RcppGreedySetCover: Scalable Set Cover

Matthias Kaeding

RWI - Leibniz Institute for Economic Research / University Duisburg-Essen

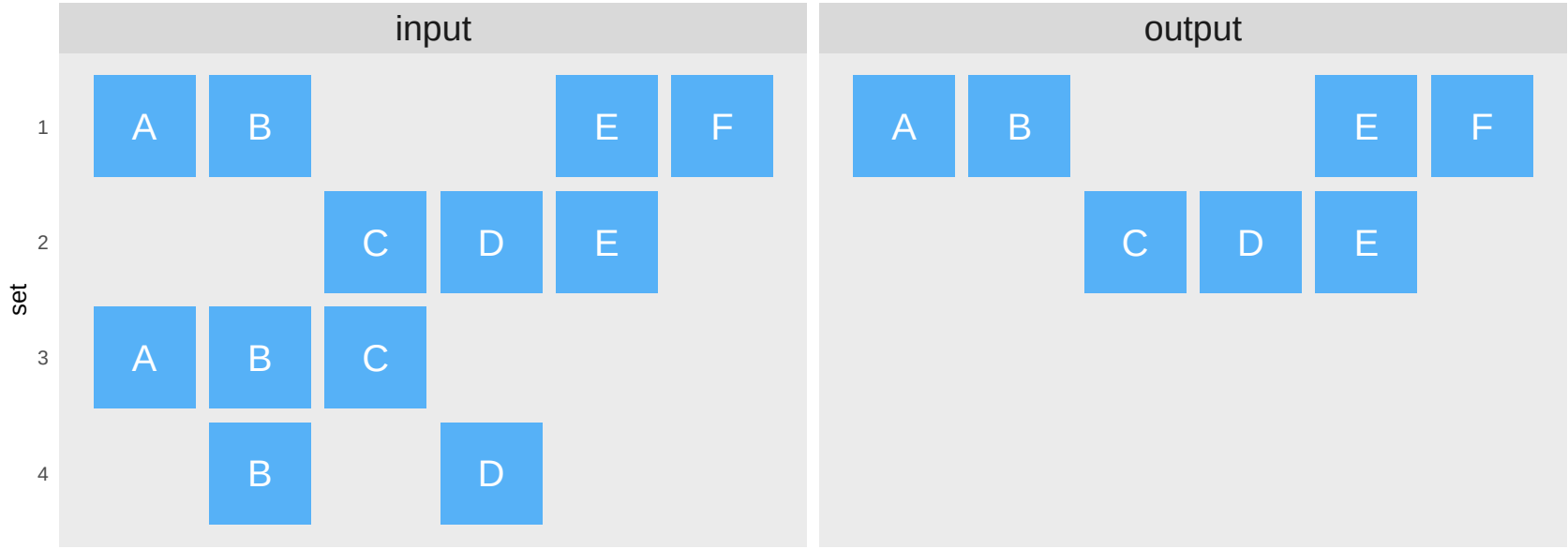
Set cover problem

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 $\mathcal{U}$.

Problem illustration



Set cover problem

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

RcppGreedySetCover

- Optimal solution available via linear programming but not feasible for large problems.
- Alternative: Greedy approximation as implemented in `RcppGreedySetCover`.
    - Single function package. Fast due to `data.table` and `Rcpp`.

Greedy 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. $\mathcal{T} \leftarrow \{T_1 \setminus \Delta, \ldots, T_n \setminus \Delta\}$.

Properties of greedy algorithm

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

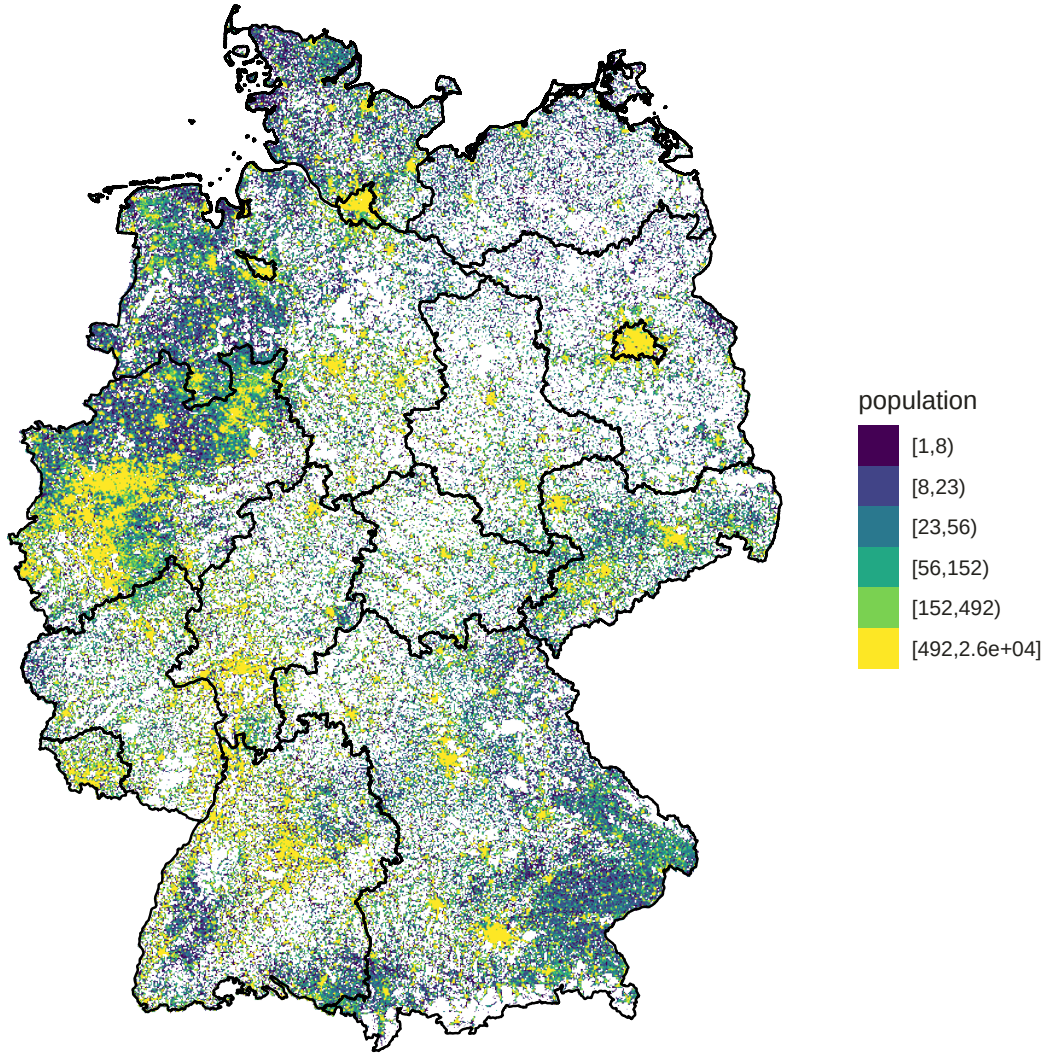
Implementation

- Preprocessing in `data.table`: Associate elements and sets with integers.
- Main part in `C++` via `Rcpp`. Major advantage: Data structures allowing fast lookup and resizing.

Data structures

- `std::vector<std::unordered_set<int>>` maps sets to elements.
  - $O(1)$ cost for element access.
- `std::unordered_map<int, std::unordered_set<int>>` maps elements to sets.
  - $O(1)$ average cost for access and removal.

Application: Data



Application: Data

Drivetimes for every populated 1km^2 grid in Germany within 40km radius, excluding drivetimes $>$ 30 minutes.

```
print(D[1:5, 1:3])
```

```
##          idm0      idm1 drivetime
## 1: 4031_3109 4032_3109      125.0
## 2: 4031_3109 4031_3110      157.2
## 3: 4031_3109 4032_3108      198.8
## 4: 4031_3109 4032_3111      298.7
## 5: 4031_3109 4034_3108      306.2
```

```
nrow(D) # Larger problem.
```

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

Application

- Input must be two column `data.frame` where the sets are in the first, the elements in the second column.

```
library(RcppGreedySetCover) # Available on CRAN
system.time(
  OUT <- greedySetCover(D[, c("idm0","idm1")])
)
```

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

```
##    user  system elapsed
##  323.22   37.50  316.63
```

Application

- Output is analogous to input.

```
head(OUT)
```

```
##          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
```

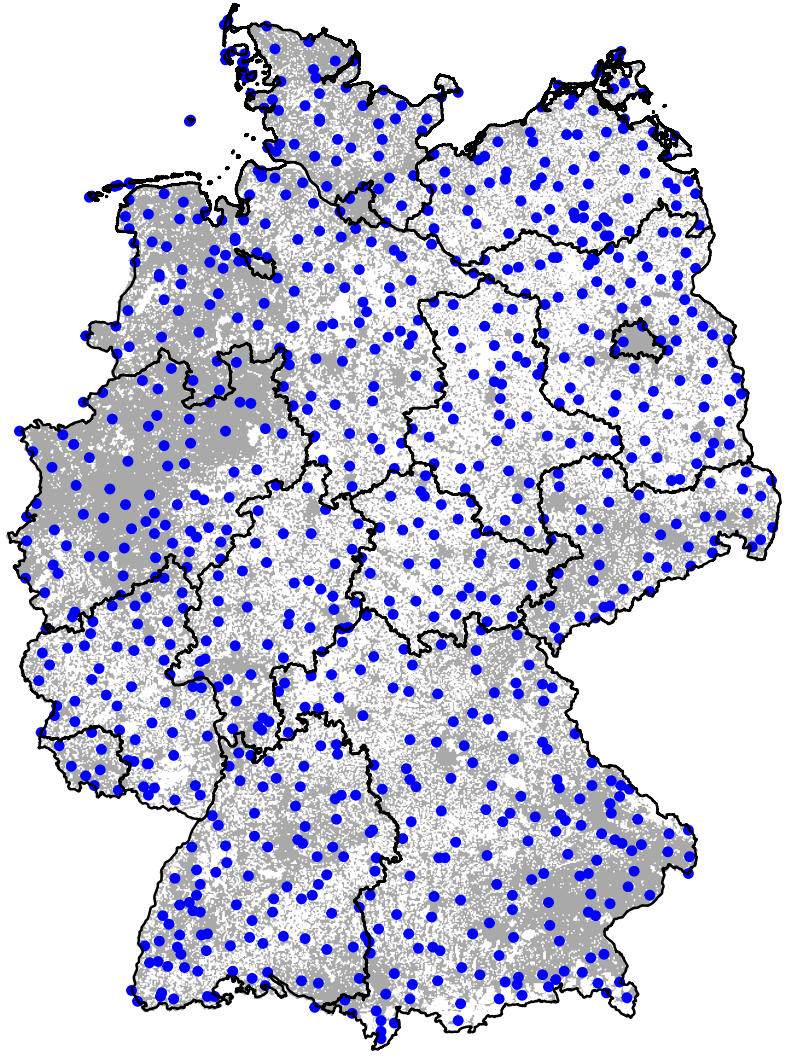
```
# Sanity check:
setequal(OUT$idm1, D$idm1)
```

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

```
# Solution is a cover.
```

Application: Result

- Blue points mark hospitals. Populated grids in darkgrey.



Future improvements

- Speed up implementation.
- Reduce dependencies to `Rcpp`.
- Extend to weighted / capacitated set cover.