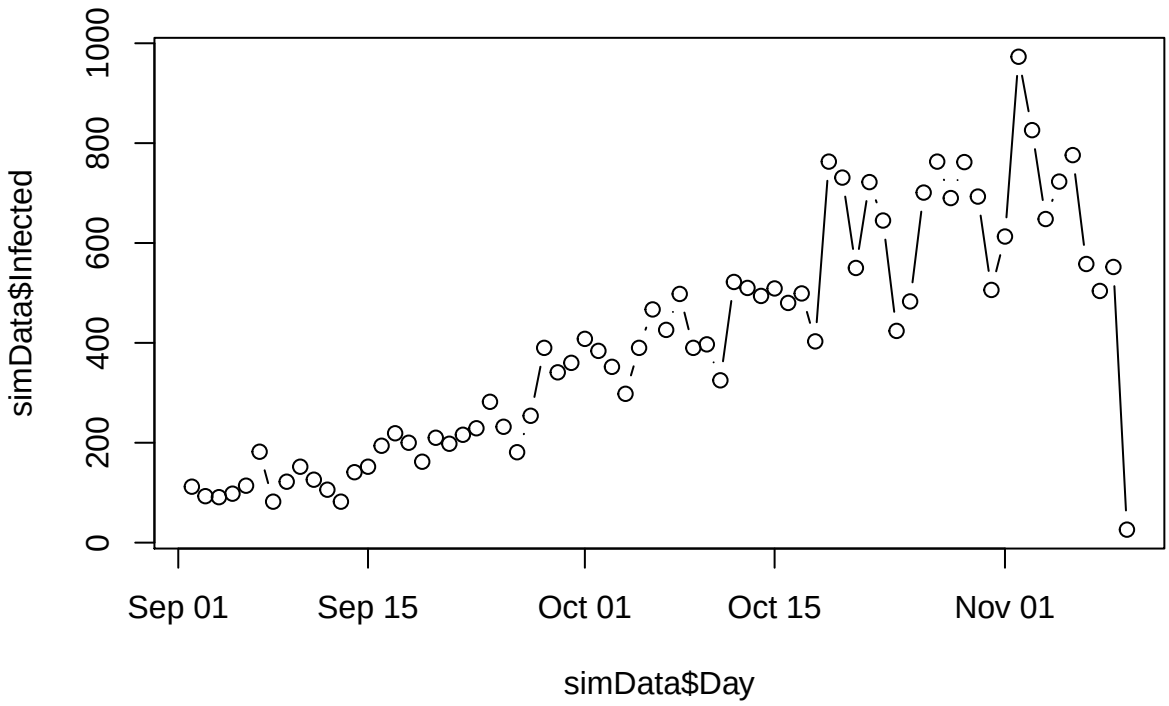
- The simulation data frame is generated from the UkDataRaw. it has two headers, Day which is the date of infection report and Infected which is the number of new cases.

```
simData <- data.frame(Day = as.Date(ukdataRaw$Date),
                      Infected = ukdataRaw$NewCases)
```

- The number of infected is plotted against dates.As can be seen from the graph, the number of infected

was generally increasing.

```
plot(simData$Infected ~ simData$Day, type="b")
```



Field Data (Real ICU Beds)

Preprocessing TL ICU Data

- Note: non ICU patients can be calculated as `TotalCOVID19Inpatient - COVID19NonInvasiveCPAP - COVID19VentilatedICU`
- We convert the data into a data.frame with
    1. Day
    2. bed
    3. intensiveBed,
    4. intensiveBedVentilation, and
    5. Day

```
Day <- as.Date(ukdataRaw$Date)
bed <- ukdataRaw$TotalCOVID19Inpatient - ukdataRaw$COVID19NonInvasiveCPAP - ukdataRaw$COVID19VentilatedI
intensiveBed <-  ukdataRaw$COVID19NonInvasiveCPAP
intensiveBedVentilation <- ukdataRaw$COVID19VentilatedICU
fieldData <- data.frame(Day = Day,
                        bed = bed,
                        intensiveBed = intensiveBed,
                        intensiveBedVentilation = intensiveBedVentilation)
str(fieldData)
```

```
%> 'data.frame':    70 obs. of  4 variables:
%>  $ Day                   : Date, format: "2020-09-02" "2020-09-03" ...
%>  $ bed                   : num  13 14 14 12 15 12 10 12 14 17 ...
%>  $ intensiveBed          : num  3 5 5 5 5 5 6 5 5 5 ...
%>  $ intensiveBedVentilation: num  2 2 2 2 2 2 2 2 2 0 ...
```

```
max(fieldData$intensiveBed)
%> [1] 64
```

- The field data based on UK data used three bed categories:
  1. `bed`: non ICU patients in hospital
  2. `intensiveBed`: ICU bed without ventilation
  3. `intensiveBedVentilation`: ICU bed with ventilation

Performing Simulations

- To perform a simulation, the setting must be configured (seed, number of repeats, sequential or parallel evaluation, variable names, dates, etc.)
- To perform a simulation, the setting must be configured (seed, number of repeats, sequential or parallel evaluation, variable names, dates, etc.)
- Seed ensures the same random number is generated
- parallel determines how the simulation will be run, parallel or not
- percCores controls the utilization of the computer processor
- the resourceNames is list of the names resources to be modeled
- resourceEval is similar to the resourceNames, it shows the actual title of the resources to be modeled.

```
seed = 123
simrepeats = 2
parallel = TRUE
percCores = 0.8
resourceNames =  c("bed", "intensiveBed", "intensiveBedVentilation")
resourceEval = c("bed", "intensiveBed", "intensiveBedVentilation")
```

- Other configuration is performed, the start date for simulation is determined. The fielddata is initialized with null and the icu (requirement) is initialized as false. The Weight of different bed types is also specified. As can be seen, the resources weight of the intensive care bed is very high compared to others.
- We can specify the field data based on `ukdataRaw` for the simulation as follows:

```
FieldStartDate = as.Date(min(fieldData$Day))
rownames(fieldData) <- NULL
icu = FALSE
icuWeights = c(1,2,10)
```

- Next, simulation data (RKI data) can be selected. The simulation data in our example, depend on the field data:

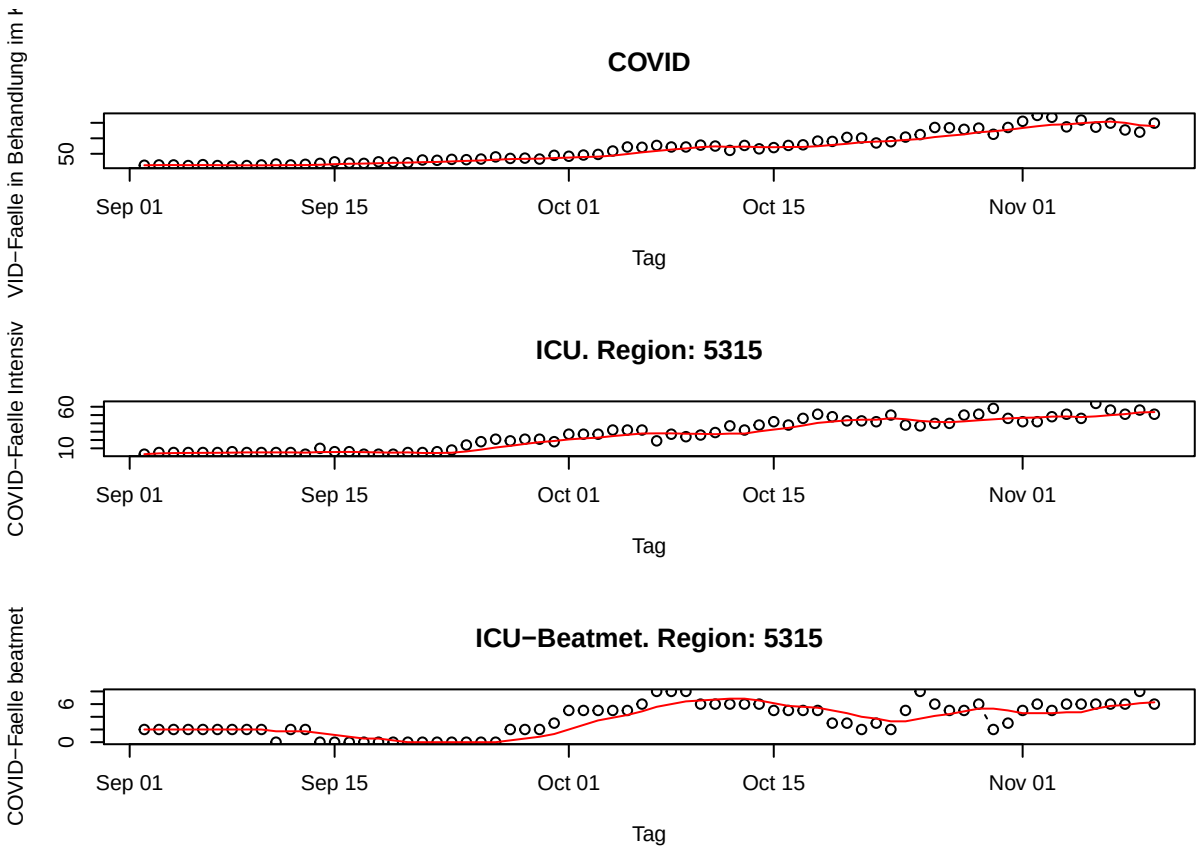
```
SimStartDate = FieldStartDate
```

- Finally, we combine all data in one data frame `data`:

```
data <- list(simData = simData,
 fieldData = fieldData)
```

- We Visualize the data. The visualizeUk is a special plotting function that takes in two parameters, the dataframe to plot, and the region to plot. If the region is not provided a default region of 05315 is used. Basically, the region is used to set the plot main (title)

```
visualizeUK(data=data)
```




```
UKdata <- data
### usethis::use_data(UKdata, overwrite = TRUE)
```

- Next we extract the simulation configuration using the babsimToolsConf function. As seen below, the configuration is the function can be changed. Later the configuration is stored in para variable and used in simmer.
- Configuration information is stored in the `conf` list, i.e., `conf` refers to the simulation configuration, e.g., sequential or parallel evaluation, number of cores, resource names, log level, etc.

```
conf <- babsimToolsConf()
conf$ResourceEval <- conf$ResourceNames
conf <- getConfFromData(conf = conf,
                        simData = data$simData,
                        fieldData = data$fieldData)
  conf$parallel = parallel
  conf$simRepeats = simrepeats
  conf$ICU = FALSE
  conf$ResourceNames = resourceNames
  conf$ResourceEval = resourceEval
  conf$percCores = percCores
  conf$logLevel = 0
```

```
  conf$w2 = icuWeights
  set.seed(conf$seed)
```

Simulation Model Parameters

- The simulations using simmer (Discrete Event simulation) package to simulate resource utilization in the hospital.
- The core of the `babsim.hospital` simulations is based on the `simmer` package.
- It uses simulation parameters, e.g., arrival times, durations, and transition probabilities.
- These are currently 42 parameters (shown below) that are stored in the list `para`.

```
para <- babsimHospitalPara()
str(para)
%> List of 29
%>  $ AmntDaysInfectedToHospital               : num 9.5
%>  $ AmntDaysNormalToHealthy                  : num 10
%>  $ AmntDaysNormalToIntensive                : num 5
%>  $ AmntDaysNormalToVentilation              : num 3.63
%>  $ AmntDaysNormalToDeath                    : num 5
%>  $ AmntDaysIntensiveToAftercare             : num 7
%>  $ AmntDaysIntensiveToVentilation           : num 4
%>  $ AmntDaysIntensiveToDeath                 : num 5
%>  $ AmntDaysVentilationToIntensiveAfter      : num 30
%>  $ AmntDaysVentilationToDeath               : num 20
%>  $ AmntDaysIntensiveAfterToAftercare        : num 3
%>  $ AmntDaysIntensiveAfterToDeath            : num 4
%>  $ GammaShapeParameter                      : num 1
%>  $ FactorPatientsInfectedToHospital         : num 0.1
%>  $ FactorPatientsHospitalToIntensive        : num 0.09
%>  $ FactorPatientsHospitalToVentilation      : num 0.01
%>  $ FactorPatientsNormalToIntensive          : num 0.1
%>  $ FactorPatientsNormalToVentilation        : num 0.001
%>  $ FactorPatientsNormalToDeath              : num 0.1
%>  $ FactorPatientsIntensiveToVentilation     : num 0.3
%>  $ FactorPatientsIntensiveToDeath           : num 0.1
%>  $ FactorPatientsVentilationToIntensiveAfter: num 0.7
%>  $ FactorPatientsIntensiveAfterToDeath      : num 1e-05
%>  $ AmntDaysAftercareToHealthy               : num 3
%>  $ RiskFactorA                              : num 0.0205
%>  $ RiskFactorB                              : num 0.01
%>  $ RiskMale                                 : num 1.5
%>  $ AmntDaysIntensiveAfterToHealthy          : num 3
%>  $ FactorPatientsIntensiveAfterToHealthy    : num 0.67
```

Run simulation

- The `babsim.hospital` simulator requires the specification of
  1. `arrivalTimes`
  2. configuration list `conf`
  3. parameter list `para` for the simulation.
- In addition to the `arrivalTimes`, a risk can be specified, i.e., a `data.frame` with the following entries can be passed to the main simulation function `babsimHospital`:

    1. `time`: arrival time
    2. `Risk`: risk (based on age and gender)
- The specification of the `Risk` values is optional.
- Output from the simulation is stored in the variable `envs`.

```
%> Windows detected. Turning off parallel processing.
```

Visualize Output

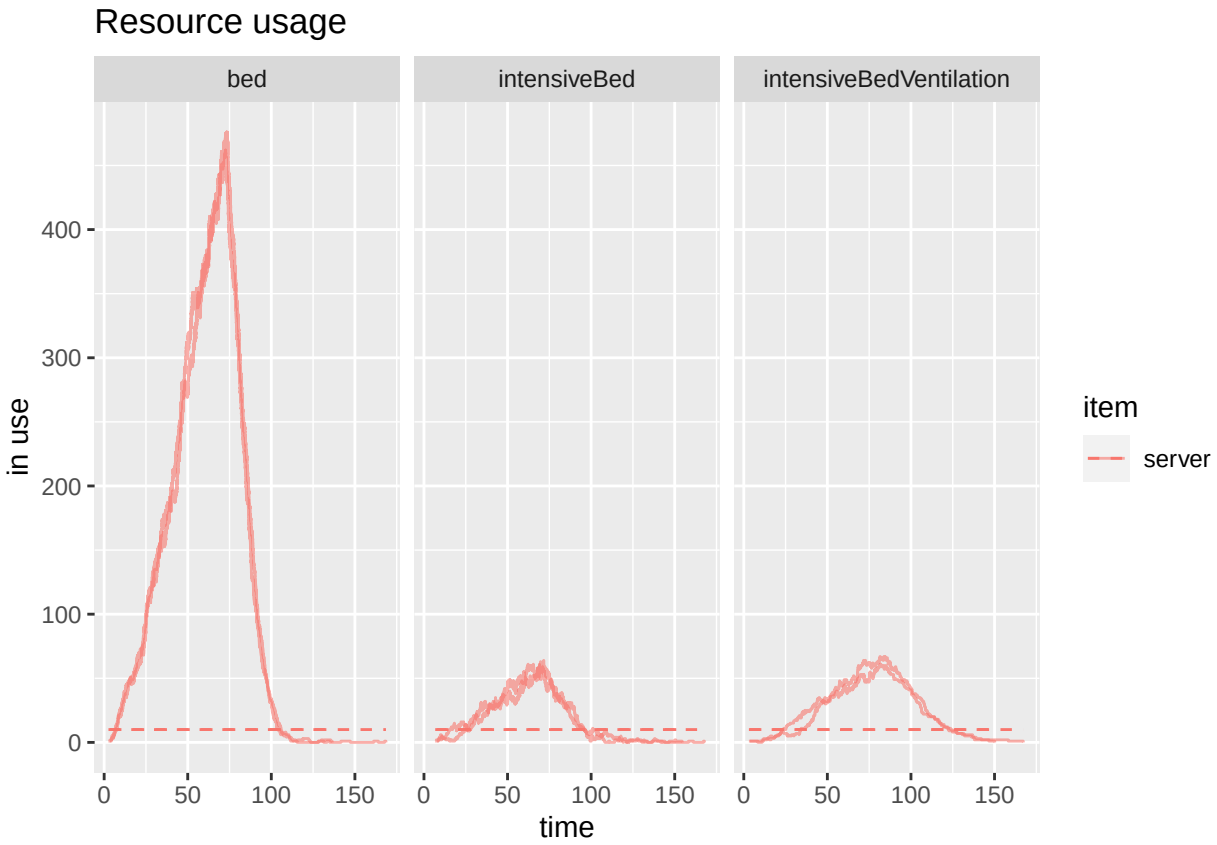
Simmer Plots

- First, we illustrate how to generate plots using the `simmer.plot` package.
- In the following graph, the individual lines are all separate replications. The smoothing performed is a cumulative average.
- Besides `intensiveBed` and `intensiveBedVentilation`, `babsim.hospital` also provides information about the number of non-ICU beds. The non-ICU beds are labeled as `bed`.
- Summarizing, `babsim.hospital` generates output for three bed categories:

1. `bed`
2. `intensiveBed`
3. `intensiveBedVentilation`

- *Resources are conceived with queuing systems in mind, and therefore they comprise two internal self-managed parts: aserver, which is the activepart, with a specified capacity that can be seized and released (seeseize); and a priorityqueueof acertain size, in which arrivals may wait for the server to be available*
- To plot resource usage for three resources side-by-side, we can proceed as follows:

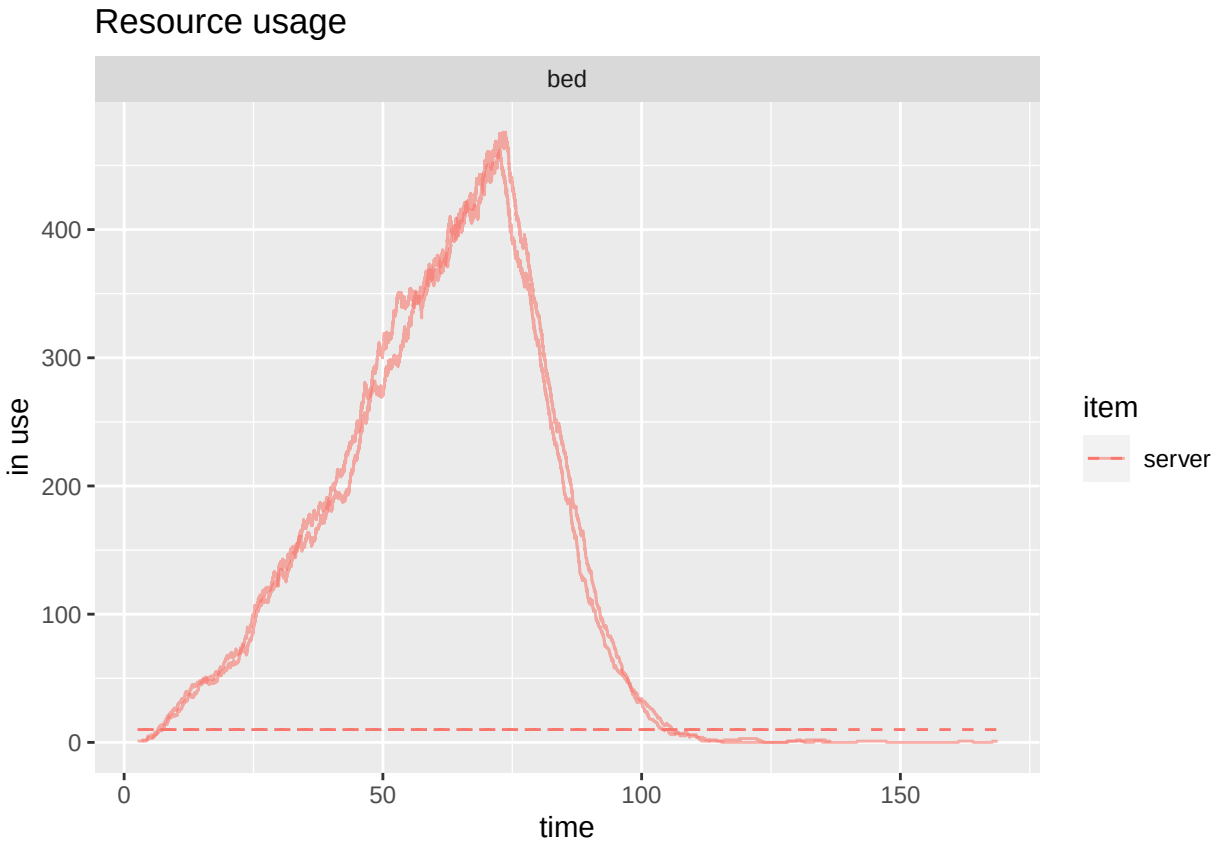
```
resources <- get_mon_resources(envs)
resources$capacity <- resources$capacity/1e5
plot(resources, metric = "usage", c("bed", "intensiveBed", "intensiveBedVentilation"), items = "server"
```



* Each resource can be plotted separately.

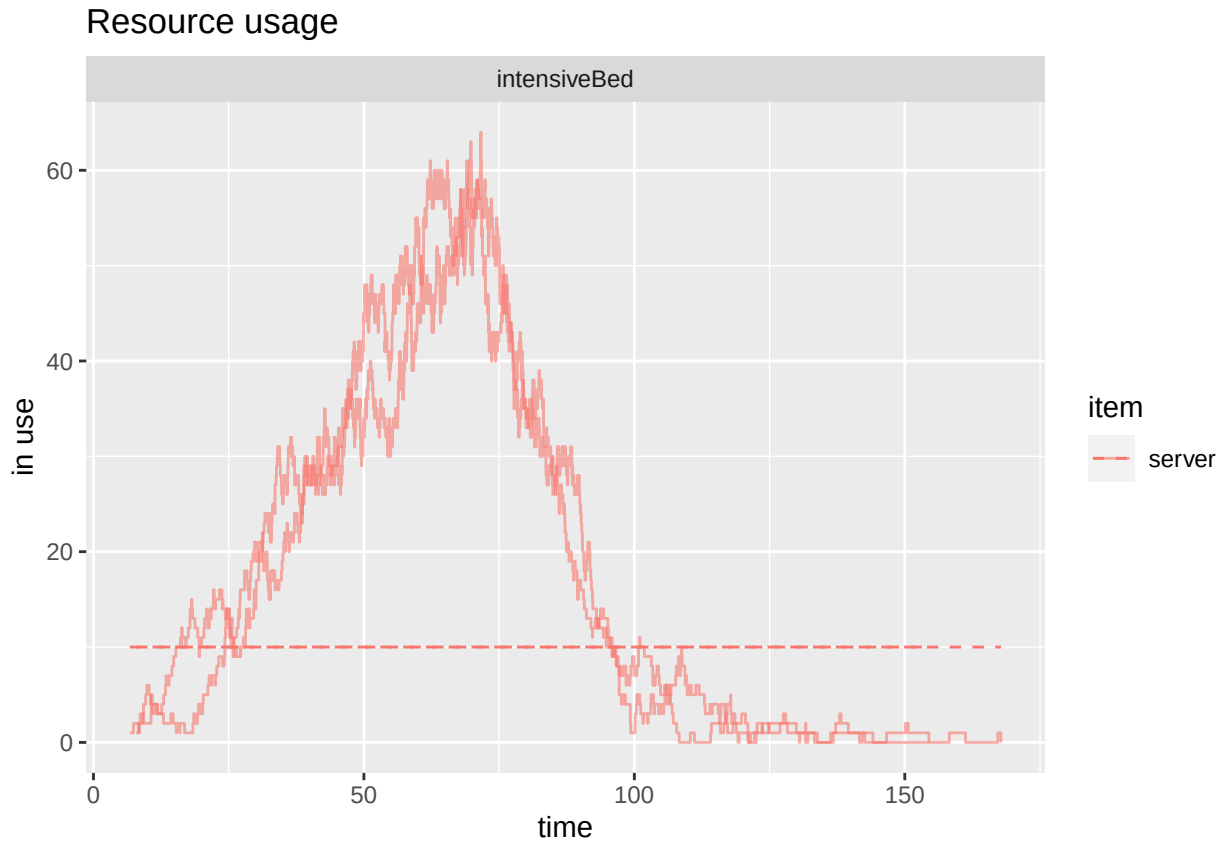
1. The following command generates a plot of non icu beds:

```
plot(resources, metric = "usage", "bed", items = "server", steps = TRUE)
```



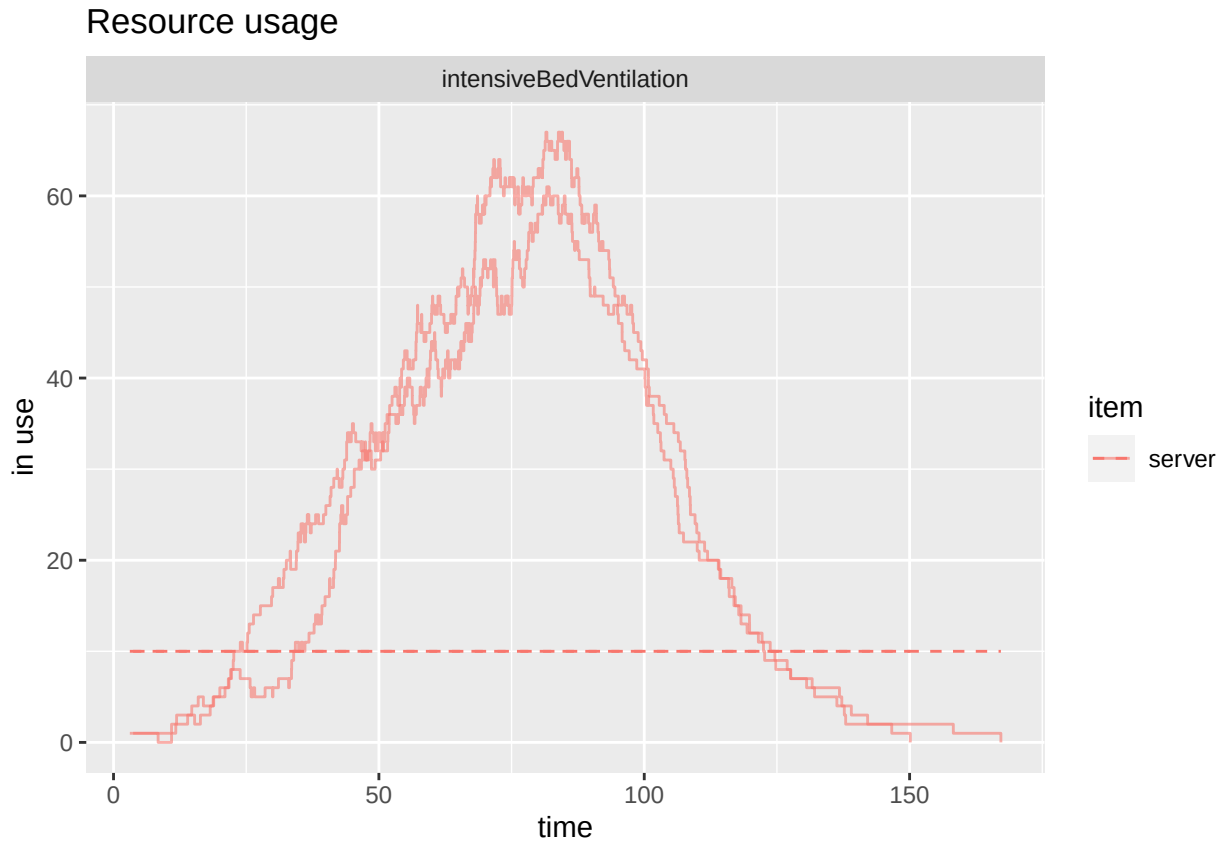
2. The following command generates a plot of icu beds without ventilation:

```
plot(resources, metric = "usage", "intensiveBed", items = "server", steps = TRUE)
```



3. The following command generates a plot of icu beds with ventilation:

```
plot(resources, metric = "usage", "intensiveBedVentilation", items = "server", steps = TRUE)
```



Evaluate Simulation Results

- `babsim.hospital` provides functions for evaluating the quality of the simulation results.
- Simulation results depend on the transition probabilities and durations, i.e., a vector of more than 30 variables.
- These vectors represent *parameter settings*.
- `babsim.hospital` provides a *default* parameter set, that is based on knowledge from domain experts (doctors, members of COVID-19 crises teams, mathematicians, and many more).
- We can calculate the error (RMSE) of the default parameter setting, which was used in this simulation, as follows:

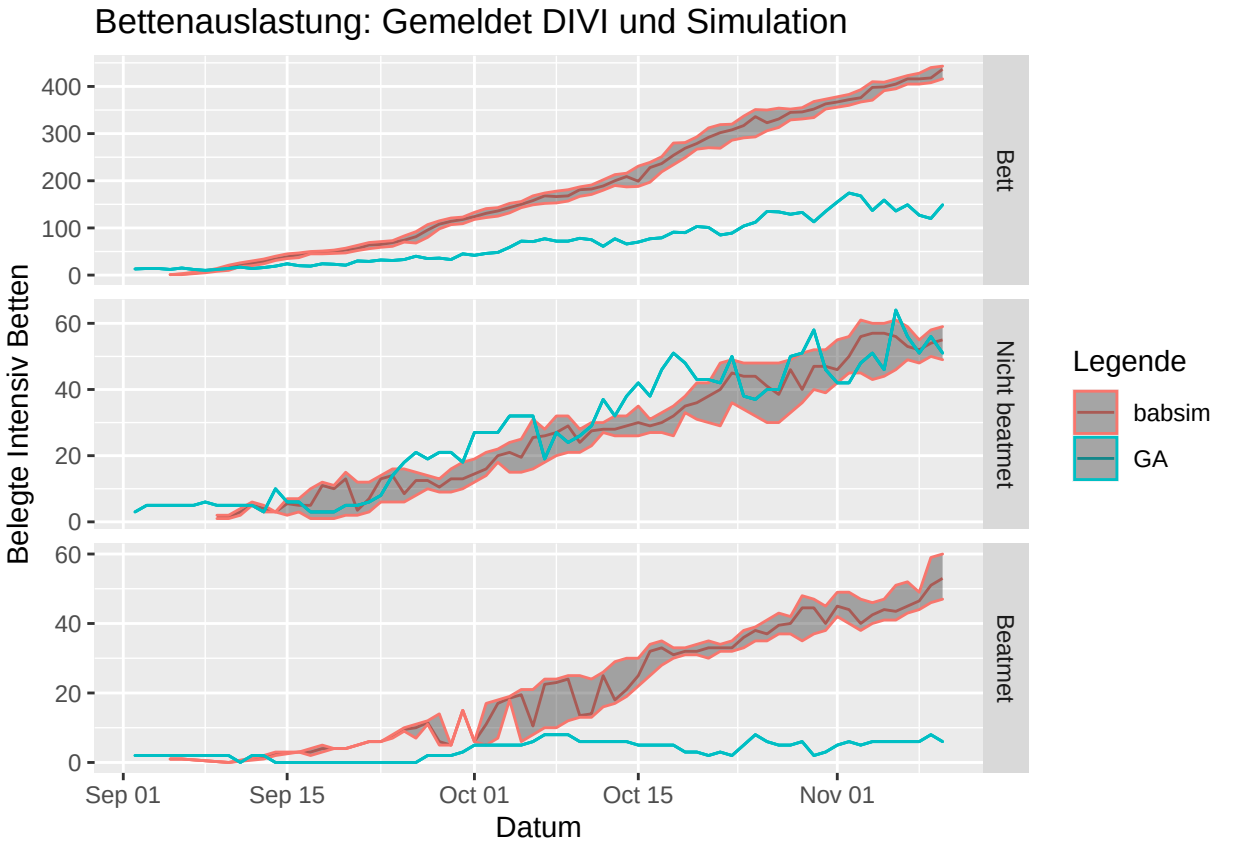
```
fieldEvents <- getRealBeds(data = data$fieldData,
                           resource = conf$ResourceNames)
res <- getDailyMaxResults(envs = envs,  fieldEvents = fieldEvents, conf=conf)
resDefault <- getError(res, conf=conf)
```

The error is 277.0660283.

- Here, we illustrate how `babsim` plots can be generated.

Original Optimisation results

```
p <- plotDailyMaxResults(res, showBeds = TRUE)
plot(p)
```



- Using `ggplot` and `plotly`can be used to generate interactive plots.

```
ggplotly(p)
```

Optimization

- As discussed above, `babsim.hospital` provides a default parameter set, which can be used for simulations.
- The function `babsimHospitalPara()` provides a convenient way to access the default parameter set:

```
para <- babsimHospitalPara()
#print(para)
```

Manual Guess-Estimation paras:

```
para$AmntDaysInfectedToHospital= 5#9.5
para$AmntDaysNormalToHealthy= 7#10
para$AmntDaysNormalToIntensive= 3#5
para$AmntDaysNormalToVentilation= 2#3.63
para$AmntDaysNormalToDeath= 4
para$AmntDaysIntensiveToAftercare= 3#7
para$AmntDaysIntensiveToVentilation= 2#4
para$AmntDaysIntensiveToDeath= 3#5
para$AmntDaysVentilationToIntensiveAfter= 3#6#8#30
para$AmntDaysVentilationToDeath= 4#20
para$AmntDaysIntensiveAfterToAftercare= 3
para$AmntDaysIntensiveAfterToDeath= 4
```

```
para$GammaShapeParameter= 1
para$FactorPatientsInfectedToHospital= 0.06#####0.07#0.1
para$FactorPatientsHospitalToIntensive= 0.25#0.09
para$FactorPatientsHospitalToVentilation= 0.005#0.01
para$FactorPatientsNormalToIntensive= 0.3##0.1
para$FactorPatientsNormalToVentilation= 0.003#0.001
para$FactorPatientsNormalToDeath= 0.1
para$FactorPatientsIntensiveToVentilation= 0.1#0.3
para$FactorPatientsIntensiveToDeath= 0.4#0.1
para$FactorPatientsVentilationToIntensiveAfter= 0.76#0.7
para$FactorPatientsIntensiveAfterToDeath= 1e-05
para$AmntDaysAftercareToHealthy= 3
para$RiskFactorA= 0.02048948
para$RiskFactorB= 0.01
para$RiskMale= 1.5
para$AmntDaysIntensiveAfterToHealthy= 3
para$FactorPatientsIntensiveAfterToHealthy= 0.67
```

- `babsim` provides an interface to optimize the parameter values of the simulation model.
- The following code is just a quick demo.

```
conf$simulationDates$StartDate
conf$simulationDates$EndDate
conf$fieldDates$StartDate
conf$fieldDates$EndDate
```

```
Sys1 <- Sys.time()
print(Sys1)
library("babsim.hospital")
library("SPOT")
library("simmer")
studyDate <- as.Date( min(conf$simulationDates$EndDate, conf$fieldDates$EndDate ))
resUK <- runoptUK(
  expName = paste0("UK-", format(Sys.time(), "%Y-%b.%d-%H.%M-V"), utils::packageVersion("babsim.hospital
  simData = data$simData,
  fieldData = data$fieldData,
  TrainSimStartDate = studyDate - 10*7, # "2020-09-02",
  TrainFieldStartDate = studyDate - 8*7, # "2020-09-15",
  TestSimStartDate = studyDate - 6*7 , #"2020-09-30",
  TestFieldStartDate = studyDate - 4*7, #"2020-10-23",
  Overlap = 0,
  seed = 101170,
  repeats = 2,
  funEvals = 50,
  size = 30,
  simrepeats = 2,
  parallel = TRUE,
  percCores = 0.9,
  icu = FALSE,
  icuWeights = c(1,2,10),
  verbosity = 0,
  resourceNames =  c("bed", "intensiveBed", "intensiveBedVentilation"),
  resourceEval = c("bed", "intensiveBed", "intensiveBedVentilation")
)
```

```
Sys2 <- Sys.time()
print(Sys2-Sys1)
```

- `runoptUK()` returns a list with two elements:
  - `result.df`, which is a data.frame
  - reslist
- `result.df` contains the best (optimized) results from the SPOT runs.
- It can be stored as `ukpara` as follows:
  - WARNING: Only execute the following code if you know what you are doing!

```
#ukpara <- resUK[[1]]
#usethis::use_data(ukpara, overwrite = TRUE)
```

Use Optimized Parameters

- ukpara contains results from several runs, the function `getBestParameter` picks out the best and maps it to the variables that are used by the BaBSim.Hospital simulator.

```
print(ukpara)
```

- Results (parameter settings) of the short `runopt()` optimization from above can be used as follows:

```
para <- getBestParameter(resUK[[1]])
```

- For your convenience, we have stored the results from a quick optimization run.
  - They are available as `ukpara` and can be converted into a `babsimhospital`parameter set as follows:

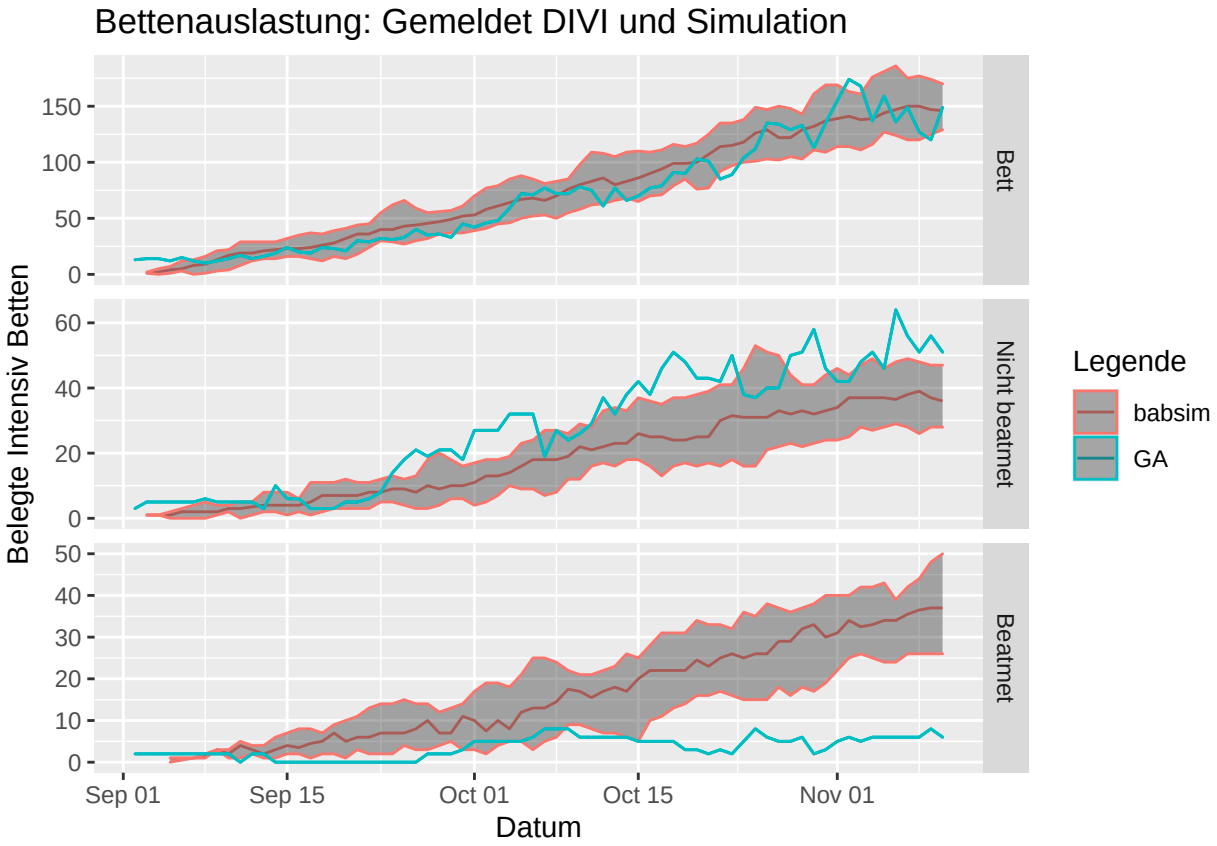
```
para <- getBestParameter(ukpara)
str(para)
%> List of 29
%>  $ AmntDaysInfectedToHospital               : num 6.09
%>  $ AmntDaysNormalToHealthy                  : num 7.1
%>  $ AmntDaysNormalToIntensive                : num 6.38
%>  $ AmntDaysNormalToVentilation              : num 4.89
%>  $ AmntDaysNormalToDeath                    : num 3.21
%>  $ AmntDaysIntensiveToAftercare             : num 8.67
%>  $ AmntDaysIntensiveToVentilation           : num 3.6
%>  $ AmntDaysIntensiveToDeath                 : num 6.69
%>  $ AmntDaysVentilationToIntensiveAfter      : num 27.4
%>  $ AmntDaysVentilationToDeath               : num 19
%>  $ AmntDaysIntensiveAfterToAftercare        : num 2.03
%>  $ AmntDaysIntensiveAfterToDeath            : num 1.3
%>  $ GammaShapeParameter                      : num 0.59
%>  $ FactorPatientsInfectedToHospital         : num 0.0741
%>  $ FactorPatientsHospitalToIntensive        : num 0.106
%>  $ FactorPatientsHospitalToVentilation      : num 0.00625
%>  $ FactorPatientsNormalToIntensive          : num 0.12
%>  $ FactorPatientsNormalToVentilation        : num 0.00179
%>  $ FactorPatientsNormalToDeath              : num 0.0987
%>  $ FactorPatientsIntensiveToVentilation     : num 0.271
%>  $ FactorPatientsIntensiveToDeath           : num 0.107
%>  $ FactorPatientsVentilationToIntensiveAfter: num 0.798
%>  $ FactorPatientsIntensiveAfterToDeath      : num 0.0064
%>  $ AmntDaysAftercareToHealthy               : num 3.48
%>  $ RiskFactorA                              : num 0.484
%>  $ RiskFactorB                              : num 0.0603
```

```
%>  $ RiskMale                                    : num 1.61
%>  $ AmntDaysIntensiveAfterToHealthy             : num 4
%>  $ FactorPatientsIntensiveAfterToHealthy       : num 0.691
```

```
conf$simRepeats = 10
res <- modelResultHospital(para=para,
                           conf=conf,
                           data = data)
%> Windows detected. Turning off parallel processing.
resOpt <- getError(res, conf=conf)
```

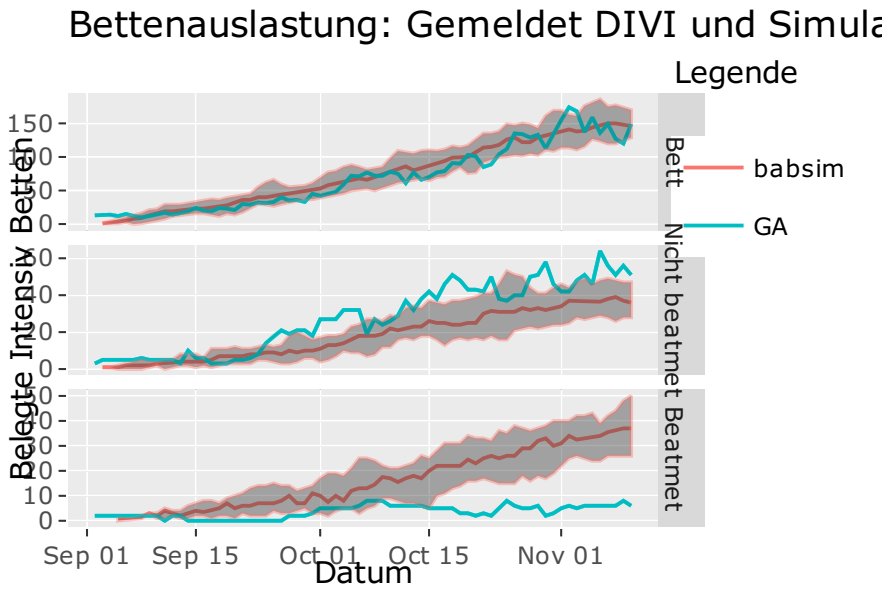
- Optimization improves the error from 277.0660283 to 90.0934252.
- This improvement can also be visualized.

```
p <- plotDailyMaxResults(res, showBeds = TRUE)
print(p)
```



- `ggplot` and `plotly`can be used to generate interactive plots.

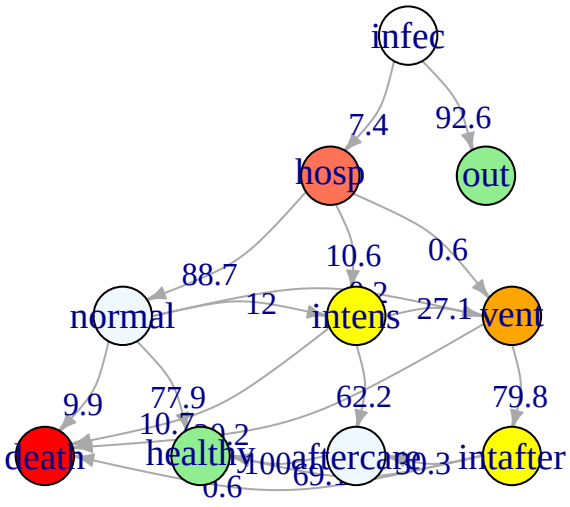
```
ggplotly(p)
```



Visualize and Analyse Parameter Settings

- `babsim.hospital` includes tools to analyse parameter settings.
- You might recall that parameter settings consist of
  1. transition probabilities, e.g., the probability that an infected individual has to go to the hospital.
  2. durations, e.g., the time span until an infected individual goes to the hospital (in days).
- The following plot illustrates the transition probabilities.
- States are as follows:
  1. `infec`: infected
  2. `out`: transfer out, no hospital required
  3. `hosp`: hospital
  4. `normal`: normal station, no ICU
  5. `intens`: ICU (without ventilation)
  6. `vent`: ICU ventilated
  7. `intafter`: intensive aftercare (from ICU with ventilation, on ICU)
  8. `aftercare`: aftercare (from ICU, on normal station)
  9. `death`: patient dies
  10. `healthy`: recovered

```
visualizeGraph(para=para, option = "P")
```

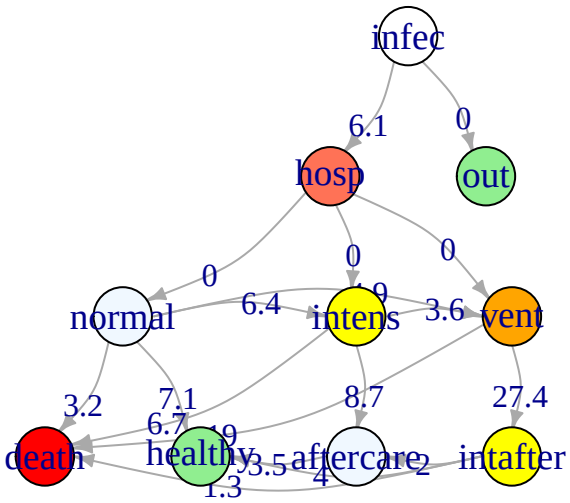


- The transition matrix, that stores the probabilities, is shown below:

```
getMatrixP(para = para )
%>       [,1]      [,2]       [,3]      [,4]      [,5]         [,6]      [,7]
%>  [1,]    0 0.9259253 0.07407474 0.0000000 0.0000000 0.000000000 0.000000
%>  [2,]    0 1.0000000 0.00000000 0.0000000 0.0000000 0.000000000 0.000000
%>  [3,]    0 0.0000000 0.00000000 0.8873109 0.1064377 0.006251458 0.000000
%>  [4,]    0 0.0000000 0.00000000 0.0000000 0.1200981 0.001792896 0.000000
%>  [5,]    0 0.0000000 0.00000000 0.0000000 0.0000000 0.271293589 0.000000
%>  [6,]    0 0.0000000 0.00000000 0.0000000 0.0000000 0.000000000 0.797698
%>  [7,]    0 0.0000000 0.00000000 0.0000000 0.0000000 0.000000000 0.000000
%>  [8,]    0 0.0000000 0.00000000 0.0000000 0.0000000 0.000000000 0.000000
%>  [9,]    0 0.0000000 0.00000000 0.0000000 0.0000000 0.000000000 0.000000
%> [10,]    0 0.0000000 0.00000000 0.0000000 0.0000000 0.000000000 0.000000
%>            [,8]        [,9]     [,10]
%>  [1,] 0.0000000 0.000000000 0.0000000
%>  [2,] 0.0000000 0.000000000 0.0000000
%>  [3,] 0.0000000 0.000000000 0.0000000
%>  [4,] 0.0000000 0.098721699 0.7793873
%>  [5,] 0.6216025 0.107103955 0.0000000
%>  [6,] 0.0000000 0.202301964 0.0000000
%>  [7,] 0.3028948 0.006398169 0.6907070
%>  [8,] 0.0000000 0.000000000 1.0000000
%>  [9,] 0.0000000 1.000000000 0.0000000
%> [10,] 0.0000000 0.000000000 1.0000000
```

- Similar to the probabilites, durations can be visualized:

```
visualizeGraph(para = para, option = "D")
```



- The corresponding matrix is shown below:

```
getMatrixD(para = para)
%>       [,1] [,2]     [,3] [,4]     [,5]     [,6]     [,7]     [,8]       [,9]
%>  [1,]    0    0 6.090185    0 0.000000 0.000000  0.00000 0.000000  0.000000
%>  [2,]    0    0 0.000000    0 0.000000 0.000000  0.00000 0.000000  0.000000
%>  [3,]    0    0 0.000000    0 0.000000 0.000000  0.00000 0.000000  0.000000
%>  [4,]    0    0 0.000000    0 6.384118 4.885070  0.00000 0.000000  3.208737
%>  [5,]    0    0 0.000000    0 0.000000 3.604919  0.00000 8.669784  6.686112
%>  [6,]    0    0 0.000000    0 0.000000 0.000000 27.41509 0.000000 18.984470
%>  [7,]    0    0 0.000000    0 0.000000 0.000000  0.00000 2.027775  1.300164
%>  [8,]    0    0 0.000000    0 0.000000 0.000000  0.00000 0.000000  0.000000
%>  [9,]    0    0 0.000000    0 0.000000 0.000000  0.00000 0.000000  0.000000
%> [10,]    0    0 0.000000    0 0.000000 0.000000  0.00000 0.000000  0.000000
%>         [,10]
%>  [1,] 0.00000
%>  [2,] 0.00000
%>  [3,] 0.00000
%>  [4,] 7.10197
%>  [5,] 0.00000
%>  [6,] 0.00000
%>  [7,] 3.99723
%>  [8,] 3.48278
%>  [9,] 0.00000
```

```
%> [10,] 0.00000
```

Project Work

Part I (26.1.2021)

- Install the required software on your computer.
- Execute this RMD file:
  - Modify the path the the Excel file (according to your group)
  - Run the optimization (see Section "Optimzation").
  - WARNING: the runs might take some time.
- Describe the parameter set `babsim.hospital::babsimHospitalPara` that is used by the `babsimHospital` simulator.

  The predictive quality of the simulation is based on a set of 29 parameters. The default values of these parameters were established in cooperation with medical professionals. Our aim to investigate and optimize these parameters to improve babsimhospital. So we use model-based optimization via SPOT and an in-depth sensitivity analysis. Sensitivity analysis is so important for the optimization process, since it allows to focus the search on the most important parameters of the simulation. This approach allows for a reduction in the dimensions of the simulation without compromising the resulting accuracy. Besides using the optimized parameters for simulations, which allows improved simulations, the optimized parameters can be used for analysing the parameter settings. Babsim.hospital includes several tools to analyze parameter settings. You might recall that parameter settings consist of • transition probabilities, e.g., the probability that an infected individual has to go to the hospital. • durations, e.g., the time span until an infected individual goes to the hospital (in days).
  - You can use the `help` function, e.g. 'help(babsim.hospital::babsimHospitalPara).
- Take a look a the source code of the `babsim.hospital::babsimHospital` function:
  - Describe states and model trajectories. The states are defined by the patients location within the Covid-19 model. These describe the patients movement from arrival and use of the hospital resources (type of bed), to the final departure, either transferred out - healthy or in the unfortunate case, death. The trajectories define when a patient will move from one state to another. This uses the above 29 parameters which describes the duration and the patient probability percentage that will transition via different branches to a new state. Each of these states is modeled within the babsim::hospital function and has a corresponding simmer environment created. The duration parameters define each of the simmer package time-out trajectories. The Probabilities are used in the branch call within each simmer state. This directs the patient to a new state, according to the trajectory percentage defined. See debug BabsimHospital code.

  States are as follows:

  1. `infec`: infected
  2. `out`: transfer out, no hospital required
  3. `hosp`: hospital
  4. `normal`: normal station, no ICU
  5. `intens`: ICU (without ventilation)
  6. `vent`: ICU ventilated
  7. `intafter`: intensive aftercare (from ICU with ventilation, on ICU)
  8. `aftercare`: aftercare (from ICU, on normal station)
  9. `death`: patient dies
  10. `healthy`: recovered

  - You can use the `help` function, e.g. 'help(babsim.hospital::babsimHospital).

*Discussion
1. babsimHospitalPara The Parameter set for Babsim.Hospital simulator takes 29 individual parameters that is used to model a Covid-19 patient infection path. This uses discrete event based simulation, whereby each state has various trajectory paths with two factors being a probability (%) and a duration (days) of which a patient may transition between the discrete states. These factors were initially theorized by leading industry specialist, (Doctors, mathematicians etc.) and combined with Covid-19 infection data from the Robert Koch Institute, presenting daily hospital resource usage. These parameters define how the simulator models Covid-19 trajectories and the ability to predict future Hospital resource usage, given a base infection rate of the virus. These parameters are easily viewable from the babsim-website-advanced functionality-Model, which illustrates the state model for both duration and percent.

Part II (2.2.2021)

- Install the required software on your computer.
- Execute the code below (this is a modified version of the code from last week):
  - Modify the path of the Excel file (according to your group)
  - Run the optimization (see Section "Optimization").
  - WARNING: the runs might take some time.
- Describe the commands as well as input and output for every code fragment in the RMD code below.
- Generate a flowchart (flow diagram) that describes the simulation and optimization process based on the code below.
- Describe the role of the random number generator:
  - What is the meaning of the command `set.seed`? Refer to code section: Arrivaltimes.
  - What happens if you use RNGkind("Wich")? The `Wichmann-Hill` random number generator. This creates a uniform distribution of random numbers between 0 and 1. It operates by using three linear congruential generators with the prime moduli being 30269, 30307 and 30323. This provides a non-repeatable cycle of $6.95 * 10^{12}$ as stated in the the corrected version publicized by Wichmann and Hill, two years after the first release.
- Optimization does not improve every bed category.
  - Do you have an explanation for this? Currently, using the default German parameters allows for more patients to be transferred to the IVB category. Unfortunately, the UK does not have as many beds in the IVB category, thus the model boundaries need to be adjusted. This was initially estimated by manually tuning the model trajectories in section (Manual Guess-Estimation paras) to see the impact of certain parameters. The guessed trajectory parameters do fit the data, however; these are very much incorrect and must not be used as boundaries may well be out of the original specifications. In the last part of Section 2, a code section: debug runopUK, illustrates boundaries have been adjusted manually for para[16, 18, 20]. Due to Computer processing restraints, the simulator has only been run once. Our overall best error from these initial adjustments resulted in rmse ~100. This resulted with a better fit of the normal beds and Intensive Bed category. However; the intensive bed ventilation is still not correctly fitting, thus require more tuning. An issue also arises here regarding the second wave and the actual number of deaths associated. As this information is not available with the raw data, it is difficult to adjust the trajectories to the death state without any corresponding evidence. Prof. Dr. Bartz highlighted the bed differences during the presentation and the difference between Germany and UK bed numbers is a factor 5 (i.e: Germay 5x times as many beds). This provides a better insight as to how the boundaries for some parameters are adjusted.
  - Consider a different situation in UK hospital? A different situation could be the above regarding the number of deaths. Assuming the second wave is more deadly, the states having trajectories linked to the death state are: Normal, Intensive, Ventilation and Intensive Aftercare. For an overall accurate model, we need data on this `death` category to adequately adjust our model trajectories.
  - A second situation could be the Hospitals are overloaded and we employ queues. This would have grave consequences in our model as patients are deemed to be `waiting` for a bed, be it either normal, intensive or intensive Ventilated. This would most likely result in many more casualties from this Covid-19 virus.

- Do you have any ideas how this can be fixed?
- Regarding the 'Death' trajectory path, we could confirm the data with the UK's Covid reporting authorities so that we could better understand form which states patients are succumbing to this virus and remodel these trajectory paths.
- Regarding the Hospital overloading, this is a very dire situation and one that must be avoided at all costs. As such, the Governments enforcement of Schools and business closures to reduce the spread of the virus is paramount. Thus we would suggest to have as much data as possible to re-tune our DES model so we can avoid this situtation.

Code for Part II (Simulation and Optimization of babsimhospital)

Technical setup

- On this part libraries are loading which we need it for simulation and optimization into the R environment -Code as comments show : installing devtools, spot and babsim.hospital if it is not installed before -"rm(list = ls())" with this command variable which are still in the environment for example : older sessions are deleted. -"suppressPackageStartupMessages" with this command messages which comes during loading packages will not be showed. -"packageVersion("SPOT")" shows that package is installed and the current version.

```
## install.packages("devtools")
## devtools::install_github("r-lib/devtools")

## url <- "http://owos.gm.fh-koeln.de:8055/bartz/spot.git"
## devtools::install_git(url = url)

## url <- "http://owos.gm.fh-koeln.de:8055/bartz/babsim.hospital"
## devtools::install_git(url = url, subdir = "babsim.hospital")

rm(list = ls())
suppressPackageStartupMessages({
library("SPOT")
library("babsim.hospital")
library("simmer")
library("simmer.plot")
library("readxl")
library("plotly")
  })

packageVersion("SPOT")
```

- Read XLSX Data 1)"Read_excel" command helps to read data from whole excel file and save as X20201111UKdata.

2) On the next line we only read date which starts from 2020-09-01.

```
X20201111UKdata <- read_excel("/home/opti/Documents/IDEA-Master-AIT-WS2020-2021/Numerical-Methods/Datas
ukdataRaw <- X20201111UKdata[as.Date(X20201111UKdata$Date) > "2020-09-01",]
# remove outlier (on last day)
ukdataRaw <- ukdataRaw[as.Date(ukdataRaw$Date) < "2020-11-10",]
```

- Generate simulation data, input to the simulator

1) Creating new frame as simulation data with 2 column "Day" and "Infected" which takes this values from UkdataRaw.

```
simData <- data.frame(Day = as.Date(ukdataRaw$Date),
                      Infected = ukdataRaw$NewCases)
```

- Generate field data (real-world data)
- Here creating fieldData table as column : "Day" , "Bed" (which gets subtraction of "Total-COVID19Inpatient", "COVID19NonInvasiveCPAP" , "COVID19VentilatedICU") , "intensiveBed", "intensiveBedVentilation"
- "rownames(fieldData) <- NULL" shows there is no row names

```
Day <- as.Date(ukdataRaw$Date)
bed <- ukdataRaw$TotalCOVID19Inpatient - ukdataRaw$COVID19NonInvasiveCPAP - ukdataRaw$COVID19VentilatedI
intensiveBed <-  ukdataRaw$COVID19NonInvasiveCPAP
intensiveBedVentilation <- ukdataRaw$COVID19VentilatedICU
fieldData <- data.frame(Day = Day,
                        bed = bed,
                        intensiveBed = intensiveBed,
                        intensiveBedVentilation = intensiveBedVentilation)
rownames(fieldData) <- NULL
```

- Combine data:

1. Create a new list of 2, combining; i=(the dates and infected) and ii= (dates and hospital resource usage) (i.e: beds)

```
data <- list(simData = simData,
 fieldData = fieldData)
```

- Configuration setup and seed setup 1.For configuration calling "BabsimToolsconf" block and using its structure and modify for parameters for this configuration.

2. This calls the function within library Package:babsim.hospital, and sets the following:

a) Get initial default configuration. (List of 14)
b) ResourceEval is changed to include (beds) plus original IB and IBV.
c) Call function library Package:babsim.hospital, getConfFromData() which takes previous dataframes simdata and fielddata. The Simulator start and end dates are set from the field data if it is not null. The return is conf.
d) The following parameters are 'User Selectable', which can overwrite the default parameters in the Package:babsim.hospital.
e) The Random Number Generator (RNG) - A integer vector which is set in order to obtain a repetitive number sequence.

```
conf <- babsimToolsConf()
conf$ResourceEval <- conf$ResourceNames
conf <- getConfFromData(conf = conf,
                        simData = data$simData,
                        fieldData = data$fieldData)
conf$parallel = TRUE
conf$simRepeats = 2
conf$ICU = FALSE
conf$ResourceNames =  c("bed", "intensiveBed", "intensiveBedVentilation")
conf$ResourceEval =  c("bed", "intensiveBed", "intensiveBedVentilation")
conf$percCores = 0.8
conf$logLevel = 0
conf$w2 = c(1,2,10)
```

```
conf$seed = 123
set.seed(conf$seed)
```

- Set simulation parameters:

1. Using the library Package:babsim.hospital, the function babsimHospitalPara is a pre-configured default list of 29 parameters which can be used for first initialization. These default trajectories contain duration and probability data.

```
para <- babsimHospitalPara()
```

- Generate arrival times

.....getArrivalTimes..... ""“RNGkind”"
changes the kind of the random uniform or normal generator and sets the seed to a time-based value. Random.seed is an integer vector, containing the random number generator (RNG) state for random number generation in R. It can be saved and restored, but should not be altered by the user.

""“set.seed()”" uses a single integer argument to set as many seeds as are required. It is intended as a simple way to get quite different seeds by specifying small integer arguments, and also as a way to get valid seed sets for the more complicated methods

""“on.exit”"

Records the expression given as its argument as needing to be executed when the current function exits (either naturally or as the result of an error). This is useful for resetting graphical parameters or performing other cleanup actions. If no expression is provided, i.e., the call is on.exit(), then the current on.exit code is removed.

The number of Infected people for each day(between 2020-09-02 and 2020-11-10, ~ 70days) are using in “getArrivaltimes block”.

1. library Package:babsim.hospital, the function get arrival times uses the random seed “Wichmann-Hill”. It takes in the number of Daily Infection cases.
2. seq_along - returns integer vector 1 to 70.
3. Create an array, 0 to 69, and then repeat each day by the amount of infections on that day.
4. For each day, the runif function provides a uniform distribution from 0 to 1, using n = number of total cases over the selection period. This is summed with each day, providing a patient distribution 0 to 69 with ~5 decimal places.
5. A new data frame is created to store the arrival times, length = sum(number of patents per day) over trial period = 27679.

Summary: The arrival time is a measure of the time of the day the patient arrived as fraction or percentage. Therefore, it is very clear why it started from zero since any time within current day is less than 100% of the total time of the day. Similarly, 110% of today is clearly the next day, hence days that starts with 1. represent the next day within the time in focus.

```
arrivalTimes <- data.frame(time= getArrivalTimes(data$simData$Infected))

# debug getArrivalTimes
#arrivalTimes <- data.frame(time= debuggetArrivalTimes(data$simData$Infected))
```

debug getArrivalTimes

```
debuggetArrivalTimes <- function (xDaily)
{
  browser()
  orng <- RNGkind()
```

```
  on.exit(RNGkind(orng[1], orng[2], orng[3]))
  RNGkind("Wichmann-Hill")
  totalCases <- sum(xDaily)
  # generate patient arrivals for particular day - (Arrival time is a ratio of the day)
  arrivalTimes <- rep(seq_along(xDaily) - 1, xDaily) + runif(totalCases,
    0, 1)
  data.frame(time = sort(arrivalTimes))
}
```

- Run simulation

Calling Simmer environment and Simulate resource:

The simmer is used here which will help in initialising the babsim hospital to envs. As stated in above the arrival time is initialized and thus the parameters are configured.

The parameters, configuration and arrival times are parsed to the babsimHospital simulator. 1) Initial configuration settings extracted. 2) Create P matrix and additional list creation. This is created from only the Patient Factors X14-X29.(No days) 3) Create functions for the duration parameters. Use RNG, runif and qgamma to return the time out variables for latter use in simmer trajectory function. 4) Create simmer function. Parse time out trajectories, allocate and seize resources. 5) Configuration machine processing abilities (Windows-Linux-Mac) 6) Return a list of 2 (2x Simulations) containing the simmer package variables - Monitoring arrivals, resources, attributes, queues, counters etc.

```
envs <- babsimHospital(arrivalTimes = arrivalTimes, conf = conf, para = para)
# debug babsimHospital:
#envs <- debugbabsimHospital(arrivalTimes = arrivalTimes, conf = conf, para = para)
```

debug BaBSimHospital

```
debugbabsimHospital <- function (arrivalTimes = NULL, conf = list(), para = list(),
  ...)
{
  library(parallel)
  browser()
  RNGkind("Wich")
  # Get 'log level' for simmer package debugging.
  # Message Printed with simulation time if level provided <= log_level. See Simmer.
  conf$logLevel <- min(1, conf$logLevel)
  # Get user configurated simulation repeats
  simRepeats <- conf$simRepeats
  # Get normal bed Capacity
  Amnt_Normal_Beds <- conf$maxCapacity
  # Get ICB bed Capacity
  Amnt_Intensive_Care_Beds <- conf$maxCapacity
  # Get ICBV bed Capacity
  Amnt_Intensive_Care_Beds_Ventilation <- conf$maxCapacity
  # Get the gamma value
  GammaShapeParameter <- para$GammaShapeParameter
  # Add new column to arrival times called 'risk'. Set all equal to 1.
  if (!"risk" %in% colnames(arrivalTimes)) {
    arrivalTimes$risk <- rep(1, length(arrivalTimes$time))
  }
  # Create the Probabilities matrix using only the patient factor parameters X14-X29.
  P <- getMatrixP(para = para)
```

```r
# Function to transform P including risk factors. Not used!
calculateAllPMatrices <- function() {
  browser()
  possibleRisks <- unique(arrivalTimes$risk)
  getSingleMatrix <- function(singleRisk) {
    updateMatrixP(P = P, u = list(k = singleRisk))
  }
  possibleMatrices <- lapply(as.list(possibleRisks), getSingleMatrix)
  names(possibleMatrices) <- round(possibleRisks, 5)
  return(possibleMatrices)
}
Ps <- calculateAllPMatrices()
# The following is the duration parameter function calculations.
# rtgamma - produces a random number between 0-1, parse result to qgamma to provide the quantile func

DurationInfected2Hospital <- function() rtgamma(n = 1, shape = GammaShapeParameter,
  rate = 1/para$AmntDaysInfectedToHospital, shift = 1,
  alpha = 0.95)
DurationNormal2Healthy <- function() rtgamma(n = 1, shape = GammaShapeParameter,
  rate = 1/para$AmntDaysNormalToHealthy, alpha = 0.95)
DurationNormal2Intensive <- function() rtgamma(n = 1, shape = GammaShapeParameter,
  rate = 1/para$AmntDaysNormalToIntensive, alpha = 0.95)
DurationNormal2Ventilation <- function() rtgamma(n = 1,
  shape = GammaShapeParameter, rate = 1/para$AmntDaysNormalToVentilation,
  alpha = 0.95)
DurationNormal2Death <- function() rtgamma(n = 1, shape = GammaShapeParameter,
  rate = 1/para$AmntDaysNormalToDeath, alpha = 0.95)
DurationIntensive2Aftercare <- function() rtgamma(n = 1,
  shape = GammaShapeParameter, rate = 1/para$AmntDaysIntensiveToAftercare,
  alpha = 0.95)
DurationIntensive2Ventilation <- function() rtgamma(n = 1,
  shape = GammaShapeParameter, rate = 1/para$AmntDaysIntensiveToVentilation,
  alpha = 0.95)
DurationIntensive2Death <- function() rtgamma(n = 1, shape = GammaShapeParameter,
  rate = 1/para$AmntDaysIntensiveToDeath, alpha = 0.95)
DurationVentilation2IntensiveAfter <- function() rtgamma(n = 1,
  shape = GammaShapeParameter, rate = 1/para$AmntDaysVentilationToIntensiveAfter,
  alpha = 0.95)
DurationVentilation2Death <- function() rtgamma(n = 1, shape = GammaShapeParameter,
  rate = 1/para$AmntDaysVentilationToDeath, alpha = 0.95)
DurationIntensiveAfter2Aftercare <- function() rtgamma(n = 1,
  shape = GammaShapeParameter, rate = 1/para$AmntDaysIntensiveAfterToAftercare,
  alpha = 0.95)
DurationIntensiveAfter2Death <- function() rtgamma(n = 1,
  shape = GammaShapeParameter, rate = 1/para$AmntDaysIntensiveAfterToDeath,
  alpha = 0.95)
DurationAftercare2Healthy <- function() rtgamma(n = 1, shape = GammaShapeParameter,
  rate = 1/para$AmntDaysAftercareToHealthy, alpha = 0.95)
DurationIntensiveAfter2Healthy <- function() rtgamma(n = 1,
  shape = GammaShapeParameter, rate = 1/para$AmntDaysIntensiveAfterToHealthy,
  alpha = 0.95)
simFun <- function(i) {
  # initialize simmer environment == Empty!
```

```
env <- simmer("Simulation", log_level = conf$logLevel)
browser()
# Model the states and the trajectory paths:
# Get transferout trajectory
transferout <- trajectory("No Hospital") %>% log_("transferout",
  level = 1) %>% set_global("No Hospital Required",
  1, mod = "+")
# Get healthy trajectory
healthy <- trajectory("healthy") %>% log_("healthy",
  level = 1) %>% set_global("Healed", 1, mod = "+")
# Get death trajectory
death <- trajectory("death") %>% log_("death", level = 1) %>%
  set_global("Dead", 1, mod = "+")
# Get aftercare trajectory
aftercare <- trajectory("aftercare") %>% log_("aftercare",
  level = 1) %>% seize("bed", 1) %>% join(trajectory() %>%
  timeout(DurationAftercare2Healthy) %>% release_all("bed"),
  healthy)
# Get intensiveAfter trajectory
# Warning: A large function that pipes branch trajectories!
intensiveAfter <- join(trajectory("intensiveAfter") %>%
  log_("intensiveAfter", level = 1) %>% seize("intensiveBed",
  1) %>% branch(function() {
  r <- get_attribute(env, "risk")
  R <- Ps[[as.character(round(r, 5))]]
  p <- c(R[7, 8], R[7, 9], R[7, 10])
  getDecision(p)
}, continue = FALSE, join(trajectory() %>% timeout(DurationIntensiveAfter2Death) %>%
  release_all("intensiveBed"), death), join(trajectory() %>%
  timeout(DurationIntensiveAfter2Healthy) %>% release_all("intensiveBed"),
  healthy)) %>% timeout(DurationIntensiveAfter2Aftercare) %>%
  release_all("intensiveBed"), aftercare)
# Get ventilation trajectory
ventilation <- join(trajectory("Intensive Care Ventilation") %>%
  log_("ventilation", level = 1) %>% seize("intensiveBedVentilation",
  1) %>% branch(function() {
  r <- get_attribute(env, "risk")
  R <- Ps[[as.character(round(r, 5))]]
  p <- c(R[6, 9], R[6, 7])
  getDecision(p)
}, continue = FALSE, join(trajectory() %>% timeout(DurationVentilation2IntensiveAfter) %>%
  release_all("intensiveBedVentilation"), intensiveAfter)) %>%
  timeout(DurationVentilation2Death) %>% release_all("intensiveBedVentilation"),
  death)
# Get intensive trajectory
intensive <- join(trajectory("Intensive Care") %>% log_("intensive",
  level = 1) %>% seize("intensiveBed", 1) %>% branch(function() {
  r <- get_attribute(env, "risk")
  R <- Ps[[as.character(round(r, 5))]]
  p <- c(R[5, 8], R[5, 6], R[5, 9])
  getDecision(p)
}, continue = FALSE, join(trajectory() %>% timeout(DurationIntensive2Ventilation) %>%
  release_all("intensiveBed"), ventilation), join(trajectory() %>%
```

```
    timeout(DurationIntensive2Death) %>% release_all("intensiveBed"),
    death)) %>% timeout(DurationIntensive2Aftercare) %>%
    release_all("intensiveBed"), aftercare)
  # Get normalStation trajectory
  normalStation <- trajectory("Normal Station") %>% log_("normalStation",
    level = 1) %>% seize("bed") %>% branch(function() {
    r <- get_attribute(env, "risk")
    R <- Ps[[as.character(round(r, 5))]]
    p <- c(R[4, 10], R[4, 5], R[4, 9], R[4, 6])
    getDecision(p)
  }, continue = FALSE, join(trajectory() %>% timeout(DurationNormal2Intensive) %>%
    release_all("bed"), intensive), join(trajectory() %>%
    timeout(DurationNormal2Death) %>% release_all("bed"),
    death), join(trajectory() %>% timeout(DurationNormal2Ventilation) %>%
    release_all("bed"), ventilation)) %>% join(trajectory() %>%
    timeout(DurationNormal2Healthy) %>% release_all("bed"),
    healthy)
  # Get hospital trajectory
  hospital <- join(trajectory("hospital") %>% log_("hospital",
    level = 1) %>% branch(function() {
    r <- get_attribute(env, "risk")
    R <- Ps[[as.character(round(r, 5))]]
    p <- c(R[3, 4], R[3, 5], R[3, 6])
    getDecision(p)
  }, continue = FALSE, intensive, ventilation), normalStation)
  # Get infected trajectory
  infected <- join(trajectory("New Infected") %>% log_("infected",
    level = 1) %>% timeout(DurationInfected2Hospital) %>%
    branch(function() {
      r <- get_attribute(env, "risk")
      R <- Ps[[as.character(round(r, 5))]]
      p <- c(R[1, 2], R[1, 3])
      getDecision(p)
    }, continue = FALSE, hospital), transferout)
  # dataframe manipulation
  add_dataframe(env, name_prefix = "patient", trajectory = infected,
    data = arrivalTimes, mon = 1, col_time = "time",
    time = "absolute", col_attributes = c("risk"))
  # Combine all trajectories into the simmer environment package 'env' and wrap().
  env %>% add_resource("bed", Amnt_Normal_Beds) %>% add_resource("intensiveBed",
    Amnt_Intensive_Care_Beds) %>% add_resource("intensiveBedVentilation",
    Amnt_Intensive_Care_Beds_Ventilation) %>% run() %>%
    wrap()
}
# PC type detection and set of parallel
switch(Sys.info()[["sysname"]], Windows = {
  messagef("Windows detected. Turning off parallel processing.")
  conf$parallel <- FALSE
}, Linux = {
  if (conf$verbosity > 1000) {
    messagef("Linux detected. Parallel processing possible.")
  }
}, Darwin = {
```

```
    if (conf$verbosity > 1000) {
      messagef("Mac detected. Parallel processing possible.")
    }
  })
  if (conf$verbosity > 1000) {
    messagef("simFun() uses the following %s arrival times:",
      length(arrivalTimes$time))
    print(arrivalTimes$time)
  }
  if (conf$parallel) {
    nCores <- detectCores(logical = FALSE)
    mc.coresN = min(nCores - 1, round(conf$percCores * nCores))
    if (conf$verbosity > 100) {
      messagef("BEGIN: babsimHospital() calling parallel simFun() with %s cores: ######################
        mc.coresN)
    }
    # Commeted out due to no debug mode in parallel - using standard loop call
    #envs <- mclapply(1:simRepeats, simFun, mc.cores = getOption("mc.cores",
    #  mc.coresN))
    envs <- lapply(1:simRepeats, simFun)
  }
  else {
    if (conf$verbosity > 100) {
      messagef("BEGIN: babsimHospital() calling sequential simFun(): ############################")
    }
    envs <- lapply(1:simRepeats, simFun)
  }
  if (conf$verbosity > 100) {
    printConf(conf)
    messagef("END: babsimHospital(): simFun: ############################")
  }
  return(envs)
}
```

*Get resources

1) Getter methods from the simmer package to get resources. - Takes a list of environments, returns dataframe.

a) Resource list contains resource bed allocations, arrival time, server, queue (not assigned), capacity (1e+06).

2) Set resource capacity to 10 (1e+06 / 1e+05) Why? Allows the plot to be re-scaled.

```
resources <- get_mon_resources(envs)
resources$capacity <- resources$capacity /1e5
```

*Plot resource usage Resources are them configured according names of the resources available

1) Takes the first two columns of 'resources' (Bed resource and time) and plots 3x distinct graphs. The 3x graphs are a result of the resource column containing these parameters.
2) The plot additional parameters are for title and legend display.

```
plot(resources, metric = "usage", c("bed", "intensiveBed", "intensiveBedVentilation"), items = "server"
```

```
# plot(resources, metric = "usage", "bed", items = "server", steps = TRUE)
# plot(resources, metric = "usage", "intensiveBed", items = "server", steps = TRUE)
```

```
# plot(resources, metric = "usage", "intensiveBedVentilation", items = "server", steps = TRUE)
```

- Generate data frame from simulation results

"GetRealbeds" function using "fieldData" and "ResourceNames"in order to seperate of the "Bed" ,"intensiveBed", IBV as data frame (n x m, 5) with name of the seized resource, time step, amount of the seized resource , name of the simulation and time.

1) Call package::babsimHospital function getrealbeds. Parse field data which is a re-arranged data from ukdataRaw. Second argument is the resource string names (Bed, IBed, IBedVent).
2) Get number of rows, setup an index variable t, starting from 0 to n-1.
3) do.call - executes function call from given resource names.
4) Return a dataframe 'fieldEvents' listing type of Beds and arrival date.

```
fieldEvents <- getRealBeds(data = data$fieldData,
                           resource = conf$ResourceNames)
```

- Extract important data

Combine the real field data with the simulation of babsim simulation results by preparing field events. Initialize the daily results (maximum) and thus get the data:

1) Call package::babsimHospital function getDailyMaxResults. Function takes in the simulation environment, field events and configuration data.
2) Set simulation dates, field start-stop, set duration from input data, get_mon_res, rounding of resource time,
3) Resource data: Mutate (create, modify delete columns, then resource binding with field columns)
4) Return Dateframe of res (resourceMaxSystem).

```
res <- getDailyMaxResults(envs = envs,  fieldEvents = fieldEvents, conf=conf)
# debug: getDailyMaxResults
#res <- debuggetDailyMaxResults(envs = envs,  fieldEvents = fieldEvents, conf=conf)
```

debug getDailyMaxResults

```
debuggetDailyMaxResults <- function (envs, fieldEvents, conf)
{
  browser()
  # Get sim dates
  resource <- time <- replication <- NULL
  ICU <- conf$ICU
  simStartDate <- as.Date(conf$simulationDates$StartDate,
    format = "%Y-%m-%d")
  fieldStartDate <- as.Date(conf$fieldDates$StartDate, format = "%Y-%m-%d")
  fieldEndDate <- as.Date(conf$fieldDates$EndDate, format = "%Y-%m-%d")
  offset <- as.numeric(fieldStartDate - simStartDate)
  duration <- as.numeric(fieldEndDate - fieldStartDate)
  total <- offset + duration
  resources <- get_mon_resources(envs)
  resources <- resources %>% dplyr::filter((time >= offset) &
    (time < total))
  resources$time <- round(resources$time)
  if (dim(resources)[1] > 0) {
    resourcesMaxSystem <- resources %>% dplyr::group_by(resource,
      time) %>% dplyr::mutate(upper = max(system)) %>%
      dplyr::mutate(lower = min(system)) %>% dplyr::mutate(med = median(system))
```

```
    resourcesMaxSystem$date <- as.Date(as.POSIXct((resourcesMaxSystem$time) *
      24 * 60 * 60, origin = simStartDate))
    resourcesMaxSystem$source <- "babsim"
  }
  else {
    resourcesMaxSystem <- resources
  }
  n <- dim(fieldEvents)[1]
  fieldEvents$server <- fieldEvents$med
  fieldEvents$queue <- rep(0, n)
  fieldEvents$capacity <- rep(Inf, n)
  fieldEvents$queue_size <- rep(Inf, n)
  fieldEvents$system <- fieldEvents$med
  fieldEvents$limit <- rep(Inf, n)
  fieldEvents$replication <- rep(1, n)
  fieldEvents$upper <- fieldEvents$med
  fieldEvents$lower <- fieldEvents$med
  resourcesMaxSystem <- dplyr::bind_rows(resourcesMaxSystem,
    fieldEvents)
  return(resourcesMaxSystem)
}
```

- Calculate error of the default parameter configuration

Error is the sum of the RMSE values for bed, intensiveBed, and intensiveBedVentilation.

1) Call package::babsimHospital function getError.
2) Perform root mean square error calculation:

a) res1 <- Resource filtering by 'r' and simulation 'babsim' - resources by simmer and trajectory paths
b) res2 <- Resource filtering by 'r' and simulation 'GA' - Actual raw data

3) A for loop cycles through the resources (B, IB, IBV), and then calls weighed_rmse fucntion taking in the actual and predicted beds.
4) The rmse error is summed, and stored back into the rmse variable.
5) The overall result is the error from all three resources.

```
errDefault <- getError(res, conf=conf)
# debuggetError
#errDefault <- debuggetError(res, conf=conf)
# Print error
print(errDefault)
```

debug getError

```
debuggetError <- function (res, conf)
{
  browser()
  # debug the rmse funtion call:
  if (conf$verbosity > 100) {
    messagef("BEGIN: getEror: ############################")
    printConf(conf)
    messagef("END: getError: ############################")
  }
  rmseBed <- 0
  resource <- conf$ResourceEval
```

```
  w <- rep(1, length(resource))
  w[2] <- conf$w2[2]
  i <- 1
  for (r in resource) {
    res1 <- res %>% filter(resource == r & source == "babsim")
    df1 <- unique(data.frame(date = res1$date, x = res1$med))
    df1 <- df1[order(df1$date), ]
    res2 <- res %>% filter(resource == r & source == "GA")
    df2 <- unique(data.frame(date = res2$date, x = res2$med))
    df2 <- df2[order(df2$date), ]
    fillDate <- which(!(df2$date %in% df1$date))
    if (length(fillDate) > 0) {
      df1 <- rbind(df1, data.frame(date = df2$date[fillDate],
        x = 0))
    }
    dfBed <- dplyr::left_join(df1, df2, by = c("date"))
    dfBed <- dfBed[complete.cases(dfBed), ]
    dfBed <- dfBed[order(dfBed$date), ]
    if (conf$verbosity > 100) {
      messagef("BEGIN: getEror: dfBed ##########################")
      printConf(dfBed)
      print(summary(dfBed$date))
      messagef("END: getError: dfBed ############################")
    }
    rmseBed <- rmseBed + w[i] * weighted_rmse(dfBed[, 3],
      dfBed[, 2])
    i <- i + 1
  }
  return(rmseBed)
}
```

- Plot results Daily max result plot shows patients are in intensive care with respect to time .Red shows our model simulation, blue real data. Black shaded area is lower and upper bounds.

1) Call package::babsimHospital function plotDailyMaxResults.
2) USer option to display beds in results.
3) ICUdataRegion - From main babsim package, sorts data via region. (NA)
4) getgolem translator function - provides language translation. (NA)
5) Configure plot titles, subtitles, legend and axis.
6) Jump to last else statement, plot results, which is the res variable parsed to the function.

7) From results, the source provides the plots of the main red line('babsim'), the aqua line is the source ('GA').
8) The GeomRibbon function from package::ggplot2, plots the lower and upper bounds, which shadow the red babsim plot. (the orange boundaries and grey shaded area)
9) All other ggplot2 package calls setup the grid, labels and titles.
10) The list of 'p' is then parsed to the plot function.
11) To enable interactive plots, an additional call to 'ggplotly()' provides this functionality.

```
p <- plotDailyMaxResults(res, showBeds = TRUE)
# debugplotDailyMaxResults
# p <- debugplotDailyMaxResults(res, showBeds = TRUE)
plot(p)
ggplotly(p)
```

debug plotDailyMaxResults:

```
debugplotDailyMaxResults <- function (results, labels = c("babsim", "DIVI"), title = "Betten: Tuerkis =
  showBeds = FALSE, icuDataRegion = NULL)
{
  browser()
  med <- lower <- upper <- resource <- NULL
  if (!showBeds) {
    results <- results[results$resource != "bed", ]
  }
  if (!is.null(icuDataRegion)) {
    icuDataRegion$bedsTotal <- icuDataRegion$faelle_covid_aktuell +
      icuDataRegion$betten_frei
    results <- as.data.frame(results)
    resultsTotal <- ddply(results[, which(!(colnames(results) ==
      "resource"))], c("source", "date"), numcolwise(sum))
    resultsTotal$resource <- "bedsTotal"
    dfDiviAll <- data.frame(resource = "bedsTotal", source = "GesamteBettenDIVI",
      date = icuDataRegion$daten_stand, upper = icuDataRegion$bedsTotal,
      lower = icuDataRegion$bedsTotal, med = icuDataRegion$bedsTotal)
    dfDiviAll <- ddply(dfDiviAll, c("resource", "source",
      "date"), numcolwise(sum))
    results <- rbind(results, resultsTotal, dfDiviAll)
    results$resource <- factor(results$resource, levels = c("bed",
      "intensiveBed", "intensiveBedVentilation", "bedsTotal"))
    dfDiviAll.extended <- dfDiviAll[which.max(dfDiviAll$date),
      ]
    dfDiviAll.extended <- rbind(dfDiviAll.extended, dfDiviAll.extended)
    dfDiviAll.extended[2, ]$date <- max(results$date)
  }
  translator <- golem::get_golem_options("translator")
  if (!is.null(translator)) {
    to_string <- as_labeller(c(bed = translator$t("label1"),
      intensiveBed = translator$t("label2"), intensiveBedVentilation = translator$t("label3"),
      bedsTotal = translator$t("bettenGesamt")))
  }
  else {
    to_string <- as_labeller(c(bed = "Bett", intensiveBed = "Nicht beatmet",
      intensiveBedVentilation = "Beatmet", bedsTotal = "Intensiv Gesamt"))
  }
  if (!is.null(translator)) {
    plotTitle <- translator$t("titlePlotBeds")
    plotXlab <- translator$t("Datum")
    plotYlab <- translator$t("bedsUsed")
    plotLegendTitle <- translator$t("Legende")
    results$source <- translator$t(as.character(results$source))
    if (!is.null(icuDataRegion)) {
      dfDiviAll.extended$source <- translator$t(as.character(dfDiviAll.extended$source))
    }
  }
  else {
    plotTitle <- "Bettenauslastung: Gemeldet DIVI und Simulation"
    plotXlab <- "Datum"
    plotYlab <- "Belegte Intensiv Betten"
```

```
    plotLegendTitle <- "Legende"
  }
  if (!is.null(icuDataRegion)) {
    results$source <- factor(results$source, levels = c(unique(results$source)[2],
      unique(results$source)[3], unique(results$source)[1]))
    p <- ggplot(results, aes(x = date, y = med, color = source)) +
      geom_line() + geom_line(data = dfDiviAll.extended,
      aes(x = date, y = med, color = source), linetype = "dashed") +
      scale_colour_manual(values = c("black", "red", "blue")) +
      geom_ribbon(aes(ymin = lower, ymax = upper), alpha = 0.4) +
      facet_grid(facets = vars(resource), labeller = labeller(resource = to_string),
        scales = "free") + ggplot2::xlab(plotXlab) +
      ggplot2::ylab(plotYlab) + ggplot2::labs(color = plotLegendTitle,
      title = plotTitle) + ggplot2::ggtitle(plotTitle)
  }
  else {
    # The code calls this plotting criteria:
    p <- ggplot(results, aes(x = date, y = med, color = source)) +
      geom_line() + geom_ribbon(aes(ymin = lower, ymax = upper),
      alpha = 0.4) + facet_grid(facets = vars(resource),
      labeller = labeller(resource = to_string), scales = "free") +
      ggplot2::xlab(plotXlab) + ggplot2::ylab(plotYlab) +
      ggplot2::labs(color = plotLegendTitle, title = plotTitle) +
      ggplot2::ggtitle(plotTitle)
  }
  return(p)
}
```

debugrunoptUK

- Comments included in code section.

```
debugrunoptUK <- function (expName = "ukl001", simData = simData, fieldData = fieldData,
  TrainFieldStartDate = Sys.Date() - 6 * 7, TrainSimStartDate = Sys.Date() -
    10 * 7, TestFieldStartDate = Sys.Date() - 4 * 7, TestSimStartDate = Sys.Date() -
    8 * 7, Overlap = 0, verbosity = 0, seed = 123, direct = FALSE,
  repeats = 1, funEvals = 35, funEvalsFactor = 0, size = 30,
  simrepeats = 2, subset = 32, parallel = FALSE, percCores = 0.8,
  icu = FALSE, icuWeights = c(1, 10), testRepeats = 3, resourceNames = c("bed",
    "intensiveBed", "intensiveBedVentilation"), resourceEval = c("bed",
    "intensiveBed", "intensiveBedVentilation"))
{
  #browser()
  library(parallel)
  # set up a null data frame
  result.df <- data.frame(x = NULL, y = NULL)
  reslist <- list()
  # Set simulation training and testing data:
  # Last day of data= "2020-11-09"
  # TrainSimData (-70 days)
  #  <----------------------|
  # TrainFieldData (-56 days)
  #        <-----------------|
  # TestSimData (-42 days)
```

```
#            <-------------|
# TestFieldData (-28 days)
#              <-----------|

# Set training data:
fieldData <- fieldData[which(fieldData$Day >= as.Date(TrainFieldStartDate)),
  ]
simData <- simData[which(simData$Day >= as.Date(TrainSimStartDate)),
  ]
TrainSimStartDate <- min(simData$Day)
EndDate <- min(max(as.Date(simData$Day)), max(as.Date(fieldData$Day)))
fieldData <- fieldData[which(fieldData$Day <= EndDate),
  ]
simData <- simData[which(simData$Day <= EndDate), ]
rownames(fieldData) <- NULL
rownames(simData) <- NULL
TrainEndDate <- as.Date(TestFieldStartDate) + Overlap
TrainSimData <- simData[which(simData$Day <= TrainEndDate),
  ]
TrainFieldData <- fieldData[which(fieldData$Day <= TrainEndDate),
  ]
syncedEndTrainDate <- min(max(TrainSimData$Day), max(TrainFieldData$Day))
TrainSimData <- simData[which(simData$Day <= syncedEndTrainDate),
  ]
TrainFieldData <- fieldData[which(fieldData$Day <= syncedEndTrainDate),
  ]
trainData <- list(simData = TrainSimData, fieldData = TrainFieldData)
print(paste("trainDataSim: ", c(min(trainData$simData$Day),
  max(trainData$simData$Day))))
print(paste("trainDataField: ", c(min(trainData$fieldData$Day),
  max(trainData$fieldData$Day))))
SIM_EQ_FIELD_TRAINDATA <- (min(as.Date(trainData$simData$Day)) ==
  as.Date(TrainSimStartDate))
if (SIM_EQ_FIELD_TRAINDATA == FALSE) {
  print(as.Date(TrainSimStartDate))
  stop("babsim.hospital::runoptDirect: Check TrainSimStartDate.")
}
SIM_EQ_FIELD_TRAINDATA <- (min(as.Date(trainData$fieldData$Day)) ==
  as.Date(TrainFieldStartDate))
if (SIM_EQ_FIELD_TRAINDATA == FALSE) {
  stop("babsim.hospital::runoptDirect: Check TrainFieldStartDate.")
}
SIM_EQ_FIELD_TRAINDATA <- (max(as.Date(trainData$simData$Day)) ==
  max(as.Date(trainData$fieldData$Day)))
if (SIM_EQ_FIELD_TRAINDATA == FALSE) {
  stop("babsim.hospital::runoptDirect: Check trainData:\n
}
FILENAME <- paste0(expName, ".RData")
print("Starting optimization loop:")
print("##########################################")
for (i in 1:repeats) {
  print(paste0("Repeat: ", i, " ################################"))
  conf <- babsimToolsConf()
```

```
trainConf <- getConfFromData(simData = trainData$simData,
  fieldData = trainData$fieldData, conf = conf)
trainConf$verbosity = verbosity
trainConf$parallel = parallel
trainConf$simRepeats = simrepeats
trainConf$ICU = icu
trainConf$ResourceNames = resourceNames
trainConf$ResourceEval = resourceEval
trainConf$percCores = percCores
trainConf$logLevel = 0
trainConf$w2 = icuWeights
trainConf$seed <- seed + i
print(paste("trainConfSim: ", c(trainConf$simulationDates$StartDate,
  trainConf$simulationDates$EndDate)))
print(paste("trainConfField: ", c(trainConf$fieldDates$StartDate,
  trainConf$fieldDates$EndDate)))
SIM_EQ_FIELD_TRAINCONF <- (min(as.Date(trainConf$simulationDates$StartDate)) ==
  as.Date(TrainSimStartDate)) & (min(as.Date(trainConf$fieldDates$StartDate)) ==
  as.Date(TrainFieldStartDate)) & (max(as.Date(trainConf$simulationDates$EndDate)) ==
  max(as.Date(trainConf$fieldDates$EndDate)))
if (SIM_EQ_FIELD_TRAINCONF == FALSE) {
  stop("babsim.hospital::runoptDirect: Check trainConf.")
}
set.seed(trainConf$seed)
funEvals <- funEvals + (i - 1) * funEvalsFactor
x0 <- getStartParameter()
# Get parameter Boundaries
# Boundaries are a default hard-coded list inside babsim::hospital.
bounds <- getBounds()
a <- bounds$lower
b <- bounds$upper
# tuning the 'UK'configured Boundaries:
a[1] <- 5   # lo AmntDaysInfectedToHospital
b[1] <- 10  # hi AmntDaysInfectedToHospital
a[16] <- 0.003 # lo FactorPatientsHospitalToVentilation - 0.005
b[16] <- 0.08  # hi FactorPatientsHospitalToVentilation - 0.02
a[20] <- 0.10  # lo FactorPatientsIntensiveToVentilation
b[20] <- 0.18  # hi FactorPatientsIntensiveToVentilation
a[14] <- 0.03  # lo $FactorPatientsInfectedToHospital 0.05
b[14] <- 0.08  # hi factor people to hospital 0.15
#b[21] <- 0.2  # hi - FactorPatientsIntensiveToDeath - 0.12

#b[18] <- 0.002 # hi -FactorPatientsNormalToVentilation - 0.002
conf <- trainConf
if (conf$verbosity > 1000) {
  print("conf before spot optimization is started")
  printConf(conf)
}
data <- trainData
g <- function(x) {
  return(rbind(a[1] - x[1], x[1] - b[1], a[2] - x[2],
    x[2] - b[2], a[3] - x[3], x[3] - b[3], a[4] -
      x[4], x[4] - b[4], a[5] - x[5], x[5] - b[5],
```

```
      a[6] - x[6], x[6] - b[6], a[7] - x[7], x[7] -
        b[7], a[8] - x[8], x[8] - b[8], a[9] - x[9],
      x[9] - b[9], a[10] - x[10], x[10] - b[10], a[11] -
        x[11], x[11] - b[11], a[12] - x[12], x[12] -
        b[12], a[13] - x[13], x[13] - b[13], a[14] -
        x[14], x[14] - b[14], a[15] - x[15], x[15] -
        b[15], a[16] - x[16], x[16] - b[16], a[17] -
        x[17], x[17] - b[17], a[18] - x[18], x[18] -
        b[18], a[19] - x[19], x[19] - b[19], a[20] -
        x[20], x[20] - b[20], a[21] - x[21], x[21] -
        b[21], a[22] - x[22], x[22] - b[22], a[23] -
        x[23], x[23] - b[23], a[24] - x[24], x[24] -
        b[24], a[25] - x[25], x[25] - b[25], a[26] -
        x[26], x[26] - b[26], a[27] - x[27], x[27] -
        b[27], x[15] + x[16] - 1, x[17] + x[18] +
        x[19] - 1, x[20] + x[21] - 1, x[23] + x[29] -
        1))
}
# Call SPOT optimiser - Using buildKriging model:
assign(expName, spot(x = x0, fun = funWrapOptimizeSim,
  lower = a, upper = b, control = list(funEvals = funEvals,
    noise = TRUE, direct = direct, designControl = list(size = size,
      retries = 1000), optimizer = optimNLOPTR,
    optimizerControl = list(opts = list(algorithm = "NLOPT_GN_ISRES"),
      eval_g_ineq = g), model = buildKriging, plots = FALSE,
    progress = TRUE), conf, data))
res <- get(expName)
reslist[[length(reslist) + 1]] <- res
x <- as.matrix(res$xbest, 1, )
print(paste0("xbest: ", x))
print("Starting test evaluation:")
print("########################################")
testPara <- mapXToPara(x)
testPara <- checkSimPara(testPara)

# Set testing data:
testFieldData <- fieldData[which(fieldData$Day >= as.Date(TestFieldStartDate)),
  ]
rownames(testFieldData) <- NULL
testSimData <- simData[which(simData$Day >= as.Date(TestSimStartDate)),
  ]
TestSimStartDate <- min(testSimData$Day)
testSimData <- testSimData[as.Date(testSimData$Day) <=
  max(as.Date(testFieldData$Day)), ]
testFieldData <- testFieldData[as.Date(testFieldData$Day) <=
  max(as.Date(testSimData$Day)), ]
syncedEndTestDate <- min(max(testSimData$Day), max(testFieldData$Day))
testSimData <- testSimData[which(testSimData$Day <=
  syncedEndTestDate), ]
TestFieldData <- fieldData[which(fieldData$Day <= syncedEndTestDate),
  ]
rownames(testSimData) <- NULL
testData <- list(simData = testSimData, fieldData = testFieldData)
```

```
  print(paste("testDataSim: ", c(min(testData$simData$Day),
    max(testData$simData$Day))))
  print(paste("testDataField: ", c(min(testData$fieldData$Day),
    max(testData$fieldData$Day))))
  SIM_EQ_FIELD_TESTDATA <- (min(as.Date(testData$simData$Day)) ==
    as.Date(TestSimStartDate))
  if (SIM_EQ_FIELD_TESTDATA == FALSE) {
    print(as.Date(TestSimStartDate))
    stop("babsim.hospital::runoptDirect: Check testData: \n
  }
  SIM_EQ_FIELD_TESTDATA <- (min(as.Date(testData$fieldData$Day)) ==
    as.Date(TestFieldStartDate))
  if (SIM_EQ_FIELD_TESTDATA == FALSE) {
    print(as.Date(TestFieldStartDate))
    stop("babsim.hospital::runoptDirect: Check testData: TestFieldStartDate.")
  }
  SIM_EQ_FIELD_TESTDATA <- (max(as.Date(testData$simData$Day)) ==
    max(as.Date(testData$fieldData$Day)))
  if (SIM_EQ_FIELD_TESTDATA == FALSE) {
    stop("babsim.hospital::runoptDirect: Check testData:\n
  }
  # set configuration tools:
  conf <- babsimToolsConf()
  testConf <- getConfFromData(simData = testSimData, fieldData = testFieldData,
    conf = conf)
  testConf$verbosity = verbosity
  testConf$parallel = parallel
  testConf$simRepeats = simrepeats
  testConf$ICU = icu
  testConf$ResourceNames = resourceNames
  testConf$ResourceEval = resourceEval
  testConf$percCores = percCores
  # Turn simmer tragectory 'print' log off!
  testConf$logLevel = 0
  testConf$w2 = icuWeights
  testConf$seed <- seed + i + 1
  set.seed(testConf$seed)
  print(paste("testConfSim: ", c(testConf$simulationDates$StartDate,
    testConf$simulationDates$EndDate)))
  print(paste("testConfField: ", c(testConf$fieldDates$StartDate,
    testConf$fieldDates$EndDate)))
  SIM_EQ_FIELD_TESTCONF <- ((testConf$simulationDates$StartDate !=
    testConf$fieldDates$StartDate) & (testConf$simulationDates$EndDate !=
    testConf$fieldDates$EndDate))
  SIM_EQ_FIELD_TESTCONF <- (min(as.Date(testConf$simulationDates$StartDate)) ==
    as.Date(TestSimStartDate)) & (min(as.Date(testConf$fieldDates$StartDate)) ==
    as.Date(TestFieldStartDate)) & (max(as.Date(testConf$simulationDates$EndDate)) ==
    max(as.Date(testConf$fieldDates$EndDate)))
  if (SIM_EQ_FIELD_TESTCONF == FALSE) {
    stop("babsim.hospital::runoptDirect: Check testConf.")
  }
  testErr <- 0
  for (j in 1:testRepeats) {
```

```
      testConf$seed <- seed + i + 1 + j
      set.seed(testConf$seed)
      envs <- modelResultHospital(para = testPara, conf = testConf,
        data = testData)
      testConf$verbosity <- 101
      err <- getError(envs, conf = testConf)
      print(paste("single test error:", err))
      testErr <- testErr + err
    }
    testErr <- testErr/testRepeats
    print(paste0("testErr: ", testErr))
    print(paste0("babsim.hospital version: ", utils::packageVersion("babsim.hospital")))
    result.df <- rbind(result.df, data.frame(y = testErr,
      x = x))
    FILENAMETMP <- paste0(expName, i, ".RData")
    print(FILENAMETMP)
    save(result.df, file = FILENAMETMP)
  }
  print(FILENAME)
  save(result.df, file = FILENAME)
  return(list(result.df, reslist))
}
```

Run optimizer

- Optimization

1) Sets up the training and test data from field and sim data start dates
2) Gets the parameters boundaries
3) Calls the SPOT optimizer using model = buildKriging
4) Returns a list of (2) of the models fit parameters - best x and best y results.

```
Sys1 <- Sys.time()
print(Sys1)
library("babsim.hospital")
library("SPOT")
library("simmer")
studyDate <- as.Date( min(conf$simulationDates$EndDate, conf$fieldDates$EndDate ))  # "2020-11-09" - th
# The following model uses the debug function runoptUK which allows for manual boundary paramters adjus
resUK <- debugrunoptUK(
  expName = paste0("UK-", format(Sys.time(), "%Y-%b.%d-%H.%M-V"), utils::packageVersion("babsim.hospital
  simData = data$simData,
  fieldData = data$fieldData,
  # TrainSimStartDate = last day of data minus 70 days
  TrainSimStartDate = studyDate - 10*7, # "2020-09-02", "2020-08-31"
  # TrainFieldStartDate = last day of data minus 56 days
  TrainFieldStartDate = studyDate - 8*7, # "2020-09-15", "2020-09-14"
  # TestSimStartDate = last day of data minus 42 days
  TestSimStartDate = studyDate - 6*7 , #"2020-09-30", "2020-09-28"
  # TestFieldStartDate = last day of data minus 28 days
  TestFieldStartDate = studyDate - 4*7, #"2020-10-23", "2020-10-12"
  Overlap = 0,
  seed = 101170,
  repeats = 2,
```

```
  funEvals = 50,
  size = 35,
  simrepeats = 2,
  parallel = TRUE,
  percCores = 0.9,
  icu = FALSE,
  icuWeights = c(1,2,10),
  verbosity = 0,
  resourceNames =  c("bed", "intensiveBed", "intensiveBedVentilation"),
  resourceEval = c("bed", "intensiveBed", "intensiveBedVentilation")
)
Sys2 <- Sys.time()
print(Sys2-Sys1)
```

Save the results from the optimiser to resUK data file. * Do not run unless optimiser has been run!!!!

```
usethis::use_data(resUK, overwrite = TRUE)
```

Get the results from the best optimiser simulation. (The one that has the lowest y-result)

```
ukpara <- resUK[[1]]
print(ukpara)
```

Create a new best parameter set called paraOpt.

```
paraOpt <- getBestParameter(resUK[[1]])
str(paraOpt)
```

```
paraOpt <- para
paraOpt$AmntDaysInfectedToHospital= 11.8
paraOpt$AmntDaysNormalToHealthy= 7.16
paraOpt$AmntDaysNormalToIntensive= 4.47
paraOpt$AmntDaysNormalToVentilation= 8.81
paraOpt$AmntDaysNormalToDeath= 6.29
paraOpt$AmntDaysIntensiveToAftercare= 8.26
paraOpt$AmntDaysIntensiveToVentilation= 4.2
paraOpt$AmntDaysIntensiveToDeath= 5.32
paraOpt$AmntDaysVentilationToIntensiveAfter = 30.4
paraOpt$AmntDaysVentilationToDeath = 17.9
paraOpt$AmntDaysIntensiveAfterToAftercare = 4.31
paraOpt$AmntDaysIntensiveAfterToDeath= 3.84
paraOpt$GammaShapeParameter = 0.798
paraOpt$FactorPatientsInfectedToHospital = 0.0757
paraOpt$FactorPatientsHospitalToIntensive = 0.0899
paraOpt$FactorPatientsHospitalToVentilation= 0.00652
paraOpt$FactorPatientsNormalToIntensive = 0.109
paraOpt$FactorPatientsNormalToVentilation = 0.000908
paraOpt$FactorPatientsNormalToDeath = 0.0931
paraOpt$FactorPatientsIntensiveToVentilation = 0.15
paraOpt$FactorPatientsIntensiveToDeath  = 0.12
paraOpt$FactorPatientsVentilationToIntensiveAfter= 0.891
paraOpt$FactorPatientsIntensiveAfterToDeath = 0.00317
paraOpt$AmntDaysAftercareToHealthy  = 3.99
paraOpt$RiskFactorA  = 0.416
paraOpt$RiskFactorB  = 0.041
paraOpt$RiskMale     = 1.38
```

```
paraOpt$AmntDaysIntensiveAfterToHealthy  = 2.3
paraOpt$FactorPatientsIntensiveAfterToHealthy  = 0.747
```

```
# Best para as provided by SPOT.
# Can be used if optimization is not possible:
# paraOpt <- getBestParameter(ukpara)
# print(paraOpt)
```

- Simulate model with paraOpt settings 10 times:

1) Parse the new optimised Uk parameters to babsim::hospital - modelResultHospital Note: This function checks the RKIriskData to adjust the arrivalTimes dataframe. As UK data is different from Germany, this function is obsolete here, and our babsim.hospital function is encapsulated further in this function.
2) If data is ICU data (from the DIVI-Intensivregister), get the risk that generates with associated arrivals. else
3) Get arrival times from data frame,
4) call babsim.hospital function as before and generate the simmer environment arrivals.

```
conf$simRepeats = 10
resOpt <- modelResultHospital(para=paraOpt,
                              conf=conf,
                              data = data)
# Debug: modelResultHospital
#resOpt <- debugmodelResultHospital(para=paraOpt, conf=conf, data = data)
```

debug modelResultHospital:

```
debugmodelResultHospital <- function (para, conf, data)
{
  browser()
  set.seed(conf$seed)
  para <- checkSimPara(para)
  if (conf$ICU) {
    rkiWithRisk <- getRkiRisk(data$simData, para)
    arrivalTimes <- data.frame(time = rkiWithRisk$time,
      risk = rkiWithRisk$Risk)
  }
  else {
    arrivalTimes <- getArrivalTimes(data$simData$Infected)
  }
  con <- list(arrivalTimes = arrivalTimes)
  con[names(data)] <- data
  data <- con
  if (conf$verbosity > 1000) {
    messagef("BEGIN: modelResultHospital() calling babsimHospital: ############################")
    printConf(conf)
    messagef("END: modelResultHospital()  ############################")
  }
  envs <- babsimHospital(arrivalTimes = data$arrivalTimes,
    conf = conf, para = para)
  fieldEvents <- getRealBeds(data = data$fieldData, resource = conf$ResourceNames)
  return(getDailyMaxResults(envs = envs, fieldEvents = fieldEvents,
    conf = conf))
}
```

- Calculate error for paraOpt simulation:

1) Call package::babsimHospital function getError.
2) Perform root mean square error calculation:

a) res1 <- Resource filtering by 'r' and simulation 'babsim' - resources by simmer and trajectory paths
b) res2 <- Resource filtering by 'r' and simulation 'GA' - Actual raw data

3) A for loop cycles through the resources (B, IB, IBV), and then calls weighed_rmse function taking in the actual and predicted beds.
4) The rmse error is summed, and stored back into the rmse variable.
5) The overall result is the error from all three resources.

```
errOpt <- getError(resOpt, conf=conf)
print(errOpt)
```

UK Results

- Plot results from optimized simulation

1) Call package::babsimHospital function plotDailyMaxResults.
2) USer option to display beds in results.
3) ICUdataRegion - From main babsim package, sorts data via region. (NA)
4) getgolem translator function - provides language translation. (NA)
5) Configure plot titles, subtitles, legend and axis.
6) Jump to last else statement, plot results, which is the res variable parsed to the function.

7) From results, the source provides the plots of the main red line('babsim'), the aqua line is the source ('GA').
8) The GeomRibbon function from package::ggplot2, plots the lower and upper bounds, which shadow the red babsim plot. (the orange boundaries and grey shaded area)
9) All other ggplot2 package calls setup the grid, labels and titles.
10) The list of 'p' is then parsed to the plot function.
11) To enable interactive plots, an additional call to 'ggplotly()' provides this functionality.

```
p <- plotDailyMaxResults(resOpt, showBeds = TRUE)
print(p)
library("plotly")
ggplotly(p)
```

- Show simulation model
  - probabilities

1) Call babsim::hospital getmatrixP - The Probabilities matrix!
2) P matrix uses only the factor parameters and stores these into the 10x10 Matrix.

```
 getMatrixP(para = paraOpt )
%>       [,1]   [,2]   [,3]    [,4]   [,5]     [,6]  [,7]    [,8]    [,9]    [,10]
%>  [1,]    0 0.9243 0.0757 0.00000 0.0000 0.000000 0.000 0.00000 0.00000 0.000000
%>  [2,]    0 1.0000 0.0000 0.00000 0.0000 0.000000 0.000 0.00000 0.00000 0.000000
%>  [3,]    0 0.0000 0.0000 0.90358 0.0899 0.006520 0.000 0.00000 0.00000 0.000000
%>  [4,]    0 0.0000 0.0000 0.00000 0.1090 0.000908 0.000 0.00000 0.09310 0.796992
%>  [5,]    0 0.0000 0.0000 0.00000 0.0000 0.150000 0.000 0.73000 0.12000 0.000000
%>  [6,]    0 0.0000 0.0000 0.00000 0.0000 0.000000 0.891 0.00000 0.10900 0.000000
%>  [7,]    0 0.0000 0.0000 0.00000 0.0000 0.000000 0.000 0.24983 0.00317 0.747000
%>  [8,]    0 0.0000 0.0000 0.00000 0.0000 0.000000 0.000 0.00000 0.00000 1.000000
%>  [9,]    0 0.0000 0.0000 0.00000 0.0000 0.000000 0.000 0.00000 1.00000 0.000000
```

```
%> [10,]    0 0.0000 0.0000 0.00000 0.0000 0.000000 0.000 0.00000 0.00000 1.000000
```

VisualizeGraph

1) call babsim::hospital visualizeGraph
2) Set up hard-coded state names (10x states).
3) Get the Probabilities Matrix that shows the % trajectory paths.
4) Set up plotting properties - plot.igraph.

```
visualizeGraph(para=paraOpt, option = "P")
```

```
debugvisualizeGraph <- function (para = babsimHospitalPara(), option = "P")
{

  browser()
  states <- c("infec", "out", "hosp", "normal", "intens",
    "vent", "intafter", "aftercare", "death", "healthy")
  para <- checkSimPara(para = para)
  P <- getMatrixP(para = para)
  D <- getMatrixD(para = para)
  M <- new("markovchain", transitionMatrix = P, states = states,
    name = "bubsim")
  for (i in 1:dim(P)[1]) {
    M@transitionMatrix[i, i] <- 0
  }
  g <- as(M, "igraph")
  for (i in states) {
    igraph::V(g)[i]$color <- "white"
  }
  igraph::V(g)[8]$color <- "lightgreen"
  igraph::V(g)[4]$color <- "yellow"
  igraph::V(g)[6]$color <- "yellow"
  igraph::V(g)[5]$color <- "orange"
  igraph::V(g)[9]$color <- "red"
  igraph::V(g)[10]$color <- "lightgreen"
  igraph::V(g)[2]$color <- "coral1"
  igraph::V(g)[3]$color <- "aliceblue"
  igraph::V(g)[7]$color <- "aliceblue"
  coords <- igraph::layout.reingold.tilford
  if (option == "P") {
    elabs <- round(igraph::E(g)$prob * 100, 1)
  }
  else {
    elabs <- round(c(D[1, 2], D[1, 3], D[3, 4], D[3, 5],
      D[3, 6], D[4, 5], D[4, 6], D[4, 9], D[4, 10], D[5,
        6], D[5, 8], D[5, 9], D[6, 7], D[6, 9], D[7,
        8], D[7, 9], D[7, 10], D[8, 10]), 1)
  }
  translator <- golem::get_golem_options("translator")
  if (!is.null(translator)) {
    if (option == "P") {
      TITLE <- translator$t("uebmap")
    }
    else {
```

```
      TITLE <- translator$t("uebmap2")
    }
  }
  else {
    if (option == "P") {
      TITLE <- "Wahrscheinlichkeiten (Prozent)"
    }
    else {
      TITLE <- "Dauern (Tage)"
    }
  }
  plot.igraph(g, vertex.color = igraph::V(g)$color, vertex.size = 25,
    vertex.label.cex = 1.2, edge.arrow.size = 0.5, edge.label = elabs,
    edge.label.cex = 1, edge.curved = 0.28, rescale = TRUE,
    layout = coords, asp = 0.9, main = TITLE)
}
```

- Show simulation model:
  - duration

1) The D matrix is the amount of days a patient will remain in a state.
2) It is a 10x10 Matrix.

```
getMatrixD(para = paraOpt)
%>       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]  [,9] [,10]
%>  [1,]    0    0 11.8    0 0.00 0.00  0.0 0.00  0.00  0.00
%>  [2,]    0    0  0.0    0 0.00 0.00  0.0 0.00  0.00  0.00
%>  [3,]    0    0  0.0    0 0.00 0.00  0.0 0.00  0.00  0.00
%>  [4,]    0    0  0.0    0 4.47 8.81  0.0 0.00  6.29  7.16
%>  [5,]    0    0  0.0    0 0.00 4.20  0.0 8.26  5.32  0.00
%>  [6,]    0    0  0.0    0 0.00 0.00 30.4 0.00 17.90  0.00
%>  [7,]    0    0  0.0    0 0.00 0.00  0.0 4.31  3.84  2.30
%>  [8,]    0    0  0.0    0 0.00 0.00  0.0 0.00  0.00  3.99
%>  [9,]    0    0  0.0    0 0.00 0.00  0.0 0.00  0.00  0.00
%> [10,]    0    0  0.0    0 0.00 0.00  0.0 0.00  0.00  0.00
```

- Plot the Duration graph.

1) Call babsim::hospital visualizeGraph
2) Set up hard-coded state names (10x states).
3) Get the duration matrix properties (Amount of days).
4) Set up plotting properties - plot.igraph.

```
visualizeGraph(para = paraOpt, option = "D")
```

Summary

- This report describes experimental methods for tuning algorithms.
- Using a simple simulated annealing algorithm, it was demonstrated how optimization algorithms can be tuned using the SPOT.
- Several tools from the SPOT for automated and interactive tuning were illustrated and the underling concepts of the SPOT approach were explained.
- Central in the SPOT approach are techniques such as exploratory fitness landscape analysis and response surface methodology.

- Furthermore, we demonstrated how SPOT can be used as optimizer and how a sophisticated ensemble approach is able to combine several meta models via stacking.