nosplit: Memory efficient aggregation of multistate event history data into occurence-exposure tables

Jonas Schöley

2021-02-11

Problem description

Reconstructing the health or labor market history of every person in a population register can easily produce a data set with hundreds of millions of rows, one for each time a person transitioned from one state to another. Aggregating this individual level data into time intervals is often a necessity both due to an interest in empirical occurrence-exposure rates and due to the prohibitive computational costs of fitting models to the individual level data set.

The state-of-the-art for aggregation of multistate life history data into age-period and cohort intervals is the "split-aggregate" method whereby first the individual level data-set is expanded into a single row per individual per time interval visited and then transition counts and state occupancy times per time interval are calculated from this expanded data set. The split step expands an already large data set even further and thus can be extremely costly in memory usage and processor time.

Solution

I present an episode-split free method to aggregate multi-state life-history data into time intervals. The "nosplit-aggregate" method first produces three summary tables from the unaltered individual level data set and then derives the interval and state specific risk sets, exposure times and transition counts via elementary calculations on the aggregated tables.

```r
# I've implemented the aggregation algorithm in dplyr
library(tidyverse)

# these packages implement the episode split method
# of aggregation and serve as benchmark
library(Epi)
library(popEpi)

# for speed comparison
library(microbenchmark)
```

```r
# Aggregate Transitions Counts and Occupancy Times
#
# Episode-split-free Risk-set and Exposure Time Calculation
# from Event History Data
#
# @param df
#   A data frame.
# @param t_in
#   Entry time into state.
# @param d_in
#   State being entered.
# @param t_out
#   Exit time from state.
# @param d_out
#   State being exited into.
# @param breaks
#   A numeric vector of break points for time-scale.
# @param wide
#   Output table in wide format (default=TRUE)?
#
# @return
#   A data frame with columns
#     orig: origin state
#     j:    age group index
#     x:    starting age of j
#     n:    width of j
#     Z:    number of entries into origin state during j
#     W:    number of exits from origin state during j
#     P:    population number in origin state at beginning of j
#     O:    total observation time of population visiting origin state in j
#     (if wide = FALSE)
#     dest: destination state
#     W_k:  number of exits from origin state to destination state during j
#     (if wide = TRUE)
#     to_*: number of exits from origin state to state * during j
NosplitAggregate <- function (
  df,
  t_in, d_in, t_out, d_out,
  breaks,
  wide = TRUE, drop0exp = TRUE,
  closed_left = TRUE
) {

  require(tidyverse)

  t_in = enquo(t_in); d_in = enquo(d_in);
  t_out= enquo(t_out); d_out = enquo(d_out)

  # total number of age intervals
  J_ = length(breaks)-1
  # index of age intervals
  j_ = 1:J_
  # width of age intervals
  n_j_ = diff(breaks)
  # unique origin states
  k_in_ = unique(pull(df, !!d_in))
  # unique destination states
  k_out_ = unique(pull(df, !!d_out))

  # find the index of an interval defined by
  # <breaks> each element in <x> is contained in
  # returns NA if x outside breaks
  FindIntervalJ <-
    function (x, breaks, cl = closed_left) {
      if (identical(cl, TRUE)) {
        # [a, b)
        right = FALSE; lowest = FALSE
      } else {
        # (a, b] with [a0, b0]
        right = TRUE; lowest = TRUE
      }
      .bincode(
        x = x, breaks = breaks,
        right = right, include.lowest = lowest
      )
    }

  # 1. Aggregation

  # tabulate exits by age, origin and destination state
  W_k_tab <-
    df %>%
    select(t_out = !!t_out, d_in = !!d_in, d_out = !!d_out) %>%
    mutate(
      # add age interval index to each exit
      j = FindIntervalJ(pull(., t_out), breaks, closed_left),
    ) %>%
    # for each observed combination of
    # age and
    # origin state and
    # destination state...
    group_by(d_in, d_out, j) %>%
    summarise(
      # ...total number of exits
      W_k = n(),
      # total time lost in age due to exit
      Lw_k = sum(breaks[j+1]-t_out),
      .groups = 'drop'
    )

  # tabulate exits by age and origin state
  # based on prior tabulation on destination specific exits
  W_tab <-
    W_k_tab %>%
    # for each observed combination of
    # age and
    # origin state...
    group_by(j, d_in) %>%
    summarise(
      # ...total exits
      W = sum(W_k),
      # ...total time lost in interval due to exit
      Lw = sum(Lw_k),
      .groups = 'drop'
    ) %>%
    # add rows for missing combinations
    # of age interval and origin state
    complete(
      d_in = k_in_, j = j_,
      fill = list(W = 0, Lw = 0)
    )

  # tabulate entries by age and state entered into
  Z_tab <-
    df %>%
    select(d_in = !!d_in, t_in = !!t_in) %>%
    mutate(
      j = FindIntervalJ(pull(., t_in), breaks, closed_left),
    ) %>%
    group_by(j, d_in) %>%
    summarise(
      # ...total entries
      Z = n(),
      # ...total entries right at start of interval
      Z0 = sum(t_in==breaks[j]),
      # ...total time lost in interval due to late-entry
      Lz = sum(t_in-breaks[j]),
      .groups = 'drop'
    ) %>%
    complete(
      d_in = k_in_, j = j_,
      fill = list(Z = 0, Z0 = 0, Lz = 0)
    )

  # tabulate concurrent entries and exits by interval
  ZW_tab <-
    df %>%
    select(t_in = !!t_in, t_out = !!t_out, d_in = !!d_in) %>%
    # aggregate individual level entry
    # and exit times into predefined age groups
    mutate(
      # add interval index to each entry
      j = FindIntervalJ(pull(., t_in), breaks, closed_left),
      # are entries and exits in same interval?
      zw = j == FindIntervalJ(pull(., t_out), breaks)
    ) %>%
    # for each combination of
    # state and
    # interval
    group_by(d_in, j) %>%
    summarise(
      # ...total concurrent entries and exits
      # there may be NAs in logic vector <zw> when
      # and entry or exit falls outside the range
      # of all intervals. as those cases don't have to
      # be counted na.rm=TRUE is applied
      ZW = sum(zw, na.rm = TRUE),
      .groups = 'drop'
    ) %>%
    complete(
      d_in = k_in_, j = j_,
      fill = list(ZW = 0)
    )

  # 2. Determine risk-sets and exposure times

  # exit counts for all possible combinations
  # of origin state, destination state and
  # age interval
  # intrastate transitions are 0 now
  # but are added later
  W_k_tab_complete <-
    W_k_tab %>%
    select(-Lw_k) %>%
    complete(
      d_in = k_in_, d_out = k_out_, j = j_,
      fill = list(W_k = 0)
    )

  # occurence-exposure table
  oe_tab <-
    bind_cols(W_tab, Z_tab[,-(1:2)], ZW_tab[,-(1:2)]) %>%
    mutate(
      x = breaks[j],
      n = n_j_[j]
    ) %>%
    # for each entry state...
    group_by(d_in) %>%
    mutate(
      # number of observations entering j via j-1
      # R_(j+1) = R_j + Z_j - W_j
      R = c(0, head(cumsum(Z) - cumsum(W), -1)),
      # population at risk at x_j
      P = R + Z0,
      # number of observations in j that did neither start
      # nor end during j
      Q = R - W + ZW,
      # number of observations entering j
      # that do not end during j
      U = Z - ZW,
      # total observation time during j
      O = Q*n + (Z + W - ZW)*n - Lz - Lw,
      # number of intrastate transitions
      I = Q + U,
    ) %>%
    ungroup() %>%
    left_join(W_k_tab_complete, by = c('d_in', 'j')) %>%
    # intrastate transitions
    mutate(
```

```r
      W_k = ifelse(d_in == d_out, I, W_k)
    ) %>%
    select(orig = d_in, dest = d_out, j, x, n, Z, W, P, O, W_k)

  # drop intervals with 0 exposure
  if (identical(drop0exp, TRUE)) {
  oe_tab <-
    oe_tab %>%
    filter(O > 0)
  }

  # convert to wide format
  if (identical(wide, TRUE)) {
    oe_tab <-
      oe_tab %>%
      mutate(dest = paste0('to_', dest)) %>%
      spread(key = dest, value = W_k)
  }


  return(oe_tab)

}
```

Demonstration

We demonstrate how to use the `NosplitAggregate()` function to aggregate individual level survival data across a range of situations. We validate the correctness and performance of `NosplitAggregate()` against the aggregation routines implemented in the `Epi` and `PopEpi` packages, which require the data to be episode-split prior to aggregation.

Left-truncation and right-censoring

`NosplitAggregate()` can be used to quickly estimate cohort life tables from survival data subject to left truncation and right censoring.

First we simulate 10 million survival times drawn from a Weibull distribution which are subject to random left truncation and right censoring.

```r
# 1 million simulated survival times
N = 1e6

# "true" survival and hazard curves corresponding to simulated data
true_dat <-
  tribble(
    ~orig, ~dest, ~dist, ~shape, ~scale,
    'alive', 'censored', 'gamma', 0.5, 20,
    'alive', 'dead', 'weibull', 1.5, 20,
  ) %>%
  group_by(orig, dest) %>%
  group_modify(
    ~ tibble(
      x = seq(0, 100, 0.1),
      s = 1-get(paste0('p',.$dist))(x, shape = .$shape, scale = .$scale),
      h = get(paste0('d',.$dist))(x, shape = .$shape, scale = .$scale)/s
    )
  )

sim_dat <-
  tibble(
    # event time
    t_event = rweibull(N, shape = 1.5, scale = 20),
    # censoring time
    t_cens = rgamma(N, shape = 0.5, scale = 20),
    # entry time
    t_in = rexp(N, 0.1),
    # entry state
    d_in = 'alive',
    # exit state
    d_out = ifelse(t_event < t_cens, 'dead', 'censored'),
    # survival time
    t_out = ifelse(d_out == 'dead', t_event, t_cens)
  ) %>%
  select(t_in, d_in, t_out, d_out) %>%
  # left truncation
  filter(
    t_out >= t_in
  )

sim_dat
```

```
## # A tibble: 361,566 x 4
##     t_in d_in  t_out d_out
##    <dbl> <chr> <dbl> <chr>
##  1 3.43  alive  4.70 dead
##  2 0.692 alive 16.7  dead
##  3 2.72  alive  9.60 censored
##  4 1.33  alive 30.1  dead
##  5 3.34  alive  9.59 censored
##  6 0.282 alive  4.81 censored
##  7 1.39  alive 13.9  dead
##  8 8.49  alive  9.60 censored
##  9 5.15  alive 39.2  censored
## 10 1.69  alive  8.52 censored
## # ... with 361,556 more rows
```

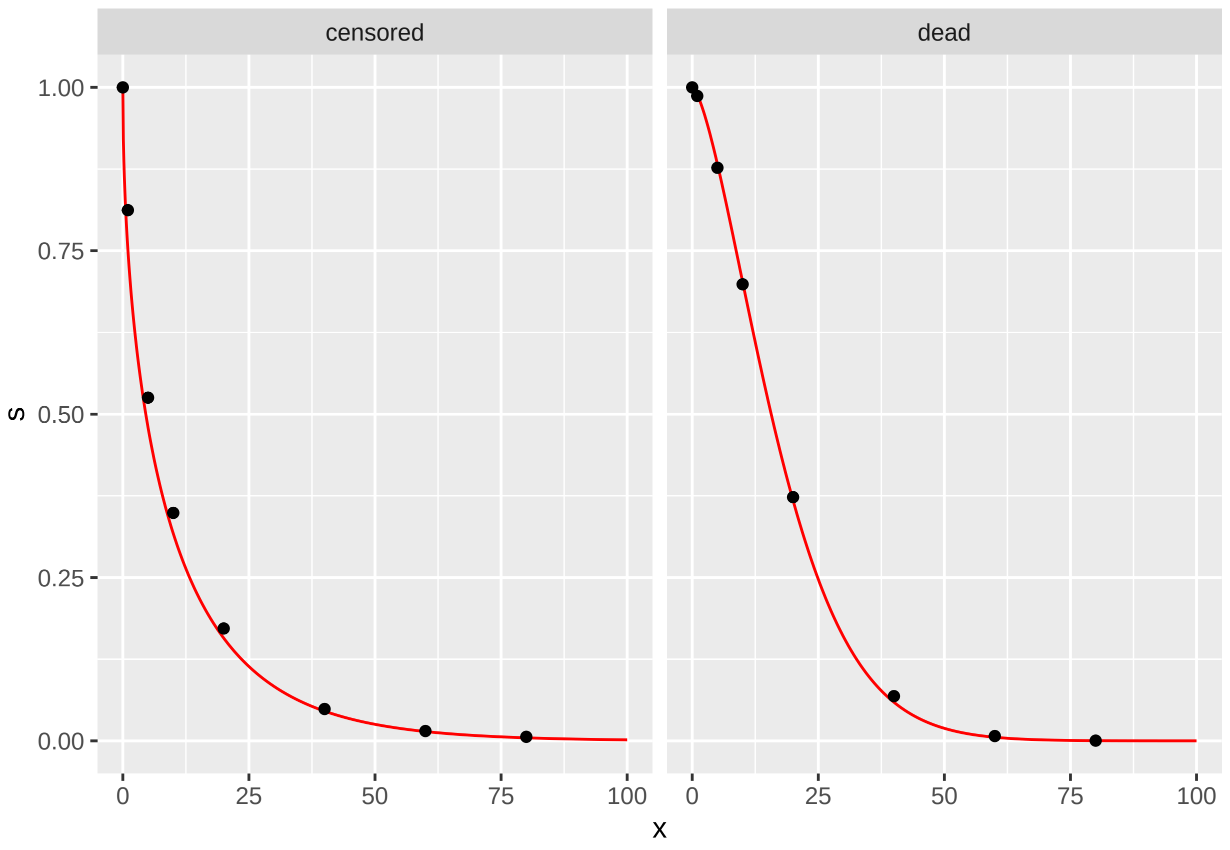
Using the NosplitAggregate() function we calculate a single-decrement cohort life-table.

```
# aggregate individual level transitions
lt <-
  NosplitAggregate(
    sim_dat, t_in, d_in, t_out, d_out,
    breaks = c(0, 1, 5, 10, seq(20, 100, 20)),
    wide = FALSE
  )
lt
```

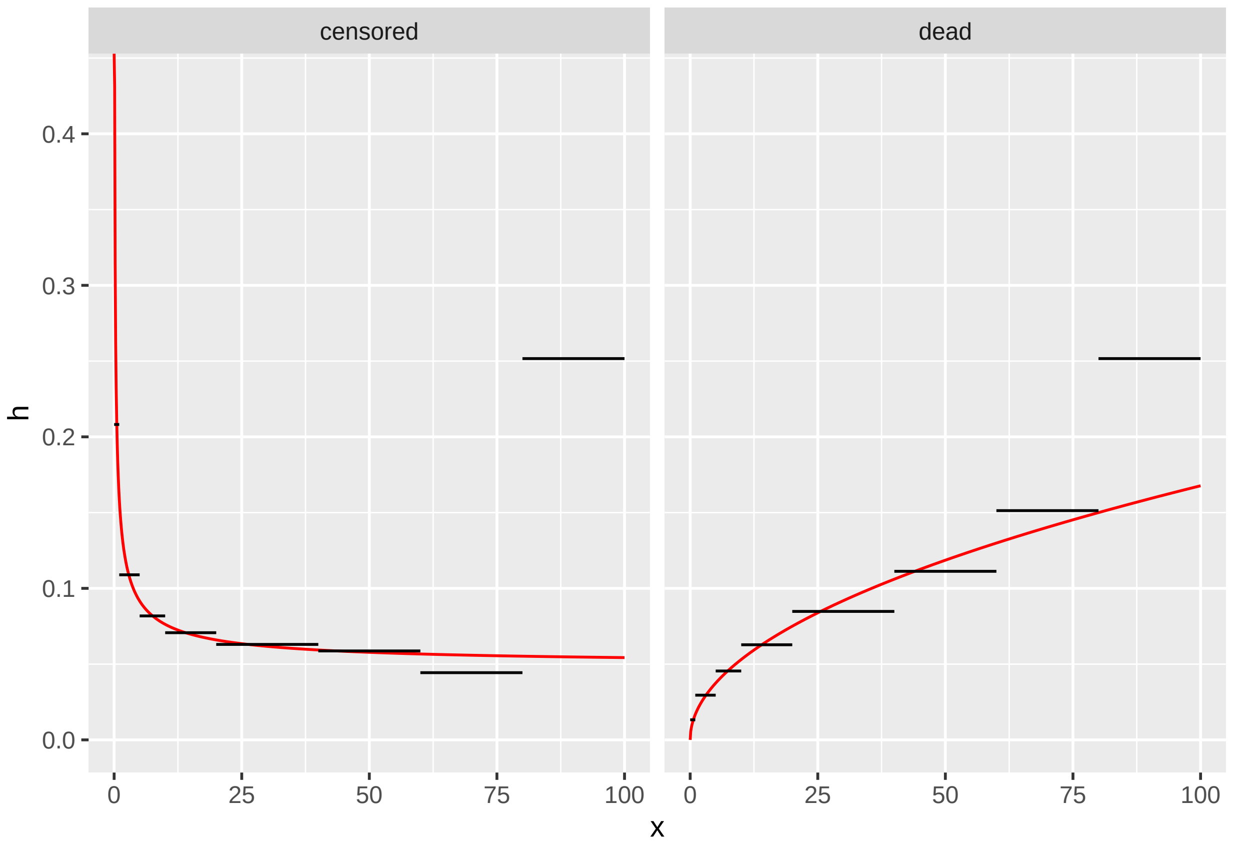
```
## # A tibble: 16 x 10
##    orig  dest         j     x     n      Z      W      P        O   W_k
##    <chr> <chr>    <int> <dbl> <dbl>  <dbl>  <dbl>  <dbl>    <dbl> <dbl>
##  1 alive censored     1     0     1  79122   8520      0  38481.   8010
##  2 alive dead         1     0     1  79122   8520      0  38481.    510
##  3 alive censored     2     1     4 171101  74878  70602 540850.  58921
##  4 alive dead         2     1     4 171101  74878  70602 540850.  15957
##  5 alive censored     3     5     5  76231 101684 166825 798897.  65364
##  6 alive dead         3     5     5  76231 101684 166825 798897.  36320
##  7 alive censored     4    10    10  31898 122978 141372 921193.  65152
##  8 alive dead         4    10    10  31898 122978 141372 921193.  57826
##  9 alive censored     5    20    20   3198  50884  50292 344320.  21692
## 10 alive dead         5    20    20   3198  50884  50292 344320.  29192
## 11 alive censored     6    40    20     16   2545   2606  14974.    879
## 12 alive dead         6    40    20     16   2545   2606  14974.   1666
## 13 alive censored     7    60    20      0     75     77    383.     17
## 14 alive dead         7    60    20      0     75     77    383.     58
## 15 alive censored     8    80    20      0      2      2      3.97    1
## 16 alive dead         8    80    20      0      2      2      3.97    1
```

Using the piecewise-exponential assumption, we can estimate the original hazard and survivorship functions from the aggregated counts. Note that estimated hazards in the highest age group can deviate quite a bit from the true values, due to low sample sizes.

```
# estimate true versus estimated survival and hazard from
# aggregated counts, assuming a piecewise exponential model
lt_pwe <-
  lt %>%
  group_by(dest) %>%
  mutate(
    m = W_k/O,
    p = exp(-m*n),
    l = c(1, cumprod(head(p, -1)))
  )

# true vs. estimated survival
ggplot(lt_pwe) +
  geom_line(
    aes(x = x, y = s, group = dest),
    data = true_dat, color = 'red'
  ) +
  geom_point(aes(x = x, y = l)) +
  facet_wrap(~dest)
```



```r
# true vs. estimated hazard
ggplot(lt_pwe) +
  geom_line(
    aes(x = x, y = h, group = dest),
    data = true_dat, color = 'red'
  ) +
  geom_segment(aes(x = x, xend = x+n, y = m, yend = m)) +
  facet_wrap(~dest)
```



Multiple decrements

Here we simulate an independent competing risks scenario with 4 ways of exiting the population under observation (including censoring). The data are subject to left-truncation.

```
N = 1e6

# true cause specific hazards and survival
true_dat <-
  tribble(
    ~orig, ~dest, ~dist, ~shape, ~scale,
    'alive', 0, 'gamma', 0.5, 20,
    'alive', 1, 'weibull', 1.5, 20,
    'alive', 2, 'weibull', 1.5, 15,
    'alive', 3, 'weibull', 0.9, 20
  ) %>%
  group_by(orig, dest) %>%
  group_modify(
    ~ tibble(
      x = seq(0, 100, 0.1),
      s = 1-get(paste0('p',.$dist))(x, shape = .$shape, scale = .$scale),
      h = get(paste0('d',.$dist))(x, shape = .$shape, scale = .$scale)/s
    )
  )

sim_dat <-
```

```
  tibble(
    # censoring time
    t0 = rgamma(N, shape = 0.5, scale = 20),
    # time events 1:3
    t1 = rweibull(N, shape = 1.5, scale = 20),
    t2 = rweibull(N, shape = 1.5, scale = 15),
    t3 = rweibull(N, shape = 0.9, scale = 20),
    # entry time
    t_in = rexp(N, 0.1),
    # entry state
    d_in = 'alive',
    # exit state
    d_out = apply(cbind(t0, t1, t2, t3), 1, which.min)-1,
    # survival time
    t_out = pmin.int(t0, t1, t2, t3)
  ) %>%
  # some left truncation
  filter(
    t_out >= t_in
  ) %>%
  select(t_in, d_in, t_out, d_out)

sim_dat
```

```
## # A tibble: 255,967 x 4
##      t_in d_in  t_out d_out
##     <dbl> <chr> <dbl> <dbl>
##  1  0.803 alive  3.90     1
##  2  7.52  alive 10.3      2
##  3  0.184 alive  8.25     2
##  4  3.16  alive  5.13     1
##  5  0.776 alive  4.95     1
##  6  4.31  alive 11.4      1
##  7 11.0   alive 12.7      1
##  8  8.85  alive 17.1      1
##  9  1.97  alive 21.4      0
## 10  6.56  alive  6.98     2
## # ... with 255,957 more rows
```

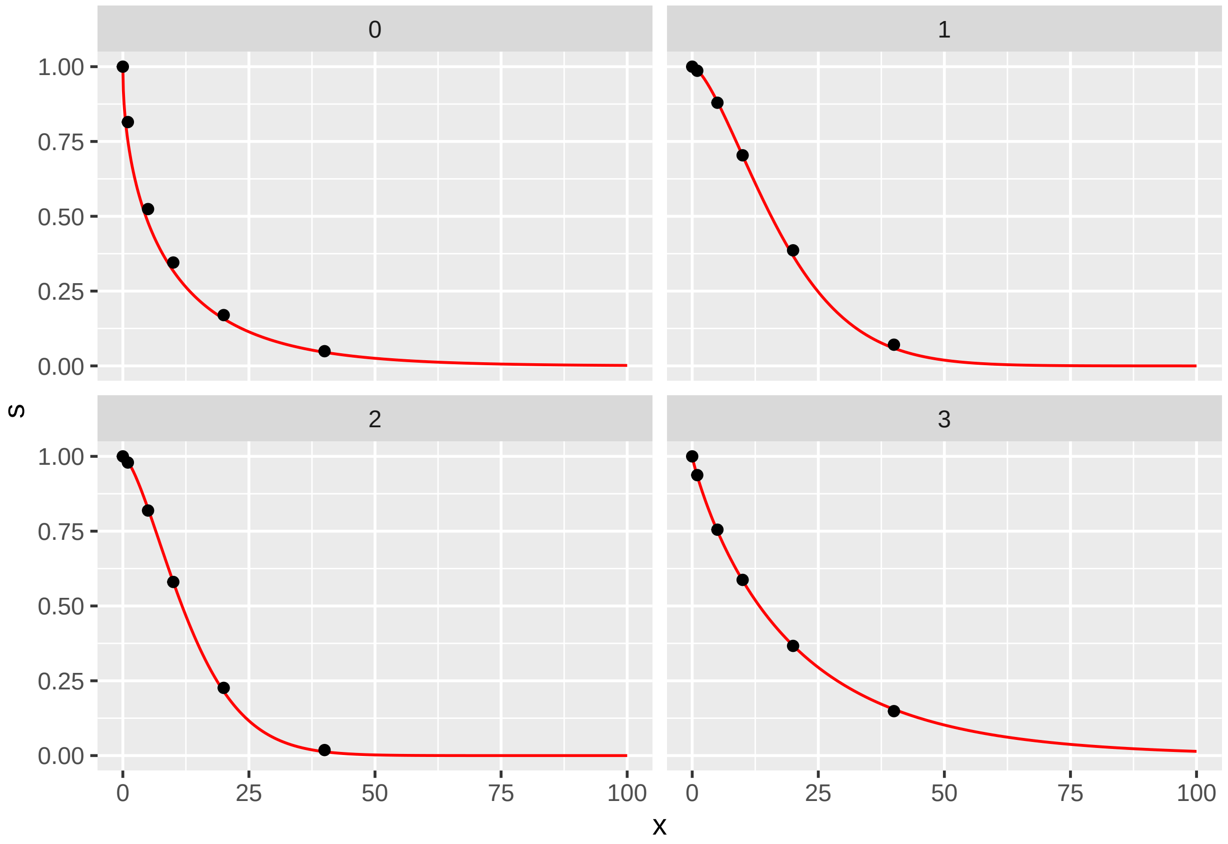
Once again we aggregate the data using `NosplitAggregate()` and approximate the true survival curve using the piecewise-exponential assumption.

```
# aggregate individual level transitions
lt <-
  NosplitAggregate(
    sim_dat, t_in, d_in, t_out, d_out,
    breaks = c(0, 1, 5, 10, seq(20, 100, 20)),
    wide = FALSE
  )
lt
```

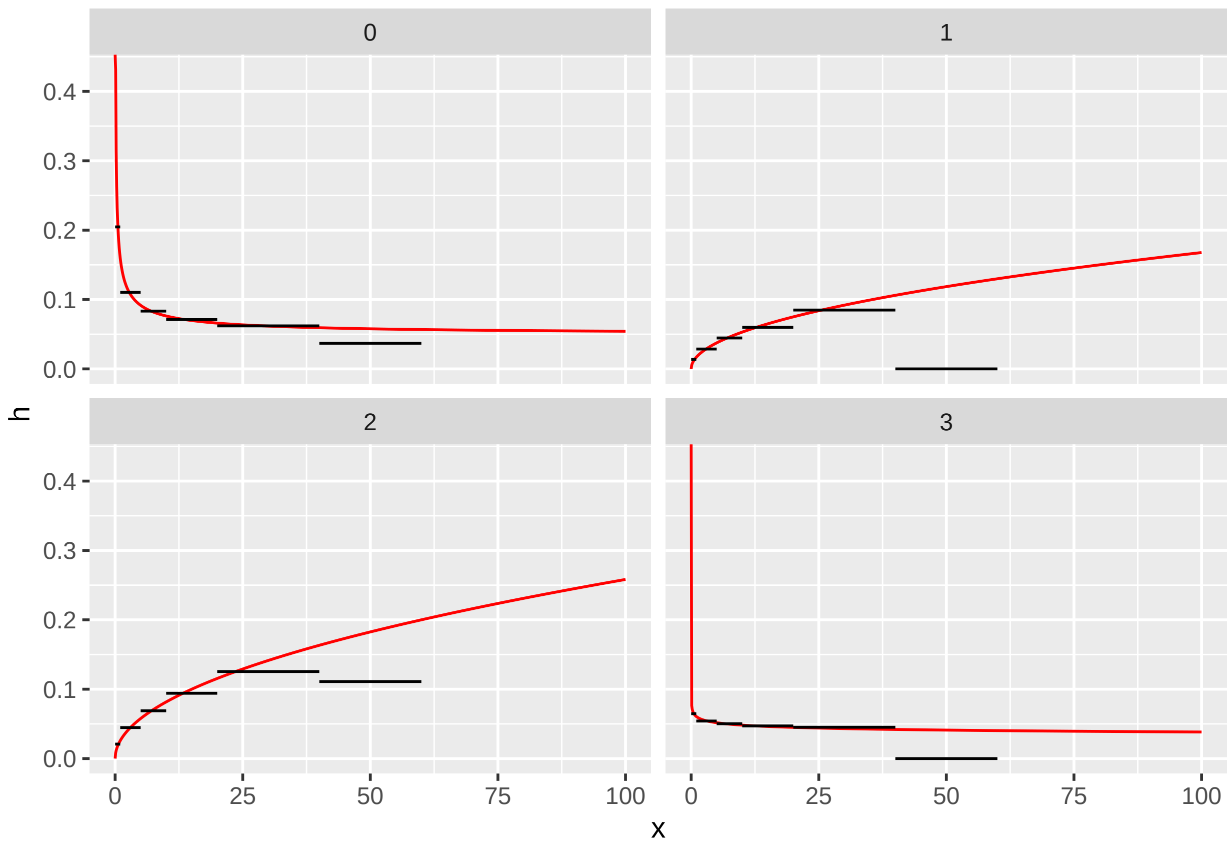
```
## # A tibble: 24 x 10
##    orig   dest     j     x     n      Z     W      P       O   W_k
##    <chr> <dbl> <int> <dbl> <dbl>  <dbl> <dbl>  <dbl>   <dbl> <dbl>
##  1 alive     0     1     0     1  76021 11029      0  36269.  7424
##   2 alive     1     1     0     1  76021 11029      0  36269.   502
##   3 alive     2     1     0     1  76021 11029      0  36269.   757
##   4 alive     3     1     0     1  76021 11029      0  36269.  2346
##   5 alive     0     2     1     4 134286 96031  64992 403758. 44553
##   6 alive     1     2     1     4 134286 96031  64992 403758. 11568
##   7 alive     2     2     1     4 134286 96031  64992 403758. 18055
##   8 alive     3     2     1     4 134286 96031  64992 403758. 21855
##   9 alive     0     3     5     5  38131 93460 103247 378285. 31526
##  10 alive     1     3     5     5  38131 93460 103247 378285. 16859
## # ... with 14 more rows
```

Here are the estimated hazards and survival curves against their true counterparts.

```
# estimate true versus estimated survival and hazard from
# aggregated counts, assuming a piecewise exponential model
lt_pwe <-
  lt %>%
  group_by(dest) %>%
  mutate(
    m = W_k/O,
    p = exp(-m*n),
    l = c(1, cumprod(head(p, -1)))
  )

# true vs. estimated survival
ggplot(lt_pwe) +
  geom_line(
    aes(x = x, y = s, group = dest),
    data = true_dat, color = 'red'
  ) +
  geom_point(aes(x = x, y = l)) +
  facet_wrap(~dest)
```



```
# true vs. estimated hazard
ggplot(lt_pwe) +
  geom_line(
    aes(x = x, y = h, group = dest),
    data = true_dat, color = 'red'
  ) +
  geom_segment(aes(x = x, xend = x+n, y = m, yend = m)) +
  facet_wrap(~dest)
```



Multiple entry-states

Same as before (independent competing risks, left-truncation, right-censoring), but now we simulate data with two different entry states and survival times depending on the entry state.

```r
N = 1e6

# true cause and entry state specific hazards and survival
true_dat <-
  tribble(
    ~orig, ~dest, ~dist, ~shape, ~scale,
    'low', 0, 'gamma', 0.5, 20,
    'low', 1, 'weibull', 1.5, 20,
    'low', 2, 'weibull', 1.5, 10,
    'low', 3, 'weibull', 0.9, 20,
    'high', 0, 'gamma', 0.5, 20,
    'high', 1, 'weibull', 1.5, 30,
    'high', 2, 'weibull', 1.5, 15,
    'high', 3, 'weibull', 0.9, 25
  ) %>%
  group_by(orig, dest) %>%
  group_modify(
    ~ tibble(
      x = seq(0, 100, 0.1),
      s = 1-get(paste0('p',.$dist))(x, shape = .$shape, scale = .$scale),
      h = get(paste0('d',.$dist))(x, shape = .$shape, scale = .$scale)/s
    )
  )

sim_dat <-
  tibble(
    # entry state
    d_in = sample(c('low', 'high'), N, replace = TRUE)
  ) %>%
  group_by(d_in) %>%
  mutate(
    # censoring time
    t0 = rgamma(n(), shape = 0.5, scale = 20),
    # time events 1:3
    t1 = rweibull(n(), shape = 1.5,
                  scale = if(d_in[1] == 'low') 20 else 30),
    t2 = rweibull(n(), shape = 1.5,
                  scale = if(d_in[1] == 'low') 10 else 15),
    t3 = rweibull(n(), shape = 0.9,
                  scale = if(d_in[1] == 'low') 20 else 25),
    # entry time
    t_in = rexp(n(), 0.1),
    # exit state
    d_out = apply(cbind(t0, t1, t2, t3), 1, which.min)-1,
    # survival time
    t_out = pmin.int(t0, t1, t2, t3)
  ) %>%
  ungroup() %>%
  # some left truncation
  filter(
    t_out >= t_in
  ) %>%
  select(t_in, d_in, t_out, d_out)

sim_dat
```

```
## # A tibble: 254,203 x 4
##      t_in d_in  t_out d_out
##     <dbl> <chr> <dbl> <dbl>
##  1  0.777 low    1.39     0
##  2  1.76  low    7.93     0
##  3  3.58  low    6.76     2
##  4 10.1   high  13.7      2
##  5  1.91  high  10.4      2
##  6  0.189 low    4.76     0
##  7  2.00  high   6.03     0
##  8  2.31  low    3.36     0
##  9  4.67  low    5.52     1
## 10  0.825 low    2.81     0
## # ... with 254,193 more rows
```

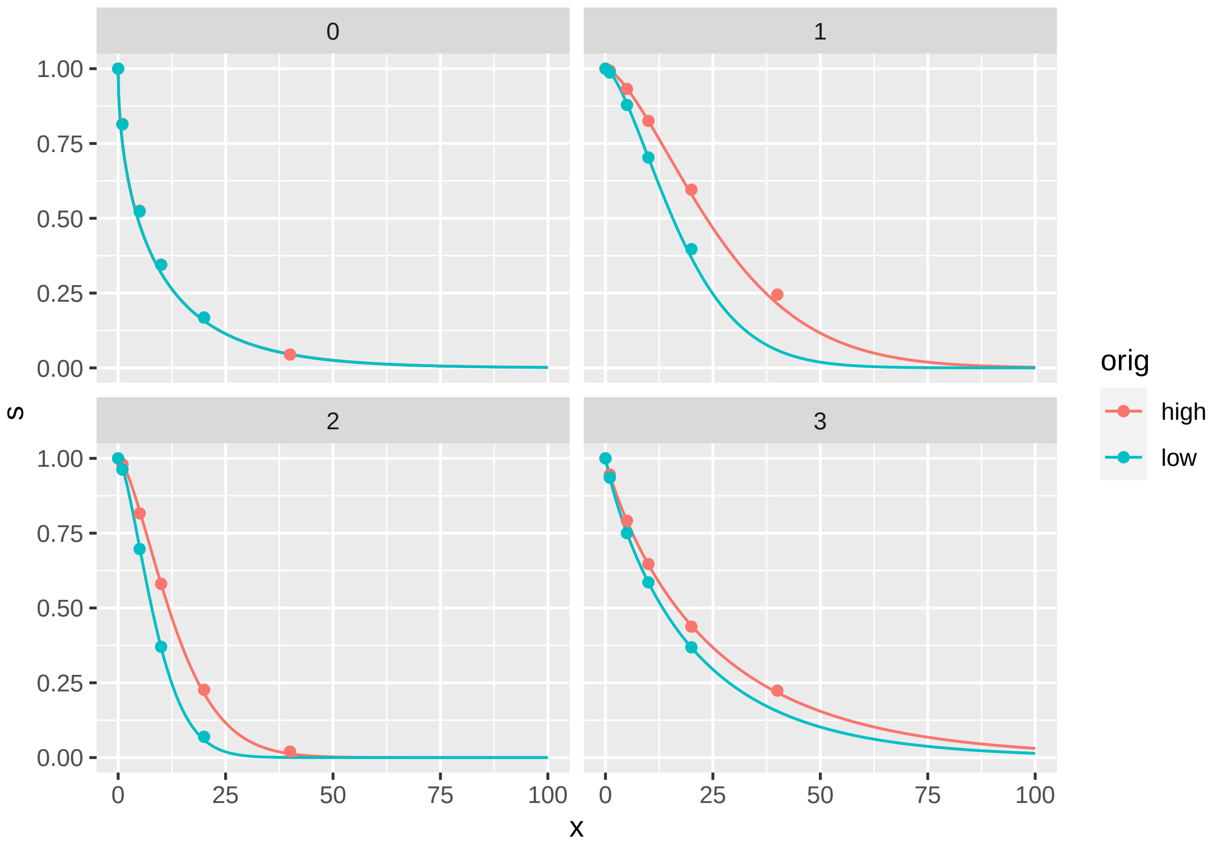
Aggregate…

```
# aggregate individual level transitions
lt <-
  NosplitAggregate(
    sim_dat, t_in, d_in, t_out, d_out,
    breaks = c(0, 1, 5, 10, seq(20, 100, 20)),
    wide = FALSE
  )
lt
```

```
## # A tibble: 44 x 10
##    orig   dest     j     x     n     Z     W     P       O   W_k
##    <chr> <dbl> <int> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl>
##  1 high      0     1     0     1 38567  5322     0  18590.  3795
##  2 high      1     1     0     1 38567  5322     0  18590.   120
##  3 high      2     1     0     1 38567  5322     0  18590.   374
##  4 high      3     1     0     1 38567  5322     0  18590.  1033
##  5 high      0     2     1     4 70810 46500 33245 215022. 23657
##  6 high      1     2     1     4 70810 46500 33245 215022.  3425
##  7 high      2     2     1     4 70810 46500 33245 215022.  9837
##  8 high      3     2     1     4 70810 46500 33245 215022.  9581
##  9 high      0     3     5     5 22331 48758 57555 224735. 18912
## 10 high      1     3     5     5 22331 48758 57555 224735.  5470
## # ... with 34 more rows
```

… and estimate survival from the aggregated data via PWE under assumption of independent decrements.

```
# estimated versus true survival
lt %>%
  group_by(orig, dest) %>%
  mutate(
    m = W_k/O,
    p = exp(-m*n),
    l = c(1, cumprod(head(p, -1)))
  ) %>%
  ggplot() +
  geom_line(
    aes(x = x, y = s, color = orig),
    data = true_dat
  ) +
  geom_point(
    aes(
      x = x, y = l, color = orig
    )
  ) +
  facet_wrap(
    ~dest
  )
```



Full multi-state

Simulating true multi-state data where the number of transitions is not fixed is not trivial and I won't implement it here. Instead I'll use a small example dataset – `nlog98` – on life histories in the Netherlands, aggregate it with my algorithm and compare the results to the episode-split approach implemented in the `Epi` and `popEpi` packages.

```
nlog98 <- read_csv('/home/jon/lucile/share/Dropbox/sci/2019-08-nosplit/dat/nlog98.csv')
```

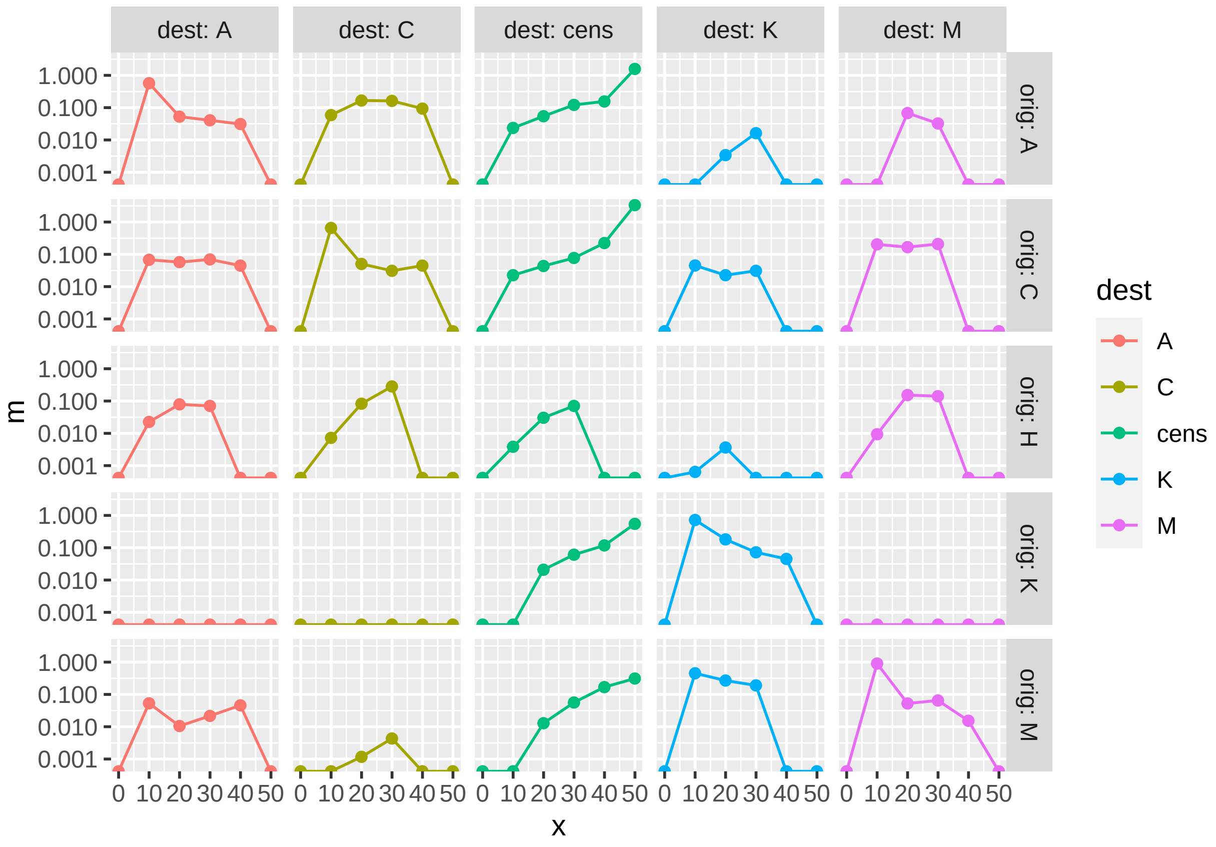
The individual level data in `nlog98` specifies origin `OR` and destination `DES` states and associated ages at entry `Tstarta` and exit `Tstopa`. As a second timescale the year at entry is given `per`. The `ID` variable identifies individuals.

The events and implied exposure times can easily be aggregated using `NosplitAggregate()`.

```
lt_nosplit <-
  NosplitAggregate(
    nlog98,
    t_in = Tstarta, t_out = Tstopa,
    d_in = OR, d_out = DES,
    breaks = seq(0, 60, 9.99),
    wide = FALSE, drop0exp = FALSE
  )
```

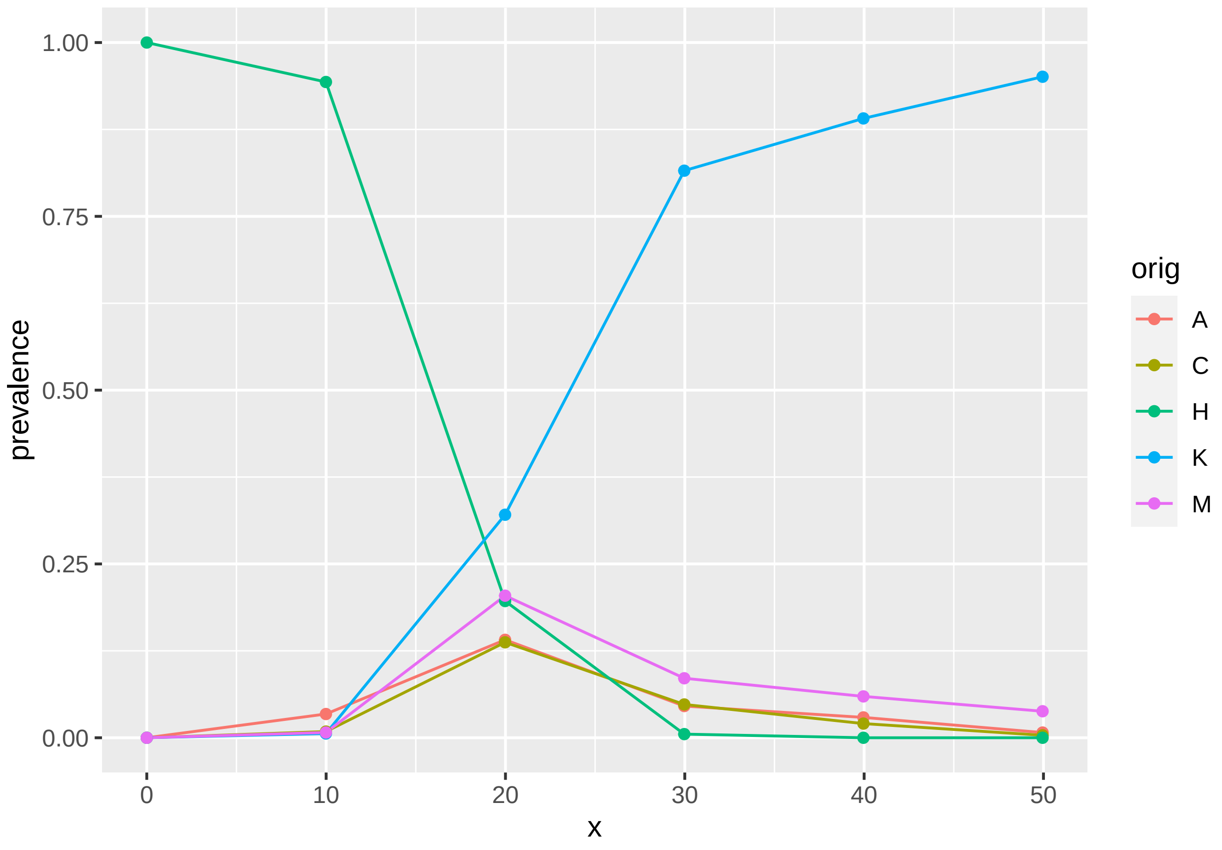
From the aggregated data we estimate the state to state transition rates within an age group. In the multi-state case the calculation of survival curves is more complicated compared to the simple independent risks cases before and we won't demonstrate it here.

```
# transition rates
lt_nosplit %>%
  mutate(
    m = ifelse(O==0, 0, W_k/O)
  ) %>%
  ggplot(aes(x = x, y = m, color = dest)) +
  geom_point() +
  geom_line() +
  facet_grid(orig~dest, labeller = 'label_both') +
  scale_y_log10()
```



We can easily calculate the state distribution by age.

```
# state prevalence by age
lt_nosplit %>%
  spread(key = dest, value = W_k) %>%
  group_by(j) %>%
  mutate(
    prevalence = O/sum(O)
  ) %>%
  ggplot(aes(x = x, y = prevalence, color = orig)) +
  geom_line() +
  geom_point()
```



Validate against popEpi

Below is an implementation of a survival time aggregation function based upon the functionality provided by the `Epi` and `popEpi` packages. It requires an episode split of the data (`splitLexisDT`) prior to aggregation (`aggre`).

```r
# Aggregate transitions counts and state occupancy times over time intervals
PopEpiAggregate <- function (df, t_in, t_out, d_in, d_out, breaks) {
  require(Epi); require(popEpi); require(tidyverse)
  v <- enquos(t_in=t_in, t_out=t_out, d_in=d_in, d_out=d_out)
  Lexis(
    entry = list(x = pull(df, !!v$t_in)),
    exit = list(x = pull(df, !!v$t_out)),
    entry.status = pull(df, !!v$d_in),
    exit.status = pull(df, !!v$d_out),
    tol = 0
  ) %>%
    splitLexisDT(breaks = breaks, timeScale = 'x') %>%
    aggre(by = list(orig = lex.Cst, x = x), type = 'unique')
}
```

Aggregate `nlog98` via the `nosplit` and via the `popEpi` approach.

```r
microbenchmark(list = alist(
  nosplit = {
  lt_nosplit <-
    NosplitAggregate(
      nlog98,
      t_in = Tstarta, t_out = Tstopa,
      d_in = OR, d_out = DES,
      breaks = seq(0, 60, 10),
      drop0exp = TRUE,
      closed_left = FALSE
    )
  },
  popepi = {
      lt_epi <-
    PopEpiAggregate(
      nlog98,
      t_in = Tstarta, t_out = Tstopa,
      d_in = OR, d_out = DES,
      breaks = seq(0, 60, 10)
    )
  }
), times = 10)
```

```
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
```

```
## Unit: milliseconds
##     expr      min       lq     mean   median        uq      max neval
##  nosplit 88.32273 91.12019 99.08249 96.65483 105.83237 115.1021    10
##   popepi 36.20660 40.96592 47.47249 44.96277  50.25041  79.5743    10
```

```
lt_nosplit
```

```
## # A tibble: 24 x 13
##    orig      j     x     n     Z     W     P       O  to_A  to_C to_cens  to_K
##    <chr> <int> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl>   <dbl> <dbl>
##  1 A         2    10    10   113    16     0 172.       97    11       4     0
##  2 A         3    20    10   104   171    97 589.       30    97      33     2
##  3 A         4    30    10    16    40    30 123.        6    20      14     2
##  4 A         5    40    10     3     8     6  32         1     3       5     0
##  5 A         6    50    10     0     1     1   0.583     0     0       1     0
##  6 C         2    10    10    46    15     0  44.8       3    31       1     2
##  7 C         3    20    10   167   169    31 575.       33    29      25    13
##  8 C         4    30    10    25    50    29 129.        9     4      10     4
##  9 C         5    40    10     3     6     4  22.2       1     1       5     0
## 10 C         6    50    10     0     1     1   0.25      0     0       1     0
## # ... with 14 more rows, and 1 more variable: to_M <dbl>
```

```
lt_epi
```

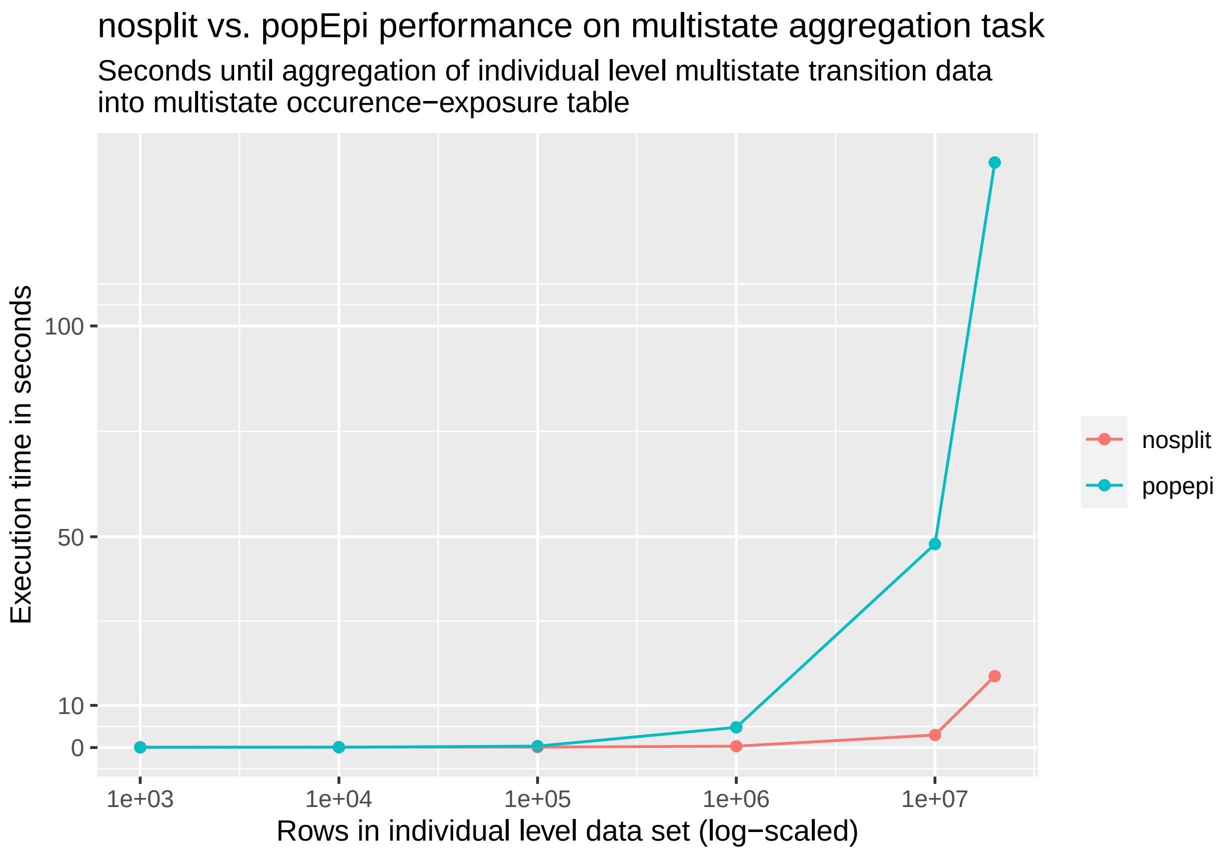
```
##     orig  x     pyrs at.risk fromAtoC fromAtocens fromAtoK fromAtoM fromCtoA
##  1:    A 10  171.910       0       11           4        0        1        0
##  2:    A 20  588.833      97       97          33        2       39        0
##  3:    A 30  122.581      30       20          14        2        4        0
##  4:    A 40   32.000       6        3           5        0        0        0
##  5:    A 50    0.583       1        0           1        0        0        0
##  6:    C 10   44.836       0        0           0        0        0        3
##  7:    C 20  575.004      31        0           0        0        0       33
##  8:    C 30  128.748      29        0           0        0        0        9
##  9:    C 40   22.250       4        0           0        0        0        1
## 10:    C 50    0.250       1        0           0        0        0        0
## 11:    H  0 5000.000     500        0           0        0        0        0
## 12:    H 10 4694.591     500        0           0        0        0        0
## 13:    H 20  817.582     291        0           0        0        0        0
## 14:    H 30   13.917       7        0           0        0        0        0
## 15:    K 10   30.918       0        0           0        0        0        0
## 16:    K 20 1349.915      22        0           0        0        0        0
## 17:    K 30 2203.916     244        0           0        0        0        0
## 18:    K 40  973.669     159        0           0        0        0        0
## 19:    K 50   78.006      44        0           0        0        0        0
## 20:    M 10   38.329       0        0           0        0        0        0
## 21:    M 20  856.251      36        0           0        0        0        0
## 22:    M 30  230.585      45        0           0        0        0        0
## 23:    M 40   64.749      14        0           0        0        0        0
## 24:    M 50    3.167       1        0           0        0        0        0
##     orig  x     pyrs at.risk fromAtoC fromAtocens fromAtoK fromAtoM fromCtoA
##     fromCtoC fromCtocens fromCtoK fromCtoM fromHtoA fromHtoC fromHtocens
##  1:        0           0        0        0        0        0           0
##  2:        0           0        0        0        0        0           0
##  3:        0           0        0        0        0        0           0
##  4:        0           0        0        0        0        0           0
##  5:        0           0        0        0        0        0           0
##  6:        0           1        2        9        0        0           0
##  7:        2          25       13       96        0        0           0
##  8:        0          10        4       27        0        0           0
##  9:        0           5        0        0        0        0           0
## 10:        0           1        0        0        0        0           0
## 11:        0           0        0        0        0        0           0
## 12:        0           0        0        0      108       35          18
## 13:        0           0        0        0       63       67          25
## 14:        0           0        0        0        1        4           1
## 15:        0           0        0        0        0        0           0
## 16:        0           0        0        0        0        0           0
## 17:        0           0        0        0        0        0           0
## 18:        0           0        0        0        0        0           0
## 19:        0           0        0        0        0        0           0
## 20:        0           0        0        0        0        0           0
## 21:        0           0        0        0        0        0           0
## 22:        0           0        0        0        0        0           0
## 23:        0           0        0        0        0        0           0
## 24:        0           0        0        0        0        0           0
##     fromCtoC fromCtocens fromCtoK fromCtoM fromHtoA fromHtoC fromHtocens
##     fromHtoK fromHtoM fromKtocens fromMtoA fromMtoC fromMtocens fromMtoK
##  1:        0        0           0        0        0           0        0
##  2:        0        0           0        0        0           0        0
##  3:        0        0           0        0        0           0        0
##  4:        0        0           0        0        0           0        0
##  5:        0        0           0        0        0           0        0
##  6:        0        0           0        0        0           0        0
##  7:        0        0           0        0        0           0        0
##  8:        0        0           0        0        0           0        0
##  9:        0        0           0        0        0           0        0
## 10:        0        0           0        0        0           0        0
## 11:        0        0           0        0        0           0        0
## 12:        3       45           0        0        0           0        0
## 13:        3      126           0        0        0           0        0
## 14:        0        1           0        0        0           0        0
## 15:        0        0           0        0        0           0        0
## 16:        0        0          28        0        0           0        0
## 17:        0        0         134        0        0           0        0
## 18:        0        0         115        0        0           0        0
## 19:        0        0          44        0        0           0        0
## 20:        0        0           0        2        0           0       17
## 21:        0        0           0        9        1          11      231
## 22:        0        0           0        6        1          13       43
## 23:        0        0           0        2        0          11        0
## 24:        0        0           0        0        0           1        0
##     fromHtoK fromHtoM fromKtocens fromMtoA fromMtoC fromMtocens fromMtoK
```

According to the benchmark on this small data-set the performance of the nosplit- and the aggregate-split aggregation approach are comparable. The performance advantage of nosplit emerges with increasing size of the event-history data as demonstrated below with a data set consisting of 1000 copies of `nlog98`.

```r
library(glue)

nlog98_varying_length <-
  map(list(1, 1e1, 1e2, 1e3, 1e4, 2e4), ~{

    nlog98_first1000rows <- slice(nlog98, 1:1000)
    nlog98_extended <-
      vector(mode = 'list', length = .x) %>%
      map(~nlog98_first1000rows) %>%
      bind_rows()

    benchmark <- microbenchmark(list = alist(

      nosplit = {NosplitAggregate(
        nlog98_extended,
        t_in = Tstarta, t_out = Tstopa,
        d_in = OR, d_out = DES,
        breaks = seq(0, 60, 10),
        drop0exp = TRUE,
        closed_left = FALSE
      )},

      popepi = {PopEpiAggregate(
        nlog98_extended,
        t_in = Tstarta, t_out = Tstopa,
        d_in = OR, d_out = DES,
        breaks = seq(0, 60, 10)
      )}

    ), times = 1, unit = 's') %>% summary()

    benchmark$nrow <- .x*1000

    return(benchmark)

  }) %>%
  bind_rows()
```

```
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  A C H K M cens
```

```r
nlog98_varying_length %>%
  ggplot(aes(x = nrow, y = median, color = expr)) +
  geom_point() +
  geom_line() +
  scale_y_continuous(breaks = c(0, 10, 50, 100, 150)) +
  scale_x_continuous(trans = 'log10') +
  labs(
    title = 'nosplit vs. popEpi performance on multistate aggregation task',
    subtitle = 'Seconds until aggregation of individual level multistate transition data\ninto multista
    y = 'Execution time in seconds',
    x = 'Rows in individual level data set (log-scaled)',
    color = NULL
  )
```



Extreme performance gains of `nosplit` over the `PopEpi` approach can be observed once the data becomes so large that the main memory is exhausted during the episode splitting operation and slower virtual/swap memory is used.

Both aggregation approaches yield virtually identical results with tiny differences in the observation times explained by floating point errors.

```
# are the differences in calculated exposures between
# `NosplitAggregate()` and `PopEpiAggregate` always smaller
# than 1 millionth person-years?
all(abs(lt_nosplit$O - lt_epi$pyrs) < 1e-6)
```

```
## [1] TRUE
```

The number of persons at risk at the start of every interval are identical. Note however that this does not have to be the case as `Epi` treats time intervals as open to the left and closed to the right, `(x, x+n]`, whereas `NosplitAggregate`, by default, does the opposite, `[x, x+n)`. For this comparison I changed the defaults.

```
all(lt_nosplit$P == lt_epi$at.risk)
```

```
## [1] TRUE
```

Censorings in `Epi` are given in a column that may be mistaken as an intrastate transition column, e.g. `fromAtoA`, `fromBtoB`. Keeping this in mind, here's the proof that `Epi` and `nosplit` count the censorings the same

```
all(
  lt_nosplit$to_cens ==
    (lt_epi$fromAtoA +
       lt_epi$fromCtoC +
       lt_epi$fromHtoH +
       lt_epi$fromKtoK +
       lt_epi$fromMtoM)
)
```

```
## [1] TRUE
```

Intrastate-age-transitions, i.e. transitions into the next age group while staying in the same state, are not counted by `Epi` whereas they are by `NosplitAggregate`. Due to this different behavior I'll exclude intrastate transitions from the following comparison of destination state counts by age and origin state.

All the transitions to `K` state are counted the same.

```
lt_nosplit %>%
  full_join(lt_epi, by = c('orig', 'x')) %>%
  filter(
    orig != 'K'
  ) %>%
  mutate(
    same_destination_counts =
      to_K ==
      (fromAtoK + fromCtoK + fromHtoK + fromMtoK)
  ) %>%
  pull(same_destination_counts) %>%
  all()
```

```
## [1] TRUE
```

All the transitions to `M` state are counted the same.

```
lt_nosplit %>%
  full_join(lt_epi, by = c('orig', 'x')) %>%
  filter(
    orig != 'M'
  ) %>%
  mutate(
    same_destination_counts =
      to_M ==
      (fromAtoM + fromCtoM + fromHtoM)
  ) %>%
  pull(same_destination_counts) %>%
  all()
```

```
## [1] TRUE
```

All the transitions to `A` state are counted the same.

```
lt_nosplit %>%
  full_join(lt_epi, by = c('orig', 'x')) %>%
  filter(
    orig != 'A'
  ) %>%
  mutate(
    same_destination_counts =
      to_A ==
      (fromCtoA + fromHtoA + fromMtoA)
  ) %>%
  pull(same_destination_counts) %>%
  all()
```

```
## [1] TRUE
```

Lexis triangles

A common scenario in demography is the aggregation of events and exposure times into intersecting intervals across multiple time-scales, namely age, calendar time and birth cohort. Any such intersection is called a Lexis-triangle. The no-split algorithm is designed to aggregate across a single time scale but can still be used for Lexis-triangle aggregation. In order to do so one separates the data set by birth cohort and runs the nosplit algorithm twice for each cohort subset, first with time on an age scale and then with time on a calendar scale. From the resulting age-cohort and period-cohort aggregates the corresponding events and exposure times under each age-period-cohort intersection can be inferred.

Here's a preliminary implementation:

```
# Aggregate Transitions Counts and Occupancy Times Into Lexis-Triangles
#
# @param df
#   A data frame.
# @param cohort
#   Calendar time when age = 0, i.e. birth cohort.
# @param age_in
#   Entry age into state.
# @param state_in
#   State being entered.
# @param age_out
#   Exit age from state.
# @param state_out
#   State being exited into.
# @param lexis_width
#   A numeric scalar giving the interval width of the
#   age, period and cohort groups
#
# @return
#   A data frame with columns
#     orig:     origin state
#     cohort:   birth cohort
#     period:   calendar time
#     age:      time since birth
#     triangle: Lexis triangle id on a per-cohort basis.
#               starts at 1 at the youngest age and increases with age
#     O:        total observation time of population visiting origin state
#     to_*:     number of exits from origin state to state *
NosplitAggregateLexis <- function (
  df, cohort, age_in, state_in, age_out, state_out,
```

```r
  lexis_width = 1, precision = 1e-8
) {

  cohort_ <- quo(cohort)
  age_in_ <- quo(t_in); state_in_ <- quo(d_in)
  age_out_ <- quo(t_out); state_out_ <- quo(d_out)

  # dimensions of Lexis surface
  min_cohort <- min(pull(df, !!cohort_))
  max_cohort <- max(pull(df, !!cohort_))
  min_age <- 0
  max_age <- max(pull(df, !!age_out_))
  min_period <- min_cohort
  max_period <- max_cohort + max_age

  # regular grid with width lexis_width where
  # all event times are contained within grid
  breaks_cohort =
    seq(
      (min_cohort%/%lexis_width)*lexis_width,
      (max_cohort%/%lexis_width)*lexis_width+lexis_width,
      lexis_width
    )
  breaks_age =
    seq(
      0,
      (max_age%/%lexis_width)*lexis_width+lexis_width,
      lexis_width
    )
  breaks_period =
    seq(
      (min_period%/%lexis_width)*lexis_width,
      (max_period%/%lexis_width)*lexis_width+lexis_width,
      lexis_width
    )

  # cohort-age counts
  ca_counts <-
    df %>%
    mutate(
      # quantize cohort to Lexis grid
      cohort = (!!cohort_%/%lexis_width)*lexis_width
    ) %>%
    group_by(cohort) %>%
    group_modify(
      ~ NosplitAggregate(
        ., !!age_in_, !!state_in_, !!age_out_, !!state_out_,
        breaks = breaks_age,
        drop0exp = FALSE, closed_left = TRUE
      )
    ) %>%
    ungroup() %>%
    mutate(
      triangle = 2*j
    )

  # cohort-period-counts
  cp_counts <-
    df %>%
    mutate(
      # translate age into period
      period_in = !!cohort_ + !!age_in_,
      period_out = !!cohort_ + !!age_out_,
      # quantize cohort to Lexis grid
      cohort = (!!cohort_%/%lexis_width)*lexis_width,
    ) %>%
    group_by(cohort) %>%
    group_modify(
      ~ NosplitAggregate(
        ., period_in, !!state_in_, period_out, !!state_out_,
        breaks =
          # period must be larger or equal to cohort
          breaks_period[breaks_period >= pull(.y, cohort)],
        drop0exp = FALSE, closed_left = TRUE, wide = TRUE
      )
    ) %>%
    ungroup() %>%
    mutate(
      triangle = 2*(j-1)+1
    )

  RecursiveDiff <- function (x) {
    k = length(x)
    y = vector(mode = 'numeric', length = k)
    y[1] = x[1]
    for (i in 2:k) {
      y[i] = x[i]-y[i-1]
    }
    return(y)
  }

  # empty surface of Lexis triangles
  # covering the smallest paralelogram shaped region
  # on the Lexis surface that contains all transition time
  # observations and is aligned with the Lexis grid intervals
  lexis_surface <-
    crossing(
      cohort = head(breaks_cohort, -1),
      triangle = seq(1, (length(breaks_age)-1)*2)
    ) %>%
    group_by(cohort) %>%
    mutate(
      age = rep(head(breaks_age, -1), each = 2),
      period = cohort + (triangle %/% 2)*lexis_width
    ) %>%
    ungroup()
  # infer total occupany times and state counts by Lexis triangle
  lexis_counts <-
    left_join(
      lexis_surface,
      bind_rows(
        ca_counts %>%
          select(orig, cohort, triangle, O, starts_with('to')),
        cp_counts %>%
          select(orig, cohort, triangle, O, starts_with('to'))
      ),
      by = c('cohort', 'triangle')
    ) %>%
    arrange(orig, cohort, triangle) %>%
    group_by(cohort) %>%
    mutate_at(
      vars(O, starts_with('to')),
      ~ RecursiveDiff(.),
    ) %>%
    mutate(O = ifelse(O < precision, 0, O)) %>%
    ungroup() %>%
    select(orig, cohort, period, age, triangle, O, starts_with('to'))

  return(lexis_counts)

}
```

We simulate data on the survival of 2 million individuals born 1900 until 1910. For each individual we know

- : the exact date of birth in fractional years,
- : the mode they entered observation (by immigration or by birth),
- : the age they entered observation,
- : the mode of exiting observation (death, emigration),
- : the age at exiting observation

```
N = 2e6
sim_dat <-
  tibble(
    # birth cohort
    cohort =
      # uniform distribution of births from 1900 to 1910
      runif(N, min = 1900, max = 1910),
    # entry state
    d_in =
      # 10% immigrated into population, 90% were born into it
      sample(c('immigration', 'birth'),
             size = N, replace = TRUE,
             prob = c(0.1, 0.9)),
    # exit state
    d_out =
      # 5% outmigrate, 95% die in country
      sample(c('emigration', 'death'),
             size = N, replace = TRUE,
             prob = c(0.05, 0.95))
  ) %>%
```

```r
  group_by(d_in) %>%
  mutate(
    # entry time
    t_in =
      # uniform immigration across ages 0 to 80, none after age 80.
      # entry age is 0 if born into population under observation
      if (d_in[1] == 'immigration')
        runif(n(), min = 0, max = 80) else 0
  ) %>%
  group_by(d_out) %>%
  mutate(
    # exit time
    t_out =
      if (d_out[1] == 'death')
        rweibull(n(), shape = 1.5, scale = 20) else
          rweibull(n(), shape = 2, scale = 40)
  ) %>%
  # left truncation
  filter(
    t_out >= t_in
  ) %>%
  ungroup()
```

Validate against `PopEpi`

And now the episode-split solution to Lexis triangle aggregation as implemented by `PopEpi`.

```r
LexisBreaks <- function (lexis_min, lexis_max, lexis_width) {
  seq(
    (lexis_min%/%lexis_width)*lexis_width,
    (lexis_max%/%lexis_width)*lexis_width+lexis_width,
    lexis_width
  )
}

microbenchmark(list = alist(
  nosplit = {
    lt_nosplit <-
      NosplitAggregateLexis(
        sim_dat, cohort, t_in, d_in, t_out, d_out,
        lexis_width = 5
      )
  },
  popepi = {
    lt_epi <-
      sim_dat %>%
      Lexis(
        entry = list(age = t_in, period = cohort + t_in),
        exit = list(age = t_out, period = cohort + t_out),
        entry.status = d_in,
        exit.status = d_out,
        data = .
      ) %>%
      splitMulti(
        breaks =
          list(
            age = LexisBreaks(0, 120, 5),
            period = LexisBreaks(1900, 2035, 5)
          ),
        timeScale = 'age'
      )
    # five year cohort intervals
    lt_epi$cohort <- lt_epi$cohort%/%5*5
    lt_epi <-
      aggre(
        lt_epi,
        by = list(state = lex.Cst,
                  cohort = cohort,
                  age = age,
                  period = period)
      )
  }
), times = 1)
```

```
## Incompatible factor levels in entry.status and exit.status:
##  both lex.Cst and lex.Xst now have levels:
##  birth immigration death emigration

## Unit: seconds
##     expr        min         lq       mean     median         uq        max
##  nosplit   3.809514   3.809514   3.809514   3.809514   3.809514   3.809514
##   popepi 201.135876 201.135876 201.135876 201.135876 201.135876 201.135876
##  neval
##      1
##      1
```

We can see that the `nosplit` approach is orders of magnitude faster at this task compared to the approach requiring an episode split first. There are some minor differences in output though which still need to be debugged.

```
# compare with nosplit
lt_nosplit %>%
  left_join(
    lt_epi %>%
      select(state, cohort, age, period, pyrs, starts_with('from')),
    by = c('orig' = 'state', 'cohort', 'period', 'age')
  ) %>%
  mutate_at(
    vars(pyrs, starts_with('from')),
    ~ ifelse(is.na(.), 0, .)
  ) %>%
  mutate(
    epi_to_death = frombirthtodeath + fromimmigrationtodeath,
    epi_to_emigration =
      frombirthtoemigration + fromimmigrationtoemigration,
    diff_O = abs(O - pyrs),
    diff_to_death = abs(to_death - epi_to_death),
    diff_to_emigration = abs(to_emigration - epi_to_emigration)
  ) %>%
  select(
    orig, cohort, period, age,
    diff_0, diff_to_death, diff_to_emigration
  ) %>%
  arrange(-diff_0, -diff_to_death)
```

```
## # A tibble: 216 x 7
##    orig        cohort period   age diff_0 diff_to_death diff_to_emigration
##    <chr>        <dbl>  <dbl> <dbl>  <dbl>         <dbl>              <dbl>
##  1 birth         1905   2030   125   5.75             0                  0
##  2 birth         1905   2035   125   5.57             1                  1
##  3 immigration   1905   1935    25   2.55             0                  1
##  4 immigration   1905   1940    35   2.55             0                  1
##  5 immigration   1905   1955    50   2.55             0                  1
##  6 immigration   1905   1960    55   2.55             0                  1
##  7 immigration   1905   1995    85   2.55             0                  1
##  8 immigration   1905   1955    45   2.55             0                  1
##  9 immigration   1905   1960    50   2.55             0                  1
## 10 immigration   1905   1965    55   2.55             0                  1
## # ... with 206 more rows
```