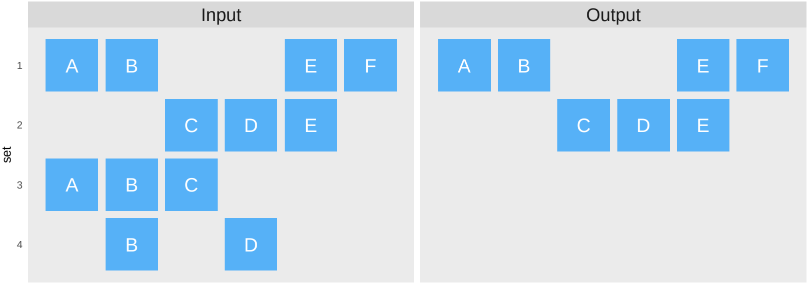
Set Cover Problem

Input: $\mathcal{S}$, collection of sets $S_1, \ldots, S_n$ covering $\mathcal{U}$:

$$S_1 \cup S_2 \cup \cdots \cup S_n = \mathcal{U}.$$

Output: Smallest subcollection from $\mathcal{S}$, covering $\mathcal{U}$.

Set Cover: A Toy Example



Set cover

- Fundamental problem in approximation algorithms with wide ranging applications; e.g. location planning, shift-planning and virus detection.
- Our application: Minimize number of hospitals, given every person in Germany can reach one hospital within 30 minutes.

RcppGreedySetCover

- Optimal solution available via linear programming, not feasible for large problems.
- Alternative: Greedy approximation.
- No fast solution in R available → `RcppGreedySetCover`!
    - Fast due to use of `data.table` and `Rcpp`.

Greedy Approximation: Algorithm

- Input: $\mathcal{S} = \{S_1, \ldots, S_n\}$.
- Initialize $\mathcal{C} \leftarrow \{\}, \mathcal{T} \leftarrow \mathcal{S}$.
- Repeat the following steps until $\mathcal{C}$ is a cover of $\mathcal{S}$:
    1. Find the largest set of *uncovered* elements, say Δ.
    2. $\mathcal{C} \leftarrow \mathcal{C} \cup \Delta$.
    3. $\mathcal{T} \leftarrow \{T_1 \setminus \Delta, \ldots, T_n \setminus \Delta\}$.

Greedy Approximation: Properties

- Tradeoff: Approximation error for speed / feasibility.
  - Error is bounded, approximation ratio depends on problem size.
- Vazirani 2001, p. 17: "[. . .], for the minimum set cover problem the obvious algorithm given above is essentially the best one can hope for."

Implementation

- Main requirement for containers: Efficient lookup and resizing.
  - No satisfactory solution in `R` available.
- Elements and sets are associated with integers $0, 1, \ldots$
  - Stored in `std::unordered_set<int>`. Grows / shrinks quickly.

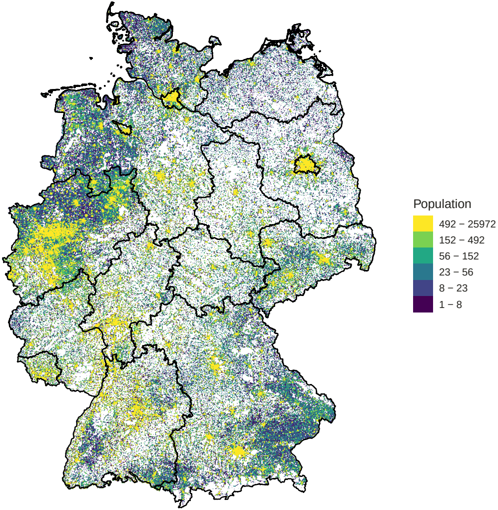
Implementation: Objects

- Map sets to elements via `std::vector<.>`.
  - Integer-representation of sets for indexing.
  - $O(1)$ cost for element access.
- Map elements to sets via `std::unordered_map<int,.>`.
  - $O(1)$ average cost for access and removal.

Implementation: Objects

- `boost::multi_index::multi_index_container` for set sizes.
- Custom interface:
  - Fast lookup of largest set size *and*
  - fast adjustment of set sizes.

Application: Data



Application: Data

Drivetimes for all populated grids-cells of 1km^2 raster in Germany within 40km radius, excluding drivetimes $>$ 30 minutes.

```r
head(D)
```

```
##         idm0      idm1 drivetime N_0 N_1
## 1: 4031_3109 4031_3110     157.2  92  23
## 2: 4031_3109 4031_3111     341.1  92  50
## 3: 4031_3109 4032_3108     198.8  92 166
## 4: 4031_3109 4032_3109     125.0  92 258
## 5: 4031_3109 4032_3111     298.7  92 244
## 6: 4031_3109 4033_3104     870.2  92 126
```

```r
nrow(D)
```

```
## [1] 164114074
```

Application

- Input: $N \times 2$ tidy `data.frame`. Sets are in the first, elements in the second column.

```
library(RcppGreedySetCover)
system.time(
  Res <- greedySetCover(D[,c("idm0","idm1")])
)
```

```
## 100% covered by 867 sets.
```

```
##    user  system elapsed
##  298.57    8.89  307.57
```

Application

- Output analogous to input.

```
head(Res)
```

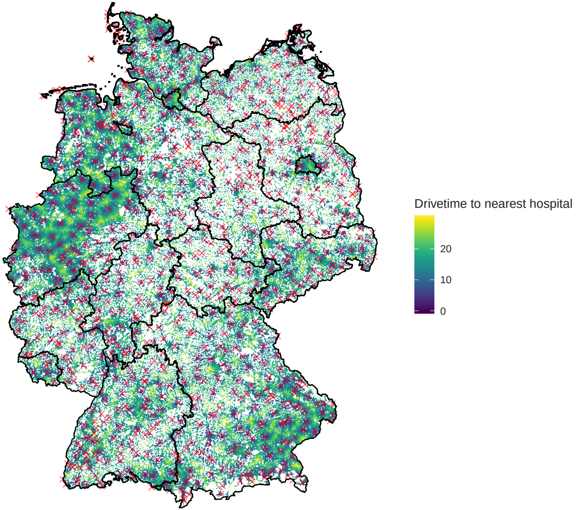
```
##          idm0      idm1
## 1: 4041_3197 4041_3189
## 2: 4041_3197 4041_3190
## 3: 4041_3197 4042_3189
## 4: 4041_3197 4046_3199
## 5: 4041_3197 4052_3180
## 6: 4046_3075 4040_3086
```

```
# Sanity check, TRUE if solution is a cover:
setequal(Res$idm1,D$idm1)
```

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

Application: Result

- Hospital marked by red X.



Future improvements

1. Speed up implementation.
2. Get rid of `data.table` dependency.
3. Extend to weighted set cover / capacitated set cover.