Hospital Capacity Planning Using Discrete Event Simulation: Introduction

Thomas Bartz-Beielstein

IDEA, TH Köln
thomas.bartz-beielstein@th-koeln.de

Frederik Rehbach

IDEA, TH Köln
frederik.rehbach@th-koeln.de

Olaf Mersmann

IDEA, TH Köln
olaf.mersmann@th-koeln.de

Eva Bartz

Bartz und Bartz GmbH
eva.bartz@bartzundbartz.de

2021-07-22

Abstract

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

Introduction

- Resource and capacity planning for hospitals.
- Ideas used in this packages are based on the paper: A novel modeling technique to predict resource-requirements in critical care: a case study (Lawton, McCooe) see (Lawt19a).
- `babsim.hospital` implements a discrete-event simulation, which uses the `simmer` package.

Data and Copyright Notes

The simulator requires two types of data:

1. **simulation data** that describes the spread of the pandemic over time and
2. **field data** that contains daily resource usage data.

Included in the package are tools to generate synthetic data, e.g., you can generate simulation and field data to run the simulations. This procedure is described in the paper "Optimization of High-dimensional Simulation Models Using Synthetic Data".

To demonstrate the usage of real-world data, we have included sample datasets from Germany.

Simulation Data

We have included a data sample from the German Robert Koch-Institute (RKI). Please take the following

copyright notice under advisement, if you plan to use the RKI data included in the package:

> Die Daten sind die „Fallzahlen in Deutschland“ des Robert Koch-Institut (RKI) und stehen unter der Open Data Datenlizenz Deutschland Version 2.0 zur Verfügung. Quellenvermerk: Robert Koch-Institut (RKI), dl-de/by-2-0
> Haftungsausschluss: „Die Inhalte, die über die Internetseiten des Robert Koch-Instituts zur Verfügung gestellt werden, dienen ausschließlich der allgemeinen Information der Öffentlichkeit, vorrangig der Fachöffentlichkeit“.

Taken from here.

Field Data

We have included a data sample from the German DIVI Register. Please take the following copyright notice under advisement. The DIVI data are not open data. The following statement can be found on the DIVI web page: > Eine weitere wissenschaftliche Nutzung der Daten ist nur mit Zustimmung der DIVI gestattet. Therefore, only an example data set, that reflects the structure of the original data from the DIVI register, is included in the `babsim.hospital` package as `icudata`.

Packages

```
suppressPackageStartupMessages({
library("SPOT")
library("babsim.hospital")
library("simmer")
library("simmer.plot")
})
```

We need at least version 2.1.8 of `SPOT`.

```
packageVersion("SPOT")
%> [1] '2.5.6'
```

Data used by `babsim.hospital`

We combine data from two different sources:

1. `simData`: simulation data, i.e., input data for the simulation. Here, we will use data from the Robert Koch-Institute in Germany.
2. `fieldData`: real data, i.e., data from the DIVI-Intensivregister. The field data is used to validate the output of the simulation.

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 will be used as simulation input.

We will take a closer look at the required input data in the following sections.

Simulation Data: RKI Data

Get Data From the RKI Server

`babsim.hospital` provides a function to update the (daily) RKI data.

```
updateRkidataFile("https://www.arcgis.com/sharing/rest/content/items/f10774f1c63e40168479a1feb6c7ca74/da
```

Users are expected to adapt this function to their local situation.

The downloaded data will be available as `rkidata`.

Due to data size limits on CRAN, the full dataset is not included in the `babsim.hospital`package. Instead, we provide a subset of the Robert Koch-Institut dataset with 10,000 observations in the package.

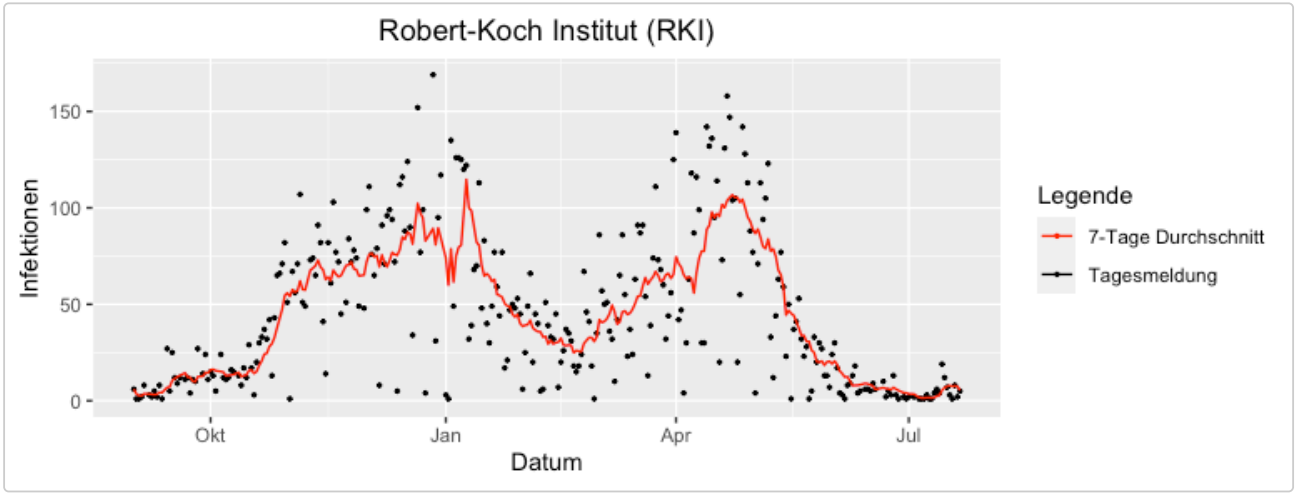
```
str(babsim.hospital::rkidata)
%> 'data.frame':    1909704 obs. of  18 variables:
%>  $ FID                 : int  1 2 3 4 5 6 7 8 9 10 ...
%>  $ IdBundesland        : int  1 1 1 1 1 1 1 1 1 1 ...
%>  $ Bundesland          : chr  "Schleswig-Holstein" "Schleswig-Holstein" "Schleswig-
      Holstein" "Schleswig-Holstein" ...
%>  $ Landkreis           : chr  "SK Flensburg" "SK Flensburg" "SK Flensburg" "SK Flensburg"
      ...
%>  $ Altersgruppe        : chr  "A00-A04" "A00-A04" "A00-A04" "A00-A04" ...
%>  $ Geschlecht          : chr  "M" "M" "M" "M" ...
%>  $ AnzahlFall          : int  1 1 1 1 1 1 2 1 1 1 ...
%>  $ AnzahlTodesfall     : int  0 0 0 0 0 0 0 0 0 0 ...
%>  $ Refdatum            : chr  "2020/09/30 00:00:00" "2020/10/29 00:00:00" "2020/11/03
      00:00:00" "2020/11/20 00:00:00" ...
%>  $ IdLandkreis         : int  1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 ...
%>  $ Datenstand          : chr  "22.07.2021, 00:00 Uhr" "22.07.2021, 00:00 Uhr" "22.07.2021,
      00:00 Uhr" "22.07.2021, 00:00 Uhr" ...
%>  $ NeuerFall           : int  0 0 0 0 0 0 0 0 0 0 ...
%>  $ NeuerTodesfall      : int  -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 ...
%>  $ Meldedatum          : chr  "2020/09/30 00:00:00" "2020/10/29 00:00:00" "2020/11/03
      00:00:00" "2020/11/19 00:00:00" ...
%>  $ NeuGenesen          : int  0 0 0 0 0 0 0 0 0 0 ...
%>  $ AnzahlGenesen       : int  1 1 1 1 1 1 2 1 1 1 ...
%>  $ IstErkrankungsbeginn: int  0 0 0 1 1 1 0 1 0 1 ...
%>  $ Altersgruppe2       : chr  "Nicht übermittelt" "Nicht übermittelt" "Nicht übermittelt"
      "Nicht übermittelt" ...
```

Copyright notice for the data:

> Die Daten sind die „Fallzahlen in Deutschland“ des Robert Koch-Institut (RKI) und stehen unter der Open Data Datenlizenz Deutschland Version 2.0 zur Verfügung. Quellenvermerk: Robert Koch-Institut (RKI), dl-de/by-2-0
> Haftungsausschluss: „Die Inhalte, die über die Internetseiten des Robert Koch-Instituts zur Verfügung gestellt werden, dienen ausschließlich der allgemeinen Information der Öffentlichkeit, vorrangig der Fachöffentlichkeit“.

The `rkidata` can be visualized as follows (here `region = 0` is Germany, `region = 5` is North Rhine-Westphalia, `region = 5374` Oberbergischer Kreis, etc.):

```
p <- ggVisualizeRki(data=babsim.hospital::rkidata, region = 5374)
print(p)
```



Preprocessed RKI Data

Not all the information from the original `rkidata` data set is required by the `babsim.hospital` simulator. The function `getRkiData()` extracts the subset of the raw `rkidata` required by our simulation, optimization, and analysis:

```
rki <- getRkiData(rki = rkidata)
%> getRkiData(): Found days with negative number of cases. Ignoring them.
str(rki)
%> 'data.frame':    3505596 obs. of  7 variables:
%>  $ Altersgruppe: chr  "A15-A34" "A35-A59" "A15-A34" "A35-A59" ...
%>  $ Geschlecht  : chr  "M" "W" "M" "W" ...
%>  $ Day         : Date, format: "2020-09-01" "2020-09-01" ...
%>  $ IdBundesland: int  1 1 1 1 1 1 1 1 1 1 ...
%>  $ IdLandkreis : int  1002 1002 1004 1004 1053 1053 1053 1053 1054 1056 ...
%>  $ time        : num  0 0 0 0 0 0 0 0 0 0 ...
%>  $ Age         : num  25 47 25 47 2 25 25 70 25 10 ...
```

As illustrated by the output from above, we use the following data: 1. `Altersgruppe`: age group (intervals, categories), represented as character string 1. `Geschlecht`: gender 1. `Day`: day of the infection 1. `IdBundesland`: federal state 1. `IdLandkreis`: county 1. `time`: number of days (`0` = start data). It will be used as `arrivalTimes` for the `simmer` simulations. 1. `Age`: integer representation of `Altersgruppe`

Field Data (Real ICU Beds)

Get Data From the DIVI Server

Similar to the `rkidata`, which is available online and can be downloaded from the RKI Server, the field data is also available online. It can be downloaded from the DIVI Server as follows, where `YYYY-MM-DD` should be replaced by the current date, e.g, `2020-10-26`.

```
updateIcudataFile("https://www.divi.de/joomlatools-files/docman-files/divi-intensivregister-
        tagesreports-csv/DIVI-Intensivregister_YYYY-MM-DD_12-15.csv")
```

Note: The data structures on the DIVI server may change, so it might be necessary to modify the following procedure. Please check the hints on the DIVI web page. Contrary to the `updateRkidataFile()` function, which downloads the complete historical dataset, the `updateIcudataFile()` function only downloads data for a single date.

The downloaded data will be available in `babsim.hospital` as `icudata`.

Important: The DIVI dataset is *not* open data. The following statement can be found on the DIVI web page:

> Eine weitere wissenschaftliche Nutzung der Daten ist nur mit Zustimmung der DIVI gestattet.

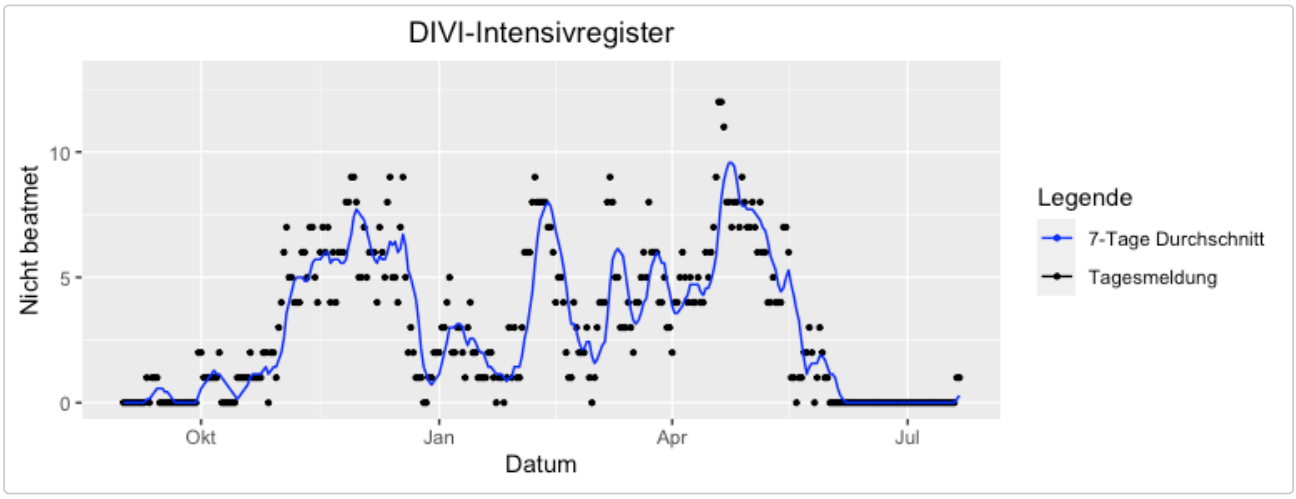
Therefore, only an example data set, that reflects the structure of the original data from the DIVI register, is included in the `babsim.hospital` package as `icudata`:

```
str(babsim.hospital::icudata)
%> 'data.frame':    128603 obs. of  11 variables:
%>  $ bundesland                  : int  1 1 1 1 1 1 1 1 1 1 ...
%>  $ gemeindeschluessel          : int  1001 1002 1003 1004 1051 1053 1054 1055 1056 1057 ...
%>  $ anzahl_meldebereiche        : int  3 5 2 1 1 2 3 3 2 1 ...
%>  $ faelle_covid_aktuell        : int  0 1 0 0 0 0 0 0 1 0 ...
%>  $ faelle_covid_aktuell_beatmet: int  0 0 0 0 0 0 0 0 0 0 ...
%>  $ anzahl_standorte            : int  2 3 2 1 1 2 3 3 2 1 ...
%>  $ betten_frei                 : int  24 116 103 9 13 11 15 17 9 7 ...
%>  $ betten_belegt               : int  31 113 114 16 41 13 24 35 28 5 ...
%>  $ daten_stand                 : Date, format: "2020-09-01" "2020-09-01" ...
%>  $ betten_belegt_nur_erwachsen : int  NA NA NA NA NA NA NA NA NA NA ...
%>  $ betten_frei_nur_erwachsen   : int  NA NA NA NA NA NA NA NA NA NA ...
```

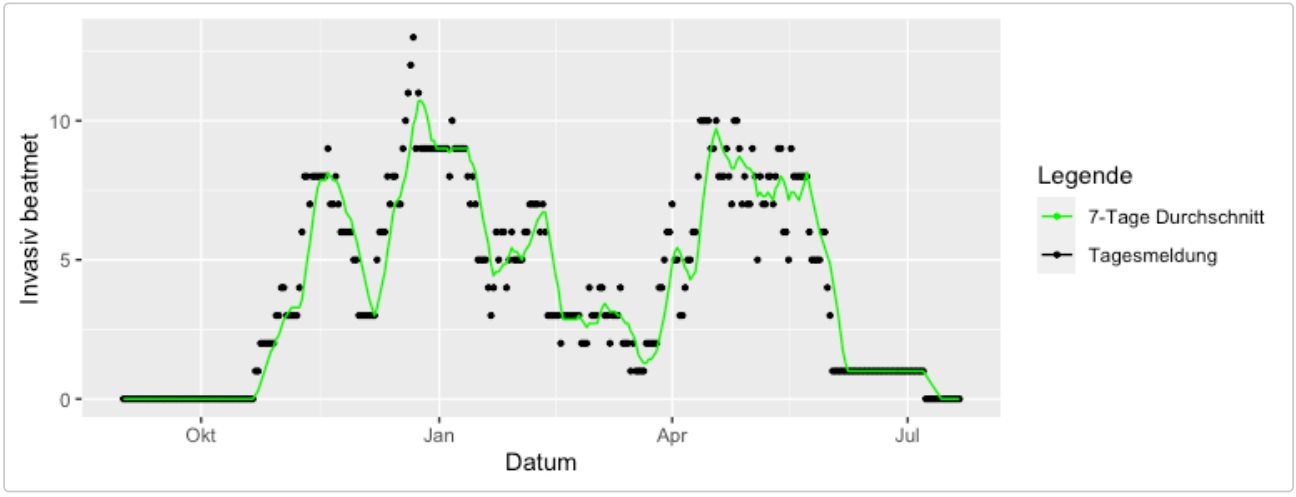
The `icudata` can be visualized as follows (`region = 0` is Germany, `region = 5` is North Rhine-Westphalia, `region = 5374` is the Oberbergischer Kreis, etc.)

```
p <- ggVisualizeIcu(region = 5374)
```

```
print(p[[1]])
```




```
print(p[[2]])
```



Additional Data

GV-ISys: Gemeindeverzeichnis-Informationssystem (GV-ISys) of the German Federal Statistics Office

- Gemeindeverzeichnis-Informationssystem (GV-ISys) of the German Federal Statistics Office, see https://www.destatis.de/DE/Themen/Laender-Regionen/Regionales/Gemeindeverzeichnis/

Preprocessing DIVI/ICU Data

Note: ICU beds without ventilation can be calculated as `faelle_covid_aktuell - faelle_covid_aktuell_beatmet`.

The function `getIcuBeds()` converts the 9 dimensional DIVI ICU dataset `icudata` (bundesland,gemeindeschluessel,..., daten_stand) into a data.frame with three columns:

1. `bed`
2. `intensiveBedVentilation`
3. `Day`

```
fieldData <- getIcuBeds(babsim.hospital::icudata)
str(fieldData)
%> 'data.frame':    324 obs. of  3 variables:
%>  $ intensiveBed           : int  103 103 96 97 97 92 94 100 94 104 ...
%>  $ intensiveBedVentilation: int  132 125 127 128 126 126 134 130 133 129 ...
%>  $ Day                    : Date, format: "2020-09-01" "2020-09-02" ...
```

The field data based on `icudata` uses two bed categories: 1. `intensiveBed`: ICU bed without ventilation 2. `intensiveBedVentilation`: ICU bed with ventilation

Performing Simulations

To run a simulation, the setting must be configured (seed, number of repeats, sequential or parallel evaluation, variable names, dates, etc.)

```
region = 5374 # Germany, 5315 is Cologne, 5 is NRW
seed = 123
simrepeats = 2
```

```
parallel = FALSE
percCores = 0.8
resourceNames =  c("intensiveBed", "intensiveBedVentilation")
resourceEval = c("intensiveBed", "intensiveBedVentilation")
```

We can specify the field data based on `icudata` (DIVI) for the simulation as follows:

```
FieldStartDate = "2020-09-01"
# Felddaten (Realdaten, ICU):
icudata <- getRegionIcu(data = icudata, region = region)
fieldData <- getIcuBeds(icudata)
fieldData <- fieldData[which(fieldData$Day >= as.Date(FieldStartDate)), ]
rownames(fieldData) <- NULL
icu = TRUE
icuWeights = c(1,1)
```

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

```
SimStartDate = "2020-08-01"
rkidata <- getRegionRki(data = rkidata, region = region)
simData <- getRkiData(rkidata)
simData <- simData[which(simData$Day >= as.Date(SimStartDate)), ]
## Auch mit fieldData cutten damit es immer das gleiche Datum ist
simData <- simData[as.Date(simData$Day) <= max(as.Date(fieldData$Day)),]
## time must start at 1
simData$time <- simData$time - min(simData$time)
rownames(simData) <- NULL
```

Finally, we combine all field and simulation data into a single `list()` called `data`:

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

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 <- getConfFromData(conf = conf,
                        simData = data$simData,
                        fieldData = data$fieldData)
conf$parallel = parallel
conf$simRepeats = simrepeats
conf$ICU = icu
conf$ResourceNames = resourceNames
conf$ResourceEval = resourceEval
conf$percCores = percCores
conf$logLevel = 0
conf$w2 = icuWeights
set.seed(conf$seed)
```

Simulation Model Parameters

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. There are currently 29 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
  - `arrivalTimes`
  - configuration list `conf`
  - parameter list `para` for the simulation.
- Arrival times were not discussed yet.
- `babsim.hospital` provides the function `getRkiRisk()` that generates arrivals with associated risks.
- `Risk` is based on age (`Altersgruppe`) and gender (`Geschlect`):

```
rkiWithRisk <- getRkiRisk(data$simData, para)
head(rkiWithRisk)
%>   Altersgruppe Geschlecht        Day IdBundesland IdLandkreis time Age
%> 1      A15-A34          M 2020-09-01            5        5374    0  25
%> 2      A15-A34          M 2020-09-01            5        5374    0  25
%> 3      A15-A34          M 2020-09-01            5        5374    0  25
%> 4      A35-A59          M 2020-09-01            5        5374    0  47
%> 5      A35-A59          W 2020-09-01            5        5374    0  47
%> 6      A35-A59          W 2020-09-01            5        5374    0  47
%>         Risk
%> 1 0.03946352
%> 2 0.03946352
%> 3 0.03946352
```

```
%> 4 0.04917457
%> 5 0.03278305
%> 6 0.03278305
```

- To perform simulations, only two parameters are required:
  - `time`: arrival time
  - `Risk`: risk (based on age and gender)
- A data.frame with these two parameters is passed to the main simulation function `babsimHospital`.
- Output from the simulation is stored in the variable `envs`.

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`

- 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 3.6251321.

- Here, we illustrate how `babsim` plots can be generated.
- Important note: Because we do not use the full data, simulation results are completely wrong!
- The following figures are included to demonstrate the working principles of the visualization procedures.

```
p <- plotDailyMaxResults(res)
plot(p)
```

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

```
plotly::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()
```

- `babsim` provides an interface to optimize the parameter values of the simulation model.
- The following code is just a quick demo.
- Warning: To run the following code, the complete `rkidata` and `icudata` data sets must be available. Please download the data from RKI and DIVI or provide your own simulation and field data!
- Note: results are stored in the directory `results`.

```
library("babsim.hospital")
library("SPOT")
library("simmer")
dir.create("results")
res202010262 <- runoptDirect(
  expName = paste0("test_", format(Sys.time(), "%Y_%b.%d_%H.%M_V"),
        utils::packageVersion("babsim.hospital"),"R"),
  region = 5374,
  rkiwerte = babsim.hospital::rkidata,
  icuwerte = babsim.hospital::icudata,
  TrainSimStartDate = Sys.Date() - 10*7,
  TrainFieldStartDate = Sys.Date() - 6*7,
  TestSimStartDate = Sys.Date() - 8*7,
  TestFieldStartDate = Sys.Date() - 4*7,
  Overlap = 0,
  seed = 101170,
  direct = TRUE,
  repeats = 1,
```

```
  funEvals = 40,
  funEvalsFactor = 0,
  size = 35,
  simrepeats = 1,
  parallel = TRUE,
  percCores = 0.9,
  icu = TRUE,
  icuWeights = c(1,1),
  verbosity=11,
  testRepeats = 1,
  tryOnTestSet = TRUE
)
```

- Results from the `runopt()` runs are stored in the `paras.rda` file.
- `babsim.hospital` provides results from the following regions (towns and counties in Germany):
  - `getParaSet(5374)`: Oberbergischer Kreis
  - `getParaSet(5315)`: City of Cologne
  - `getParaSet(5)`: North-Rhine Westphalia
  - `getParaSet(0)`: Germany

Use Optimized Parameters

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

```
para <- getBestParameter(getParaSet(5315))
res <- modelResultHospital(para=para,
                           conf=conf,
                           data = data)
resOpt <- getError(res, conf=conf)
```

- Optimization improves the error
- This improvement can also be visualized.

```
p <- plotDailyMaxResults(res)
print(p)
```

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

```
plotly::ggplotly(p)
```

Smooth Parameters

- Smooth a parameter set using another parameter set
- Calculate the average of two parameter sets to smooth out any local anomalies.
- Mostly useful to smooth out a local (say OBK) parameter set using a global one (say NRW).
- Technically this function calculates `(1-weight) * para + weight * other` ensuring that the names etc. of `para` are preserved.
- Parameters:
  - para Parameter set to smooth
  - other Other parameters to average in
  - weight Weight of other parameters
- return Weighted parameter set

```
para <- smoothParameter(getBestParameter(getParaSet(5374)),
```

```
                getBestParameter(getParaSet(5)) )
```

Visualize and Analyze Parameter Settings

- `babsim.hospital` includes 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).
- The following plot illustrates the transition probabilities.
- States are as follows:
  - `infec`: infected
  - `out`: transfer out, no hospital required
  - `hosp`: hospital
  - `normal`: normal station, no ICU
  - `intens`: ICU (without ventilation)
  - `vent`: ICU ventilated
  - `intafter`: intensive aftercare (from ICU with ventilation, on ICU)
  - `aftercare`: aftercare (from ICU, on normal station)
  - `death`: patient dies
  - `healthy`: recovered
- The transition matrix, that stores the probabilities, is shown below:

```
para <- babsimHospitalPara()
getMatrixP(para = para)
%>       [,1] [,2] [,3] [,4] [,5]  [,6] [,7]    [,8]  [,9] [,10]
%>  [1,]    0  0.9  0.1  0.0 0.00 0.000  0.0 0.00000 0e+00 0.000
%>  [2,]    0  1.0  0.0  0.0 0.00 0.000  0.0 0.00000 0e+00 0.000
%>  [3,]    0  0.0  0.0  0.9 0.09 0.010  0.0 0.00000 0e+00 0.000
%>  [4,]    0  0.0  0.0  0.0 0.10 0.001  0.0 0.00000 1e-01 0.799
%>  [5,]    0  0.0  0.0  0.0 0.00 0.300  0.0 0.60000 1e-01 0.000
%>  [6,]    0  0.0  0.0  0.0 0.00 0.000  0.7 0.00000 3e-01 0.000
%>  [7,]    0  0.0  0.0  0.0 0.00 0.000  0.0 0.32999 1e-05 0.670
%>  [8,]    0  0.0  0.0  0.0 0.00 0.000  0.0 0.00000 0e+00 1.000
%>  [9,]    0  0.0  0.0  0.0 0.00 0.000  0.0 0.00000 1e+00 0.000
%> [10,]    0  0.0  0.0  0.0 0.00 0.000  0.0 0.00000 0e+00 1.000


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

- Similar to the probabilities, 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] [,10]
%>  [1,]    0    0  9.5    0    0 0.00    0    0    0     0
%>  [2,]    0    0  0.0    0    0 0.00    0    0    0     0
%>  [3,]    0    0  0.0    0    0 0.00    0    0    0     0
%>  [4,]    0    0  0.0    0    5 3.63    0    0    5    10
%>  [5,]    0    0  0.0    0    0 4.00    0    7    5     0
%>  [6,]    0    0  0.0    0    0 0.00   30    0   20     0
%>  [7,]    0    0  0.0    0    0 0.00    0    3    4     3
%>  [8,]    0    0  0.0    0    0 0.00    0    0    0     3
%>  [9,]    0    0  0.0    0    0 0.00    0    0    0     0
```

```
%> [10,]    0    0  0.0    0    0 0.00    0    0    0     0
```

Extend RKI Data

- `babsim.hospital`can be used to simulate scenarios, i.e., possible developments of the pandemic.
- To simulate these scenarios, arrival events must be generated.
- The function `extendRki()` adds new arrival events.
- To generate new arrivals, three parameters must be specified:
  - `data`: an already existing data set, i.e., the history
  - `EndDate`: last day of the simulated data (in the future)
  - `R0`: base reproduction values (R0) at the first day of the scenario and at the last day of the scenario. A linear interpolation between these two values will be used, e.g., if `R0 = c(1,2)` and ten eleven days are specified, the following R0 values will be used: (1.0, 1.1, 1.2, 1.3, …, 1.9,2.0).

```
data <- getRkiData(babsim.hospital::rkidata)
%> getRkiData(): Found days with negative number of cases. Ignoring them.
n <-  as.integer( max(data$Day)-min(data$Day) )
StartDay <- min(data$Day) + round(n*0.9)
data <- data[which(data$Day >=  StartDay), ]
EndDate <- max(data$Day) + 2
dataExt <- extendRki(data = data,
                     EndDate = EndDate,
                     R0 = c(0.1, 0.2))
```

- To illustrate the `extendRki()` data extension procedure, a short example is shown below:

```
visualizeRkiEvents(data = data, region=5374)
```

- The following plot shows the result of the data extension:

```
visualizeRkiEvents(data = dataExt, region = 5374)
```

Sensitivity Analysis

Quick Analysis

- OBK Data

```
library("rpart")
library("rpart.plot")
library("babsim.hospital")
library("SPOT")
param <- getParaSet(5374)
n <- dim(param)[2] - 1
y <- param[,1]
x <- param[,2:dim(param)[2]]
fitTree <- buildTreeModel(x=x,
                y=y,
                control = list(xnames = paste0('x', 1:n)))
rpart.plot(fitTree$fit)
```

```
getParameterNameList(c(24, 25, 3, 10))
```

```
library("rpart")
library("rpart.plot")
param <- getParaSet(5315)
n <- dim(param)[2] - 1
y <- param[,1]
x <- param[,2:dim(param)[2]]
fitTree <- buildTreeModel(x=x,
                y=y,
                control = list(xnames = paste0('x', 1:n)))
rpart.plot(fitTree$fit)
```

Sensitivity Analysis

- `babsim.hopital` uses the `R` package SPOT (sequential parameter optimization toolbox) to improve parameter settings.
- `SPOT` implements a set of tools for model-based optimization and tuning of algorithms (surrogate models, optimizers, DOE).
- `SPOT` can be used for sensitivity analysis, which is in important under many aspects, especially:
  - understanding the most important factors (parameters) that influence model behavior. For example, it is of great importance for simulation practitioners and doctors to discover relevant durations and probabilities.
  - detecting interactions between parameters, e.g., do durations influence each other?
- Before visualizations are presented, we show the underlying parameter setting.

```
res <- res202010262[[2]][[1]]
xBest <- res$xbest
n <- length(xBest)
print(xBest)
```

- The corresponding parameter names are:

```
n <- n-1
t(getParameterNameList(1:n))
```

- The fitness landscape can be visualized using the function `plotModel`.
- Note, `plotModel` requires two parameter values.
- In the following example, `GammaShapeParameter` (x16) and `AmntDaysNormalToHealthy` (x2) were chosen.
- The plot can be interpreted as follows:
  - The model error is reduced, if patients stay longer on the normal station before they leave the hospital (healthy).
  - The effect of the parameter `GammaShapeParameter` is smaller than the effect of the parameter `AmntDaysNormalToHealthy`.

```
SPOT::plotModel(res$modelFit, which = c(16,2) ,xlab = c("Modellierungsparameter (Varianz),
        GammaShapeParameter", "x2: Normalstation zu Genesen (AmntDaysNormalToHealthy)"))
```

A regression-based parameter screening can be performed to discover relevant (and irrelevant) model parameters:

```
fitLm <- SPOT::buildLM(x=res$x,
                y=res$y,
                control = list(useStep=TRUE))
```

```
summary(fitLm$fit)
```

- Parameter x_7 should be considered important.

```
getParameterName(7)
```

- This finding is supported by a simple regression tree analysis:

```
library("rpart")
library("rpart.plot")
fitTree <- buildTreeModel(x=res$x,
                  y=res$y,
                  control = list(xnames = paste0('x', 1:n)))
rpart.plot(fitTree$fit)
```

Estimate Risks

An exponential model with two parameters was chosen to model `risk` as a function of `age` and `gender`: $r(x) = a \exp(b\,x)$.

```
age <- c(2,10,25,47,70,90)
risk <- c(0.01,0.07,0.15,0.65,3,12.64)
fit <- nls(risk ~ a * exp( b * age),
           start = list(a = 1, b = 0),
           control= nls.control(maxiter = 50, tol = 1e-05, minFactor = 1/1024,
                                printEval = FALSE, warnOnly = FALSE))
print(coef(fit))
%>          a          b
%> 0.02048948 0.07138200


{plot(age,2*risk)
    # female:
    lines(age, 1* predict(fit, list(x = age)))
    # male:
    lines(age, 2* predict(fit, list(x = age) ), col ="red")}
```

The RKI Data Classes

Raw data: rkidatafull

The full, unmodified RKI data set, can be downloaded from the RKI web page. Once downloaded, it is accessible as `rkidataFull`.

Note: `rkidataFull` is a large data set, which is not included in the CRAN version.

```
dim(babsim.hospital::rkidataFull)
```

Class rkidata

The `rkidata` data set is a subset of the `rkidataFull` data set. * It contains data from `2020-09-01` until today. *

The `rkidata` subset is used, because the COVID-19 pandemic behavior changed over time. The period from September is sometimes referred to as the second wave. * Note: the full `rkidata` set is not included in the CRAN version. * The CRAN version includes a smaller data set:

```
str(babsim.hospital::rkidata)
%> 'data.frame':    1909704 obs. of  18 variables:
%>  $ FID                 : int  1 2 3 4 5 6 7 8 9 10 ...
%>  $ IdBundesland        : int  1 1 1 1 1 1 1 1 1 1 ...
%>  $ Bundesland          : chr  "Schleswig-Holstein" "Schleswig-Holstein" "Schleswig-
      Holstein" "Schleswig-Holstein" ...
%>  $ Landkreis           : chr  "SK Flensburg" "SK Flensburg" "SK Flensburg" "SK Flensburg"
      ...
%>  $ Altersgruppe        : chr  "A00-A04" "A00-A04" "A00-A04" "A00-A04" ...
%>  $ Geschlecht          : chr  "M" "M" "M" "M" ...
%>  $ AnzahlFall          : int  1 1 1 1 1 1 2 1 1 1 ...
%>  $ AnzahlTodesfall     : int  0 0 0 0 0 0 0 0 0 0 ...
%>  $ Refdatum            : chr  "2020/09/30 00:00:00" "2020/10/29 00:00:00" "2020/11/03
      00:00:00" "2020/11/20 00:00:00" ...
%>  $ IdLandkreis         : int  1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 ...
%>  $ Datenstand          : chr  "22.07.2021, 00:00 Uhr" "22.07.2021, 00:00 Uhr" "22.07.2021,
      00:00 Uhr" "22.07.2021, 00:00 Uhr" ...
%>  $ NeuerFall           : int  0 0 0 0 0 0 0 0 0 0 ...
%>  $ NeuerTodesfall      : int  -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 ...
%>  $ Meldedatum          : chr  "2020/09/30 00:00:00" "2020/10/29 00:00:00" "2020/11/03
      00:00:00" "2020/11/19 00:00:00" ...
%>  $ NeuGenesen          : int  0 0 0 0 0 0 0 0 0 0 ...
%>  $ AnzahlGenesen       : int  1 1 1 1 1 1 2 1 1 1 ...
%>  $ IstErkrankungsbeginn: int  0 0 0 1 1 1 0 1 0 1 ...
%>  $ Altersgruppe2       : chr  "Nicht übermittelt" "Nicht übermittelt" "Nicht übermittelt"
      "Nicht übermittelt" ...
```

Class simData

To convert data from the `rkidata` format, the function `getRkiData()` can be used. The function generates data that can be used as `simmer` arrival events, one arrival is listed in each row. Each arrival includes the following information:

- age group (classification used by RKI)
- gender
- day, the infection was registered (in German: `Meldedatum`)
- region information (id of the state)
- region information (id of the county)
- age as a numerical value (1:1 correspondence with age group)

Note: As mentioned earlier, the CRAN package does not include the full RKI dataset. Only a sample is included as `rkidata` To perform real simulations, the user has to download the full RKI data set as described above.

```
rkiSimData <- getRkiData(babsim.hospital::rkidata)
%> getRkiData(): Found days with negative number of cases. Ignoring them.
str(rkiSimData)
%> 'data.frame':    3505596 obs. of  7 variables:
%>  $ Altersgruppe: chr  "A15-A34" "A35-A59" "A15-A34" "A35-A59" ...
%>  $ Geschlecht  : chr  "M" "W" "M" "W" ...
%>  $ Day         : Date, format: "2020-09-01" "2020-09-01" ...
%>  $ IdBundesland: int  1 1 1 1 1 1 1 1 1 1 ...
%>  $ IdLandkreis : int  1002 1002 1004 1004 1053 1053 1053 1053 1054 1056 ...
%>  $ time        : num  0 0 0 0 0 0 0 0 0 0 ...
%>  $ Age         : num  25 47 25 47 2 25 25 70 25 10 ...
```

Adding risk information to simData class data

The function `getRkiRisk(simData, para)` adds a numerical risk value, which is based on `Age` and `Geschlecht` to `simData` data. The parameters `RiskFactorA`, `RiskFactorB`, and `RiskMale` from `para` are used for the risk calculation.

```
para <- babsimHospitalPara()
print(para$RiskFactorA)
%> [1] 0.02048948
print(para$RiskFactorB)
%> [1] 0.01
print(para$RiskMale)
%> [1] 1.5
rkiRiskSimData <- getRkiRisk(rkiSimData, para)
str(rkiRiskSimData)
%> 'data.frame':    3505596 obs. of  8 variables:
%>  $ Altersgruppe: chr  "A15-A34" "A35-A59" "A15-A34" "A35-A59" ...
%>  $ Geschlecht  : chr  "M" "W" "M" "W" ...
%>  $ Day         : Date, format: "2020-09-01" "2020-09-01" ...
%>  $ IdBundesland: int  1 1 1 1 1 1 1 1 1 1 ...
%>  $ IdLandkreis : int  1002 1002 1004 1004 1053 1053 1053 1053 1054 1056 ...
%>  $ time        : num  0 0 0 0 0 0 0 0 0 0 ...
%>  $ Age         : num  25 47 25 47 2 25 25 70 25 10 ...
%>  $ Risk        : num  0.0395 0.0328 0.0395 0.0328 0.0314 ...
```

The Class arrivaldata

- The `babsim.hospital` simulator function `babsimHospital()`, which implements a `simmer` class, processes arrival events.
- An element of the `arrivaldata` class can be generrated as follows:

```
arrivalTimes <- data.frame(time = rkiRiskSimData$time, risk = rkiRiskSimData$Risk)
str(arrivalTimes)
%> 'data.frame':    3505596 obs. of  2 variables:
%>  $ time: num  0 0 0 0 0 0 0 0 0 0 ...
%>  $ risk: num  0.0395 0.0328 0.0395 0.0328 0.0314 ...
```

Optimization Details

Bounds

```
getParameterDataFrame()
```

```
bounds <- getBounds()
print(bounds)
%> $lower
%>  [1] 6.0e+00 7.0e+00 3.0e+00 3.0e+00 3.0e+00 5.0e+00 3.0e+00 3.0e+00 2.5e+01
%> [10] 1.7e+01 2.0e+00 1.0e+00 2.5e-01 5.0e-02 7.0e-02 5.0e-03 7.0e-02 1.0e-04
%> [19] 8.0e-02 2.5e-01 8.0e-02 5.0e-01 1.0e-06 2.0e+00 1.0e-06 1.0e-06 1.0e+00
%> [28] 2.0e+00 5.0e-01
%>
```

```
%> $upper
%>  [1] 14.0000 13.0000  7.0000  9.0000  7.0000  9.0000  5.0000  7.0000 35.0000
%> [10] 25.0000  5.0000  7.0000  2.0000  0.1500  0.1100  0.0200  0.1300  0.0020
%> [19]  0.1200  0.3500  0.1200  0.9000  0.0100  4.0000  1.1000  0.0625  2.0000
%> [28]  5.0000  0.7500
```

References

Appendix

Run Scripts

The following R scripts demonstrate, how optimizations runs for the Oberbergische Kreis (OBK), Koeln, and NRW can be started.

Note: runs take several minutes/hours.

Oberbergischer Kreis (OBK)

```
library("babsim.hospital")
library("SPOT")
library("simmer")

runoptDirect(
  expName = paste0("obk_", format(Sys.time(), "%Y_%b.%d_%H.%M_V"),
         utils::packageVersion("babsim.hospital"),"R"),
  region = 5374,
  rkiwerte = babsim.hospital::rkidata,
  icuwerte = babsim.hospital::icudata,
  TrainSimStartDate = Sys.Date() - 10*7, # 11*7, #10*7, # "2020-09-03",
  TrainFieldStartDate = Sys.Date() - 6*7, # 8*7, # "2020-10-03",
  #TestSimStartDate = Sys.Date() - 8*7, # 6*7 , #"2020-09-23",
  #TestFieldStartDate = Sys.Date() - 4*7, #"2020-10-23",
  Overlap = 0,
  seed = 101170,
  direct = TRUE,
  repeats = 1000,
  funEvals = 1000,
  funEvalsFactor = 0,
  size = 250,
  simrepeats = 10,
  parallel = TRUE,
  percCores = 0.9,
  icu = TRUE,
  icuWeights = c(1,1),
  verbosity=11,
  testRepeats = 10,
  tryOnTestSet = FALSE
)
```

Koeln (Cologne)

```
library("babsim.hospital")
library("SPOT")
library("simmer")

runoptDirect(
  expName = paste0("koeln_", format(Sys.time(), "%Y_%b.%d_%H.%M_V"),
        utils::packageVersion("babsim.hospital"),"R"),
  region = 5315,
  rkiwerte = babsim.hospital::rkidata,
  icuwerte = babsim.hospital::icudata,
  TrainSimStartDate = Sys.Date() - 10*7, # 10*7, # "2020-09-03",
  TrainFieldStartDate = Sys.Date() - 6*7, # "2020-10-03",
  # TestSimStartDate = Sys.Date() - 8*7, # 6*7 , #"2020-09-23",
  # TestFieldStartDate = Sys.Date() - 4*7, #"2020-10-23",
  Overlap = 0,
  seed = 101170,
  direct = TRUE,
  repeats = 1000,
  funEvals = 1000,
  funEvalsFactor = 0,
  size = 250,
  simrepeats = 10,
  parallel = TRUE,
  percCores = 0.9,
  icu = TRUE,
  icuWeights = c(1,1),
  verbosity=11,
  testRepeats = 10,
  tryOnTestSet = FALSE
)
```

NRW

```
library("babsim.hospital")
library("SPOT")
library("simmer")

runoptDirect(
  expName = paste0("nrw_", format(Sys.time(), "%Y_%b.%d_%H.%M_V"),
        utils::packageVersion("babsim.hospital"),"R"),
  region = 5374,
  rkiwerte = babsim.hospital::rkidata,
  icuwerte = babsim.hospital::icudata,
  TrainSimStartDate = Sys.Date() - 10*7, #10*7, # "2020-09-03",
  TrainFieldStartDate = Sys.Date() - 6*7, # "2020-10-03",
  # TestSimStartDate = Sys.Date() - 8*7, #6*7 , #"2020-09-23",
  # TestFieldStartDate = Sys.Date() - 4*7, #"2020-10-23",
  Overlap = 0,
  seed = 101170,
  direct = TRUE,
  repeats = 1000,
  funEvals = 1000,
  funEvalsFactor = 0,
  size = 250,
  simrepeats = 10,
  parallel = TRUE,
  percCores = 0.9,
```

```
  icu = TRUE,
  icuWeights = c(1,1),
  verbosity=11,
  testRepeats = 10,
  tryOnTestSet = FALSE
)
```