Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

icd: Efficient Computation of Comorbidities from ICD Codes Using Sparse Matrix Multiplication in R

Jack O. Wasey
Children's Hospital of Philadelphia

Steven M. Frank
Johns Hopkins Medical Institutions

Mohamed A. Rehman
Johns Hopkins All Children's Hospital

Abstract

Medical research often describes the existing diseases (comorbidities) in its study populations, and retrospective research may rely on this information for risk adjustment. Sets of International Classifcation of Diseases (ICD) codes are often used for high-level classification of patients into comorbidities (e.g., 'Cancer' or 'Heart Disease') to help summarize characteristics of a population. Similar tools were not available in R, and do not scale to big data sets. The **icd** extension for R includes validated mappings of ICD codes to comorbidities, and uses an efficient, fast and scalable algorithm to apply these mappings to patient data. This enables reproducible workflows with hundreds of millions of patient records as are increasingly seen in large national and international databases.

Keywords: medical informatics, administrative data, matrix algebra, R, C++, ICD, mapping.

1. Introduction

ICD diagnostic codes are used to define tens of thousands of diseases. Originally from the World Health Organization (WHO), their purpose was to allow epidemiologists to study global health (**?**). In the USA, and elsewhere, the WHO ICD codes have been extended for administrative purposes, notably for billing (**?**). All over the world, ICD codes are the primary method by which diseases are recorded, often using national variants. This makes ICD codes important in many kinds of medical research. **icd** was made to meet the need for ICD code interpretation in R, and, in particular, it is designed to compute comorbidities

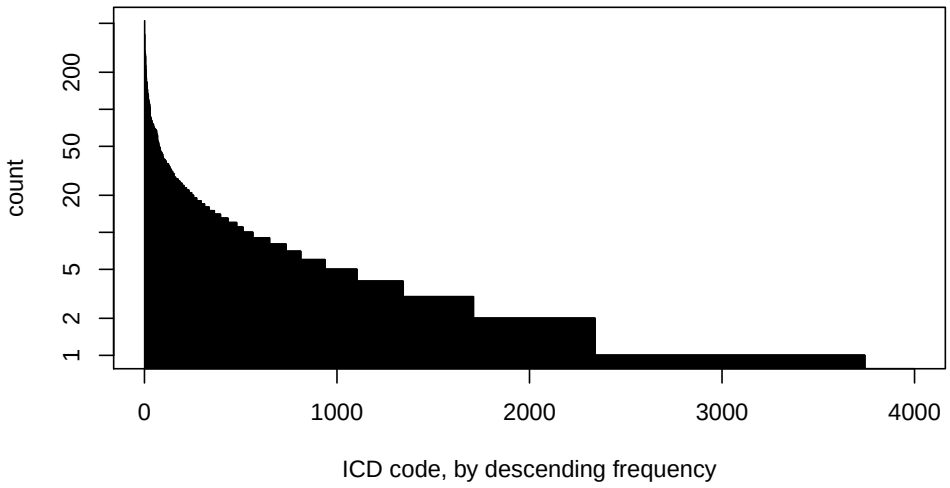

Figure 1: Frequency distribution of ICD codes in 5,000 pediatric hospital inpatients showing nearly 4,000 unique ICD codes; 1,401 appear only once, making them and the many other low-frequency codes of low or negative value in inter-group comparisons.

quickly, enabling a reproducible workflow when used with big or small data.

1.1. What is a comorbidity?

Given there are tens of thousands of diagnostic codes, it is extremely difficult to use them directly in most statistical models: there are a few relatively common ICD codes, but there is a long tail to the frequency distribution (see Figure 1 for an example) since even common diagnoses can be finely sub-divded. The almost universal solution is to group these codes in a standardized way (e.g., Quan *et al.* 2005; Elixhauser, Steiner, Harris, and Coffey 1998), and to use the presence and absence of any disease code in the comorbidity groups as covariates in models. The term comorbidity describes a disease which is present alongside a primary medical problem.

For example, a diabetic patient presents to hospital with a stroke: diabetes is the comorbidity and stroke is the presenting complaint. During or after an hospital admission, medical coders review the records and assign specific ICD (and other) codes. In this example, the patient might get the ICD-10 code E11.2 meaning 'Type 2 diabetes mellitus with renal complications,' and I63.0 meaning 'Cerebral infarction due to thrombosis of precerebral arteries.' In this case, the comorbidities might be regarded as: 'Diabetes' and 'Renal'. The term comorbidity generally refers to classes of diseases, not specific diagnoses, and that is how the term is used in the rest of this paper.

Term	Description
comorbidity	broad category of disease, e.g., Cancer
comorbidity map	set of comorbidities, each defined by diagnostic codes
patient-visit	record identifier, unique for each encounter with a patient, but could represent a patient at one moment, or a summation of all conditions a patient has ever had

Table 1: Terminology

1.2. Uses of comorbidities

Comparing groups

Almost every report of a medical investigation includes a table showing the characteristics of

Quan *et al.* (2011) and Agency for Healthcare Research and Quality (2018) only offer SAS code for this task].

Risk adjustment

Identifying differences between groups in retrospective medical data is not hard: they almost always exist. The eternal challenge of retrospective research is to account for those differences to make causal inferences. A common, somewhat historic strategy is to use scoring systems to encapsulate the severity of the comorbidities in one number, for example the Charlson Score (Charlson, Pompei, Ales, and MacKenzie 1987). Another approach is to use propensity scores or propensity matching: the inputs to propensity models include comorbidity information, which is a simple score like the Charlson Score, or a binary representation of the presence or absence of comorbidities for each patient or patient-visit (See Terminology). More sophisticated matching also relies on comorbidities for better risk adjustment, (e.g., Diamond and Sekhon 2012).

1.3. Mapping ICD codes onto comorbidities

The ubiquity of ICD codes means there is a long history of grouping them to form comorbidities, and these standardized groups have been used for decades as the basis for medical studies. As revisions of the ICD codes have appeared, and various diseases have waxed and waned in significance, people have worked to categorize them into these standardized comorbidity groups. The two main groups were developed by Charlson (Charlson *et al.* 1987), and Elixhauser (Elixhauser *et al.* 1998). Such mappings between comorbidity categories and sets of diagnostic codes will henceforth be called, comorbidity maps. The benefits of using existing comorbidity maps are:

- consistency and comparability with existing research
- they are well validated
- researchers can avoid the effort and errors involved in developing new comorbidity schemes.

Charlson did not initially use ICD codes to define the comorbidities, but various authors (e.g., Quan *et al.* 2005) have since classified ICD codes into those 17 comorbidity categories. Elixhauser (1998) developed ICD-based comorbidities for 31 diseases, and the US-based Agency for Healthcare Research and Quality (AHRQ) used this as a foundation for its own comorbidity groups (Agency for Healthcare Research and Quality 2018). Elixhauser and Charlson comorbidities have undergone refinement over the years, especially by Quan *et al.* (2011) whose meticulous work on ICD-9 and ICD-10 codes has been included in **icd**, alongside the AHRQ mappings.

1.4. Medical codes

Medical coding has a complicated history beginning with epidemiology, and snowballing to include medical billing and research. There are several major coding schemes, including the WHO ICD family, various national modifacations and extensions of the WHO ICD codes, the extensive USA ICD-CM (Clinical Modification), and several unrelated schemes with similar

goals. There are codes for diagnoses, procedures, medical equipment, and causes and circumstances of disease or injury. Codes sometimes have detail beyond what seems useful for routine clinical care, for example:

\begin{CodeChunk}

\begin{CodeInput} R> library("icd") R> explain_code("V97.33XD") \end{CodeInput}

```
[1] "Sucked into jet engine, subsequent encounter"
```

\end{CodeChunk}

Because the WHO ICD is a subset of ICD Clinical Modification, **icd** is driven by ICD-9-CM and ICD-10-CM, both of which are available in the public domain, allowing analysis of WHO codes and more detailed USA or other national sets of codes. Unfortunately, the WHO exercises copyright over the international ICD-10 scheme, so it cannot be included. This does not affect comorbidity computations.

ICD-9 codes are primarily numeric, have fewer codes defined, and have a more variable format, sometimes with a prefix of `V' or`E'. ICD-10 codes have top-level code in the form of a letter, then two numbers, with a longer section of mixed numbers and letters after the decimal divider. Both ICD-9 and ICD-10 codes can be presented with or without a decimal divider, thus there is potential ambiguity between ICD-9 and ICD-10 (e.g., V10 is in both schemes), and between ICD-9 codes without separators (e.g., 0100 and 100).

Breakdown of an ICD-10 code

This is a breakdown of an ICD-10-CM code, chosen to illustrate the hierarchical nature of the codes, and difference betwen WHO ICD-10 and ICD-10-CM. The following two codes are shared by the WHO definitions and ICD-10-CM.[1] \begin{CodeChunk}

\begin{CodeInput} R> explain_code(c("S62", "S62.6")) \end{CodeInput}

```
[1] "Fracture at wrist and hand level"
[2] "Fracture of other and unspecified finger(s)"
```

\end{CodeChunk}

These three are refinements offered only by ICD-10-CM:

\begin{CodeChunk}

\begin{CodeInput} R> explain_code(c("S62.60", "S62.607", "S62.607S")) \end{CodeInput}

```
[1] "Fracture of unspecified phalanx of finger"
[2] "Fracture of unspecified phalanx of left little finger"
[3] "Fracture of unspecified phalanx of left little finger, sequela"
```

\end{CodeChunk} These codes are all contained in the sub-chapter, `Injuries To The Wrist, Hand And Fingers,' (S60 to S69) and the chapter`Injury, poisoning and certain other consequences of external causes.' (S00 to T88).

Quirks of ICD codes

[1] Here the **icd** function `explain_code` is used. More detail on this can be found in Section 5.2.

There are multiple possible notations of the same ICD codes, and ICD-9-CM codes are particularly variable:

- presence or absence of a decimal point divider
- use of X as a filler
- zero-padding for three-digit codes < 100

In real data, ICD codes may not be constrained to a definitive list, so may contain frankly invalid codes, or codes which are valid in one edition of ICD, but not another, yet do fall clearly into one comorbidity.

1.5. Motivation and goals

The original motivation for **icd** was lack of any R software in the Comprehensive R Archive Network (CRAN) repository, or elsewhere, to compute comorbidities from ICD codes, nor any data that was easily parsable by R for interpreting ICD codes or representing the Charlson or Elixhauser schemes or the official lists of codes themselves.

The requirements at the outset were:

- enable a quickly reproducible workflow which includes a comorbidity computation
- accurate computation of Charlson and Elixhauser families of comorbidities
- faithful representation of comorbidity maps as intended by the original authors, with a reproducible audit trail back to the original SAS (SAS Institute, Cary NC, USA) code or published data
- performant enough for big data (Simpao, Ahumada, and Rehman 2015): updating the input patient data or comorbidity map should be possible without waiting minutes or hours to recompute the comorbidities
- extensible to allow addition of other comorbidity maps and scoring systems

Handling big data well aligns with another goal of enabling analysis of moderately big health care datasets with modest computing power. WHO ICD codes are used internationally, and it is important to enable their use without expensive computing resources, or reliance on the Internet for cloud computing. An open source licence (GNU General Public License v3.0 Free Software Foundation 2007) was chosen for this reason. This goal also helps an analyst or researcher to use a laptop to deal with all but the very biggest datasets, avoiding the complexity of using cloud computing and the risks of moving protected health information across the internet.

During development it became clear that the following were also important:

- handling of invalid data to minimize effect on comorbidities
- finding out the meaning of ICD codes
- navigating the ICD code hierarchies

Main computational problems

1. String matching

The central computational problem is one of using string matching to map ICD codes from patient data to the comorbidities, and recording the results. String matching is a slow operation, and memory intensive; the innovation described here is the use of matrix algebra to solve the problem. This requires a brief pre-processing step with string matching, whereas other solutions use string matching throughout.

2. Inexact comorbidity definitions

The comorbidity definitions in published literature do not precisely specify each individual code. This is partly a function of the various annual and international revisions of ICD codes, and also the need for brevity in publications, so ranges of codes are specified. For example, in the Valvular Heart Disease comorbidity in the ICD-9 Elixhauser scheme, we see it contains the range 394.0 – 397.1 . In the ICD-10 version of the Elixhauser map, there are many top-level codes, such as I34, I35 and I36, which themselves are never or rarely used for diagnostic coding. Therefore, there are few or no exact matches between candidate ICD codes and the codes or ranges in the comorbidities, so some kind of string matching must be done, or a determination of whether a code falls in a specified range. This is dealt with in detail in Section 5.1.

3. Big data

Bulk health care data, even for one hospital, often has hundreds of thousands or millions of encounters of different types. National databases and multi-hospital patient registries often have many more. Initial work using pure R code, with some optimization, resulted in computations taking minutes for about ten thousand patients. Although the problem is parallelizable, performance analysis on early versions using string matching showed that there was a lot of pressure on the CPU caches, so massive parallelization – when even an option – would have had diminishing returns with scaling. This is demonstrated by the benchmarks in the Results.

The problem of determining which patients have which disease classes can be expressed in pseudocode as nested loops:

```
for each patient-visit, get the ICD codes
  for each ICD code
    for each comorbidity
      search the lists of codes for that comorbidity
      if a matching diagnostic code is found, then
        record that comorbidity for the current patient-visit
```

This is an $\mathcal{O}(n)$ computation because the search element is limited to the fixed comorbidity map, thus making it asymptotically unimportant.

The benchmarks in the Results section show **icd** is much more efficient than string matching approaches.

Big data solutions cannot often be solved simply by increasing hardware capacity. The availability of many computing cores in the cloud does not help an algorithm which is limited to a single thread, or an algorithm which is ignorant of how CPU memory caching works. **icd** was designed to work efficiently on big health care data using parallel processing, and by minimizing the memory requirements of the problem.

2. Main features

2.1. User-facing

- Comorbidity computations
  - compute comorbidities based on ICD-9 or ICD-10 codes
  - offer a framework for comorbidity computations based on other medical codes
- ICD code processing and comprehension
  - validate ICD codes
  - convert between different ICD code representations
  - explain ICD codes with human-readable descriptions
  - convert between wide and long format patient data[2]
  - compare national and annual revisions
  - navigate the ICD hierarchies
- Scoring systems based on comorbidities
  - Charlson scores
  - Van Walraven scores
  - AHRQ Clinical Classification Software (CCS) scores
  - Centers for Medicare & Medicaid Services (CMS) Hierarchical Condition Code (HCC) scores

2.2. Internal

- ICD and comorbidity map data is extracted directly from published sources (journal articles and, where available, SAS code) in a reproducible and verifiable manner
- C and C++ code and accelerated matrix algebra to give accurate results quickly with big data
- extensive test suite

2.3. Included comorbidity maps

This package contains mappings to convert ICD codes to comorbidities using methods from several sources, based on the AHRQ, Charlson or Elixhauser systems. Updated versions of

[2] **icd** does not yet convert (also known as 'cross-walk') between ICD-9-CM and ICD-10-CM.

these lists from Agency for Healthcare Research and Quality (2018) and Quan *et al.* (2011) are included, along with the original Elixhauser *et al.* (1998) mapping. Since some data is provided in SAS source code format, this package has internal functions to parse this SAS source code and generate R data structures. Some lists are transcribed directly from the published articles, but interpretation of SAS code used for the original publications is preferable.

For example, here are the names of the comorbidities in the Charlson map:

\begin{CodeChunk}

\begin{CodeInput} R> names(icd10_map_charlson) \end{CodeInput}

```
 [1] "MI"          "CHF"         "PVD"         "Stroke"      "Dementia"
 [6] "Pulmonary"   "Rheumatic"   "PUD"         "LiverMild"   "DM"
[11] "DMcx"        "Paralysis"   "Renal"       "Cancer"      "LiverSevere"
[16] "Mets"        "HIV"
```

\end{CodeChunk}

and the ICD-10 codes from the first two comorbidities:

\begin{CodeChunk}

\begin{CodeInput} R> icd10_map_charlson[1:2] \end{CodeInput}

\begin{CodeOutput} $MI [1] “I21” “I210” “I2101” “I2102” “I2109” “I211” “I2111” “I2119” [9] “I212” “I2121” “I2129” “I213” “I214” “I22” “I220” “I221” [17] “I222” “I228” “I229” “I252” attr(,“icd_short_diag”) [1] TRUE attr(,“class”) [1] “icd10” “character”

$CHF [1] “I099” “I110” “I130” “I132” “I255” “I420” “I425” “I426” [9] “I427” “I428” “I429” “I43” “I50” “I501” “I502” “I5020” [17] “I5021” “I5022” “I5023” “I503” “I5030” “I5031” “I5032” “I5033” [25] “I504” “I5040” “I5041” “I5042” “I5043” “I509” “P290” attr(,“icd_short_diag”) [1] TRUE attr(,“class”) [1] “icd10” “character” \end{CodeOutput} \end{CodeChunk}

Note that these codes are of `icd10` and `character` class, and that they carry the attribute `icd_short_diag` which is set to `TRUE`.

3. Methods

The main internal feature of this software is the algorithm for assigning the diagnostic codes of a patient into comorbidities. The core of this implementation is a matrix multiplication, but this is only possible with some pre-processing.

3.1. Algorithm for comorbidity computation

1. ICD-9 and ICD-10 data preparation differs
   - ICD-9: all possible ICD-9 codes are pre-calculated in ICD-9 maps.
   - ICD-10: Partial matching to create map with relevant codes.[3]: $\mathcal{O}(n)$

[3]This partial matching is not based on regular expressions, but simply identifying whether the more significant characters of a longer code match a known parent. e.g., using the previous example, if S62.6 appeared in a comorbidity map, then S62.607S would be matched.

2. Reduce the problem to a matrix multiplication
    a. Compute the intersection of ICD codes from the patient data and the comorbidity map of interest: $\mathcal{O}(\log n)$ for time with a tree-based data structure. For ICD-10, this is done at the same time as Item 1, for no additional time complexity.
    b. Encode the ICD codes as consecutive integers: $\mathcal{O}(n)$ but could be optimized to $\mathcal{O}(1)$ by using the knowledge that there are now no duplicates.
    c. Generate a sparse patient-visit matrix, with one row per patient, and a column for each ICD code defined by the consecutive integers from the previous step: $\mathcal{O}(n)$
    d. Generate a dense comorbidity matrix, with one column for each comorbidity, and one row for each ICD code: $\mathcal{O}(1)$ in relation to n patient-visits.
3. Perform matrix multiplication: $\mathcal{O}(R_m \cdot n)$ time complexity where R_m is the ratio of non-zero to zero elements per row. For typical patient data, the mean is around five to ten codes per patient-visit, with many administrative data being limited to thirty. This low number compares favorably to the matrix width of tens of thousands.[4]

3.2. Data preparation

ICD-9

Step 1 of the algorithm is only applicable to ICD-9 codes. The comorbidity maps for ICD-9 codes already contain all the synactically valid permutations of child codes according to the ICD-9 specification. This means exact matching can be done in subsequent steps. The pre-calculation of the ICD-9 maps is done at package creation time, using internal functions which are included for reproducibility.

ICD-10

ICD-10 codes are more complex and have many more possible permutations: the same ICD-9 technique could be used, but the ICD-10 maps would be huge, and vulnerable to unexpected modifications; for example, a national ICD-10 variant using a different letter suffix which would not be captured. The solution is to have ICD-10 comorbidity maps which only include the level of details specified by the original author, often just the top-level three-digit codes, and use partial string matching,[5] combining steps 1 and 2a during one scan of the data.

3.3. Problem reduction

There are some major simplifications:

1. Only a fraction of possible ICD codes typically appear in health data, in step 2a

[4]Sparse matrix multiplication with dense matrices is already $\mathcal{O}(n)$ using the widespread Compressed Row – or Column – Storage (CRS or CCS) scheme. Even if dense matrix multiplication were used, it would still be $\mathcal{O}(n)$, despite being $\mathcal{O}(n^3)$ in the general case: in this problem, only one dimension of one matrix grows with increasing patient-visits. These are asymptotic limits for time, not for memory or bandwidth.

[5]This partial matching is not based on regular expressions, but simply identifying whether the more significant characters of a longer code match a known parent. e.g., using the previous example, if S62.6 appeared in a comorbidity map, then S62.607S would be matched.

2. ICD codes are codified as integers in step 2b
3. Sparse matrix representation can be used to dramatically reduce the memory requirement of the patient-visit matrix, in step 2c

Reduce number of codes

Reducing the total number of codes in both patient-visit and comorbidity data is performed in step 2a of the algorithm. For matrix multiplication, the columns n of the patient-visit matrix must match the rows q of the comorbidity map both in number. This means a common vocabulary for the codes in the patient-visits and the comorbidity maps must be established. This is accomplished in step 2b by using a factor,[6] where the levels represent the intersection of codes from the patient-visit data and the whole comorbidity map. Thus the integer factor indices become row or column indices in matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively. I.e., we only need to represent codes which are in both the patient-visit dad and the comorbidity map, making matrix $\boldsymbol{A}$ narrower and matrix $\boldsymbol{B}$ shorter. String searching strategies miss this optimization. Referring back to the pseudocode in Section 1.5.1, integer representation would allow much faster binary or linear search, but this is obviated by a matrix multiplication. Although the matrix multiplication probably performs more computations than needed, it has simpler flow control and is a problem for which highly-optimized solutions exist.

1. Prepare
    - ICD-9: All permutations already in maps
    - ICD-10: String match to reduce map to relevant codes
2. Reduce the problem
    a. Intersection of patient and map codes
    b. Encode as integer indices
    c. Generate sparse patient-visit matrix
    d. Generate dense comorbidity matrix
3. Perform matrix multiplication

Figure 2: Recap of algorithm

Sparse matrix representation

In a large data set, less common codes are more likely to appear, resulting in more nearly empty columns in the patient-visit matrix. Since this representation is used primarily to facilitate calculation of huge datasets, we ignore the fact that sparse format may be less efficient than dense for small datasets. Row-major representation also makes sense, with each row corresponding to a patient-visit. This is done in step 2c of the algorithm.

[6] R uses a data structure called a `factor` in which each element is stored as an integer index into an array of strings. The strings are unique, whereas the integers may be repeated.

The comorbidity matrix is more sparse than the patient-visit matrix, but is implemented as a dense data structure in step {itm:gendense} of the algorithm for two reasons: firstly, linear algebra software is often optimized for row-major sparse multiplications with dense matrices; secondly, this matrix has a maximum size limited by the comorbidity definitions, so does not need to scale with very large numbers of patients. For example, ignoring the problem reduction step described above, the maximum size of the ICD-9 AHRQ comorbidity matrix is: $14678 \times 30 \times 4 = 1.8 \times 10^6$, i.e., less than two megabytes,[7] which compares favorably to the eight megabyte CPU cache in the modest workstation used in the benchmarks presented in Section 4.

There is a detailed example below in Section 3.4, but a better idea of the bulk structure of this matrix can be seen below for a fictious comorbidity map with five categories, and also in Figure 3. Note that the code represented by the last row appears in the first and third comorbidities, whereas the others are all unique to one comorbidity.

$$B_{p,q} = \begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{matrix}$$

Matrix multiplication

This section first describes steps 2c and 2d of the algorithm in Section 3.1. Let $\boldsymbol{A}$ represent the matrix of comorbidities associated with each patient-visit, where each row, m, is a patient-visit, and each column, n, represents a different code. Each cell of the matrix is therefore either unity or zero. Unity indicating that the patient-visit on row m is associated with the code on column n; or zero, if not.

$$\boldsymbol{A}_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

Let matrix B be the comorbidity map, where each row, p, represents a different code, and each column, q, represents a comorbidity.

[7] Again the calculation is done using a 32-bit word for each flag.

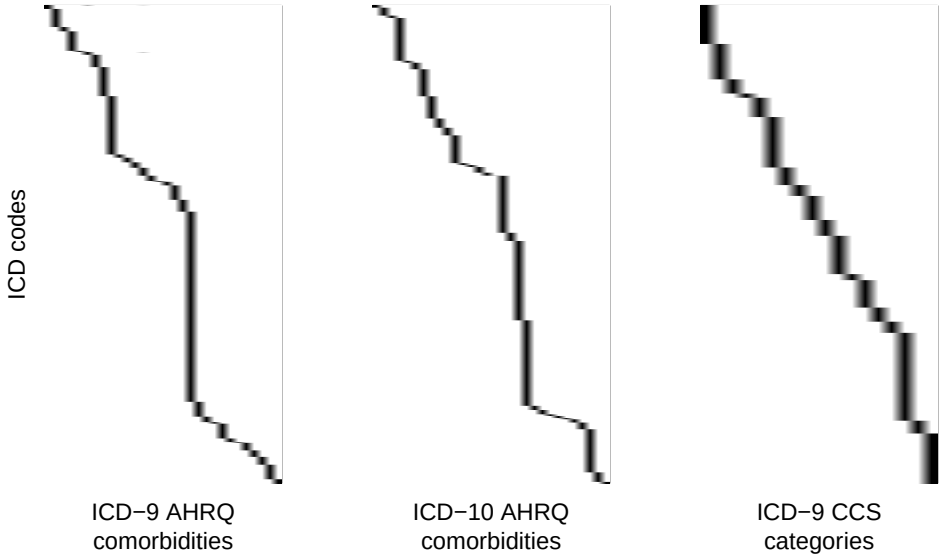

Figure 3: These are visualizations of some complete maps, black representing the appearance of a particular ICD code in a comorbidity column

$$\boldsymbol{B}_{p,q} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,q} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{p,1} & b_{p,2} & \cdots & b_{p,q} \end{pmatrix}$$

Given there are tens or hundreds of thousands of possible ICD-9 or ICD-10 codes, the possible width of $\boldsymbol{A}$ is large (n columns). There are also many ICD codes for each comorbidity, so the height of $\boldsymbol{B}$ is large (p rows), although typical comorbidity maps only cover a subset of possible codes. Many data sets have tens of millions of patient-visits, each with typically up to 30 diagnostic codes, the mean being around five to ten, depending on the dataset, so the memory requirement using lower estimates, using four bytes per flag,[8] is $10^7 \times 10^4 \times 4 = 4 \times 10^{11}$, which is 400 gigabytes simply to represent the patient-visit to disease relationships.

Since the integer levels of the `factor` of ICD codes is common between the patient-visit and comorbidity matrices, $n = p$ and the comorbidities for each patient-visit is their matrix product.

$$\boldsymbol{C}_{m,q} = \boldsymbol{A}\boldsymbol{B} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,q} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m,1} & c_{m,2} & \cdots & c_{m,q} \end{pmatrix}$$

[8] A C++ `int` is used for each comorbidity flag, which is typically, in 2018, a four-byte word. However, the matrices used by **icd** hold only true or false values, represented as 1 or 0. It appears inefficient to use an entire `int`. An alternative is bit-packing of 32 `bool` bits into each `int`, which has a stormy history in the C++ Standard Template Library (Järvi, Gregor, Willcock, Lumsdaine, and Siek 2006), and, more importantly, is not supported by any major linear algebra library.

3.4. Worked example using ICD-10 codes

Take four patient-visits with the following ICD-10 codes in wide format, seen in Table 2, and this simple comorbidity map:

```
R> list(
R>   "Rheumatic Heart Disease" = "I098",
R>   "Hypertension" = c("I10", "I11"),
R>   "Heart failure" = c("I50", "I110")
R> )
```

Patient-Visit	Code 1	Code 2	Code 3
Encounter one	K401		
Encounter two	I0981	C450	
Encounter three	M352	I10	
Encounter four	I110	H40001	I10

Table 2: Four patient-visits with some ICD-10 codes in 'wide' format for worked example

There are several things to note, which represent common features in real health care data:

- There are patient-visit codes which do not appear in the comorbidity map.
- There are codes in the comorbidity map which do not appear in the patient-visit codes.
- I11 appears in one comorbidities and its child code I110 appears in another.
- Patient two has code I0981, but only the parent code I098 appears in the comorbidity map.

Let $\boldsymbol{A}$ be a simplified set of patient-visits, where the columns represent the ICD-10 codes I0981 (rheumatic heart failure), I10 (essential hypertension), and I110 (hypertensive heart disease with heart failure). Again, each row is a different patient-visit.

Let $\boldsymbol{B}$ be a simplified comorbidity map, where the columns represent congestive heart failure and hypertension, in that order. Note that 110 is found in both these comrobidities.

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \qquad \boldsymbol{C} = \boldsymbol{AB} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix}$$

Note that cell $\boldsymbol{C}_{4,2} = 2$, because the condition I110 is in two comorbidities, so the final result can be given as a logical matrix $\boldsymbol{C} \neq 0$ Thus, the final result is:

$$(\boldsymbol{C} \neq 0) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Table 3 shows how this can be represented to the user.

Patient-Visit	Rheum	HTN	CHF
Encounter one			
Encounter two	yes		
Encounter three		yes	
Encounter four		yes	yes

Table 3: Output of the worked example using ICD-10 codes. 'Rheum' is Rheumatic Disease, 'HTN' is hypertension, 'CHF' is Congestive Heart Failure.

3.5. Worked example with anonymous patient data

The US State of Vermont offers anonymized public hospital discharge data (Vermont Department of Health 2016). A sample is included in **icd** and is used here to illustrate a real-world comorbidity calculation.[9]

\begin{CodeChunk}

\begin{CodeInput} R> head(vermont_dx[1:10]) \end{CodeInput}

\begin{CodeOutput} visit_id age_group sex death DRG DX1 DX2 DX3 DX4 DX5 1 7 40-44 male TRUE 640 27801 03842 51881 41519 99591 2 10 75 and over female FALSE 470 71526 25000 42830 4280 4019 3 13 75 and over female FALSE 470 71535 59651 78052 27800 V8537 4 16 55-59 female FALSE 470 71535 49390 53081 27800 V140 5 37 70-74 male FALSE 462 71536 4241 2859 2720 4414 6 41 70-74 male FALSE 462 71536 V1259 V1582 V160 V171 \end{CodeOutput} \end{CodeChunk}

The data is in 'wide' format. ICD codes have the 'short' structure, without the decimal divider. Diagnoses are spread of the `DX` columns.

\begin{CodeChunk}

\begin{CodeInput} R> v <- wide_to_long(vermont_dx)[c("visit_id", "icd_code")] R> head(v) \end{CodeInput}

\begin{CodeOutput} visit_id icd_code 1 7 27801 2 7 03842 3 7 51881 4 7 41519 5 7 99591 6 7 42842 \end{CodeOutput} \end{CodeChunk}

[9] A condition of use of this data requires the following text be included: "Hospital discharge data for use in this study were supplied by the Vermont Association of Hospitals and Health Systems-Network Services Organization (VAHHS-NSO) and the Vermont Green Mountain Care Board (GMCB). All analyses, interpretations or conclusions based on these data are solely that of [the requestor]. VAHHS-NSO and GMCB disclaim responsibility for any such analyses, interpretations or conclusions. In addition, as the data have been edited and processed by VAHHS-NSO, GMCB assumes no responsibility for errors in the data due to coding or processing by hospitals, VAHHS-NSO or any other organization, including [the requestor]."

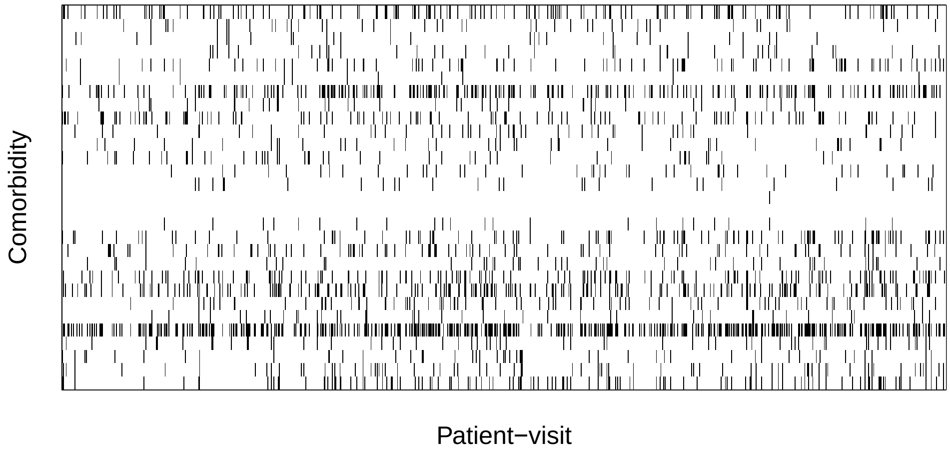

Figure 4: This visualization of the result of the comorbidity calculation shows a black cell for each positive comorbidity in one thousand patients from Vermont, USA.

This 'long' format is the structure required as input for the comorbidity functions, which can now be done:

\begin{CodeChunk}

\begin{CodeInput} R> v_cmb <- comorbid_charlson(v, return_df = TRUE) \end{CodeInput} \end{CodeChunk} \begin{CodeChunk}

\begin{CodeOutput} visit_id MI CHF PVD Stroke Dementia Pulmonary Rheumatic PUD 7 FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE 10 FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE 13 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE 16 FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE 37 FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE 41 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE LiverMild DM DMcx Paralysis Renal Cancer LiverSevere Mets HIV FALSE FALSE FALSE FALSE TRUE TRUE FALSE FALSE FALSE FALSE TRUE FALSE \end{CodeOutput} \end{CodeChunk}

4. Results

There are now three active CRAN packages which calculate comorbidities: **icd**, **medicalrisk** (McCormick and Joseph 2016), and **comorbidity** (Gasparini 2018). The latter two work using the strategy described in the psuedocode above.

The following is a limited performance comparison using synthetic data. This synthetic data shares some characteristics of real data: firstly, the more patients there are, the more coverage there is of the entire code space of the ICD scheme; secondly, there are both valid and invalid ICD codes; thirdly, each patient is assigned twenty codes. From the author's experience, the mean number of codes per patient is around five to ten, so twenty per patient represents twenty a mild stress-test of the algorithms.

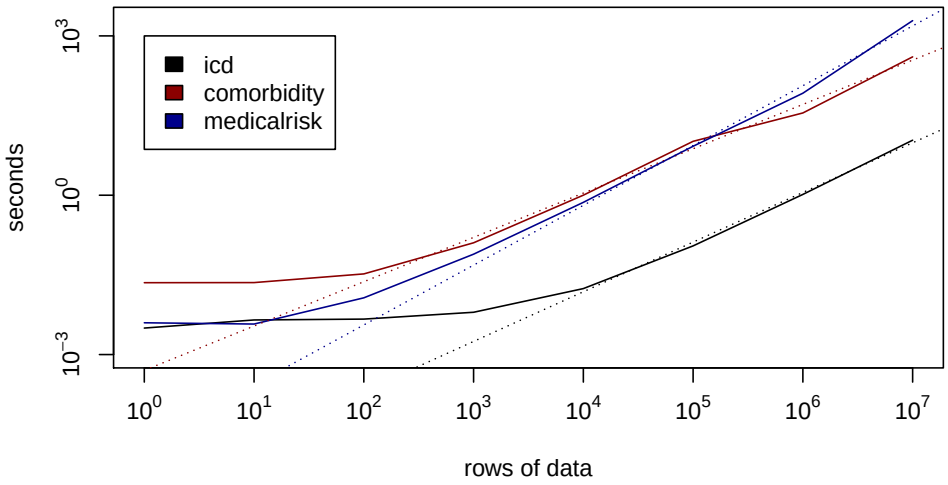

Figure 5: Performance comparison of comorbidity packages up to 10,000,000 rows, with 500,000 patient-visits and 20 comorbidities per visit. Models are fitted where the log-log relationship becomes linear, where rows > 1,000. Using an eight core 3.40GHz CPU, R 3.4.4 using Linux, kernel 4.15. **comorbidity** was run with and without parallel option, and the best result was chosen for each number of iterations.

Figure 5 shows time to compute comorbidites for increasing numbers of rows of data, showing **icd** is dramatically faster than the alternatives. Lines are fitted from where the relationship becomes linear at 10,000 rows of data, and these are used to extrapolate to estimates of the much larger computations seen in Figure 7. Figure 6 shows the relative speed up **icd** offers in comparison to the alternatives.

5. Implementation Details

5.1. Derivation of the comorbidity maps

It is important to have a reproducible audit trail for foundational work, so **icd** contains code which parses SAS source code in order to derive the original intent of the author in how the comorbidity maps were implemented. The AHRQ and Quan *et al.* provide such SAS source code, whereas other maps are only available in the form of tabulated data in journal articles.

Parsing orignal SAS code

For example, Quan *et al.* offer the following code for pulmonary disease in the ICD-9 Charlson map. The lines split for clarity.

```
%LET DC6=%STR('4168','4169',
'490','491','492','493','494','495','496',
'500','501','502','503','504','505',
'5064','5081','5088');
```

Note that 497 – 499 are undefined in ICD-9. What should be done if 497 appears in a data

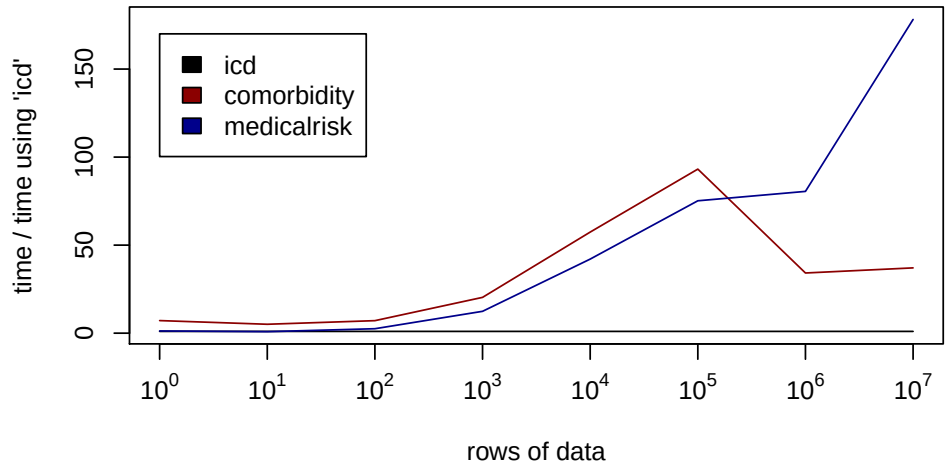

Figure 6: Relative speed-up using icd compared to the alternatives, using the same numbers of patient-visits and comorbidities in Figure 5.

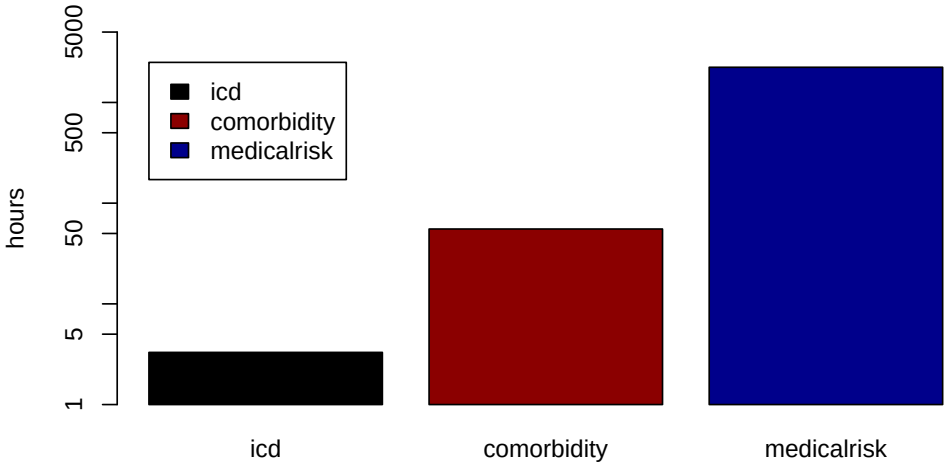

Figure 7: Predicted duration of computation for one hundred-million patient-visits, with twenty patients per patient

set? Here an argument is made that this is completely invalid. Firstly, see the sub-chapter definitions:

\begin{CodeChunk}

\begin{CodeInput} R> sc <- c("Chronic Obstructive Pulmonary Disease And Allied Conditions", R+ "Pneumoconioses And Other Lung Diseases Due To External Agents") R> icd9_sub_chapters[sc] \end{CodeInput}

```
$`Chronic Obstructive Pulmonary Disease And Allied Conditions`
start   end
"490" "496"

$`Pneumoconioses And Other Lung Diseases Due To External Agents`
start   end
"500" "508"
```

\end{CodeChunk}

497 is not a valid code itself. It could be a typo for any of the other combinations, of which three are valid:

\begin{CodeChunk}

\begin{CodeInput} R> explain_code(c("947", "749", "794", "479"), warn = FALSE) \end{CodeInput}

```
[1] "Cleft palate and cleft lip"
[2] "Nonspecific abnormal results of function studies"
[3] "Burn of internal organs"
```

\end{CodeChunk}

Given the wide range of disease processes, and no guarantee at all that the coder made a mistake only in the last digit, **icd** discards the code to avoid giving a false positive comorbidity flag.

```
R> "497" %in% icd9_map_charlson
```

```
[1] FALSE
```

The comorbidity maps are therefore constructed by generously including all children (valid or invalid) of the explicitly defined three-digit codes, but they do not extrapolate to other three-digit codes. Suppose additional digits were defined by a country's extension of ICD-9, but do not appear in the WHO or ICD-9-CM definitions. **icd** already includes all possible structurally valid ICD-9 child codes in the map. This can be seen here, where two ficticious decimal places are seen in the map, but not three:

```
R> "49699" %in% icd9_map_quan_deyo[["Pulmonary"]]
```

```
[1] TRUE
```

```
R> "496999" %in% icd9_map_charlson

[1] FALSE
```

More generosity is offered when calculating the comorbidities, so even if invalid codes appear, the intent of this code to fall within the pulmonary comorbidity hierarchy is clear enough:

\begin{CodeChunk}

\begin{CodeInput} R> alice <- data.frame(id = “alice”, icd9 = “49699”) R> comorbid_charlson(alice, return_df = TRUE)[[“Pulmonary”]] \end{CodeInput}

```
[1] TRUE
```

\end{CodeChunk}

Parsing range definitions

For some mappings, no source code was available, but the comorbidities are described in journals using ranges. e.g., in the ICD-9 Elixhauser mapping, we find 243 – 244.2 in the thyroid disease definition. This is a subset of the entire range of thyroid disesases in ICD-9: 244.3, 244.8 and 244.9 also exist. **icd** takes great care with false positives here by carefully excluding parent codes which might have been captured by a pattern matching approach. In this case, 244 should not be considered a match because it includes codes where which were clearly excluded. A number of ICD code range operators are defined to facilitate this:

\begin{CodeChunk}

```
R> head("243" %i9da% "244.2")
```

\begin{CodeOutput} [1] “243” “243.0” “243.00” “243.01” “243.02” “243.03” attr(,“icd_short_diag”) [1] FALSE attr(,“class”) [1] “icd9” “character” \end{CodeOutput}

```
R> "244" %in% ("243" %i9da% "244.2")

[1] FALSE
```

\end{CodeChunk} Note that 244 is too broad to fit the original Elixhauser description, because it implies all its children, which would go beyond 244.2.

5.2. Other Functionality

Validation

icd allows checking whether codes are valid, and, in the USA, whether they are ‘billable’, i.e., leaf nodes, or merely intermediate members of the hierarchy. Research and clinical data may also contain relevant non-billable codes, and this is accounted for by the comorbidity calculations.

\begin{CodeChunk}

\begin{CodeInput} R> is_valid(c(“441”, “441.0”, “441.01”, “XXX”)) \end{CodeInput}

```
[1]  TRUE  TRUE  TRUE FALSE
```

\begin{CodeInput} R> is_billable(c("441", "441.0", "441.01", "XXX")) \end{CodeInput}

```
[1] FALSE FALSE  TRUE FALSE
```

\begin{CodeInput} R> head(R+ data.frame(code = children("441"), R+ billable = is_billable(children("441"
\end{CodeInput}

```
   code billable
1   441    FALSE
2  4410    FALSE
3 44100     TRUE
4 44101     TRUE
5 44102     TRUE
6 44103     TRUE
```

\end{CodeChunk}

Hierarchy and explanation

Functions are provided to navigate the ICD-9 and ICD-10 hierarchies. This example first takes the children of the ICD-9 code 441.0: \begin{CodeChunk}

```
R> children("441")
```

\begin{CodeOutput} [1] "441" "4410" "44100" "44101" "44102" "44103" "4411" "4412" [9] "4413" "4414" "4415" "4416" "4417" "4419" attr(,"icd_short_diag") [1] TRUE attr(,"class") [1] "icd9" "character" \end{CodeOutput} \end{CodeChunk} Several five-digit codes begin with 4410. `explain_code` condenses all these children to their common parent, before interpreting the ICD code: \begin{CodeChunk}

\begin{CodeInput} R> explain_code(children("4410")) \end{CodeInput}

```
[1] "Dissection of aorta"
```

\end{CodeChunk} What does each one mean? \begin{CodeChunk}

\begin{CodeInput} R> explain_code(children("4410"), condense = FALSE) \end{CodeInput}

```
[1] "Dissection of aorta"
[2] "Dissection of aorta, unspecified site"
[3] "Dissection of aorta, thoracic"
[4] "Dissection of aorta, abdominal"
[5] "Dissection of aorta, thoracoabdominal"
```

\end{CodeChunk}

5.3. Software libraries

This package is built on the strong foundations of R (R Core Team 2018), **Rcpp** (Eddelbuettel and Francois 2011), **RcppEigen** (Eddelbeuttel and Bates 2013), and **Eigen** (Guennebaud, Jacob *et al.* 2010), the highly optimized C++ linear algebra library. Eigen was chosen because of its performance oriented approach to matrix multiplication using advanced x86 instructions when possible, and, in the case of the row-major sparse multiplication with a dense matrix, a multithreaded solution. Once this was impemented, the bottlenecks moved to the data preparation before the matrix multiplication, which itself was optimized by simple R techniques such as vectorization.

5.4. Extensions

R's S3 class system is used for extensibility, so it is straightforward to include additional ICD schemes, e.g., for different national systems. Likewise, it is also easy to add new comorbidity maps, or use user-defined ones. Several authors have contributed code which extends **icd** to solve other problems:

- Van Walraven risk scores (van Walraven, Austin, Jennings, Quan, and Forster 2009), analagous to Charlson scores, based on the Elixhauser comomribidites.
- assignment of CMS HCC categories (Evans 2011; Pope *et al.* 2004)
- emulation of the AHRQ CCS (Agency for Healthcare Research and Quality 2012)

6. Limitations and future work

In general, computation of comorbidities as used in medical research ignores the fact that both the comorbidity maps and the ICD schemes change from year-to-year. **icd** takes the approach to be inclusive where appropriate, so extra or missing codes result in the same or very similar comorbidity results. An extension could compute comorbidities using the correct annual revision according to the year of the patient encounter.

Many countries have their own variations on the WHO ICD codes, and these are not yet included in **icd**. These may affect comorbidity calculations. Also missing are WHO versions of ICD codes, which may be possible in the future, dependent upon licensing restrictions.

The synthetic data used for benchmarking contained real and invalid codes. More benchmarking could be done using real data with different proportions of invalid codes, and different distributions of codes.

7. Conclusions

icd gives R the ability to do a common medical research task by doing fast and accurate conversion of ICD-9 and ICD-10 into comorbidities. The key innovations are: the reduction of the problem to an equivalent smaller task; and the use of sparse matrix multiplication to compute comorbidities. This solution offers an efficient and elegant method that reduces

time and memory complexity. The benchmarks show that this technique scales to the biggest health care data sets, and answers the goal of making this possible with modest computing power in a reproducible workflow.

Acknowledgments

I am deeply grateful to Steve Frank and Mohamed Rehman for their support. Thanks to my talented colleagues at the Children's Hospital of Philadelphia, notably Aaron Masino, Allan Simpao, Jorge Galvez and Jonathan Tan. Thanks also to several people who have contributed code, notably the work on Van Walraven scores (William Murphy), HCC (Anobel Odisho), CCS (Vitaly Druker) and an alternative ICD decoding format (Ed Lee). A full list of contributors may be seen on the **icd** CRAN page, and on the project page.

References

Agency for Healthcare Research and Quality (2012). "Clinical Classifications Software for ICD-10 Data." http://www.ahrq.gov/research/data/hcup/icd10usrgd.html. Agency for Healthcare Research and Quality, Rockville, MD, USA.

Agency for Healthcare Research and Quality (2018). "Elixhauser Comorbidity Software for ICD-10-CM Healthcare Cost and Utilization Project (HCUP)." https://www.hcup-us.ahrq.gov/toolssoftware/comorbidityicd10/comorbidity_icd10.jsp. Agency for Healthcare Research and Quality, Rockville, MD.

Charlson ME, Pompei P, Ales KL, MacKenzie CR (1987). "A New Method of Classifying Prognostic Comorbidity in Longitudinal Studies: Development and Validation." *Journal of Chronic Diseases*, **40**(5), 373–383. ISSN 0021-9681. doi:10.1016/0021-9681(87)90171-8.

Diamond A, Sekhon JS (2012). "Genetic Matching for Estimating Causal Effects: A General Multivariate Matching Method for Achieving Balance in Observational Studies." *Review of Economics and Statistics*. ISSN 0034-6535. doi:10.1162/REST_a_00318.

Eddelbeuttel D, Bates D (2013). "Fast and Elegant Numerical Linear Algebra Using the RcppEigen Package | Bates | Journal of Statistical Software." *Journal of Statistical Software*, **52**(5). doi:10.18637/jss.v052.i05.

Eddelbuettel D, Francois R (2011). "Rcpp: Seamless R and C++ Integration | Eddelbuettel | Journal of Statistical Software." *Journal of Statistical Software*, **40**(8). doi:10.18637/jss.v040.i08.

Elixhauser A, Steiner C, Harris DR, Coffey RM (1998). "Comorbidity Measures for Use with Administrative Data." *Medical Care January 1998*, **36**(1), 8–27. ISSN 0025-7079.

Evans MA (2011). "Evaluation of the CMS-HCC Risk Adjustment Model." p. 127.

Free Software Foundation (2007). "GNU General Public License." URL http://www.gnu.org/licenses/gpl.html.

Gasparini A (2018). "Comorbidity: An R Package for Computing Comorbidity Scores." *Journal of Open Source Software*, **3**, 648. doi:10.21105/joss.00648. URL https://doi.org/10.21105/joss.00648.

Guennebaud G, Jacob B, *et al.* (2010). "Eigen v3, A C++ Template Library for Linear Algebra: Matrices, Vectors, Numerical Solvers, and Related Algorithms." http://eigen.tuxfamily.org. Online; accessed 5-May-2018.

Järvi J, Gregor D, Willcock J, Lumsdaine A, Siek J (2006). "Algorithm Specialization in Generic Programming: Challenges of Constrained Generics in C++." In *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '06, pp. 272–282. ACM, New York, NY, USA. ISBN 978-1-59593-320-1. doi:10.1145/1133981.1134014.

McCormick P, Joseph T (2016). *medicalrisk: Medical Risk and Comorbidity Tools for ICD-9-CM Data*. R package version 12, URL https://CRAN.R-project.org/package=medicalrisk.

Pope GC, Kautter J, Ellis RP, Ash AS, Ayanian JZ, Lezzoni LI, Ingber MJ, Levy JM, Robst J (2004). "Risk Adjustment of Medicare Capitation Payments Using the CMS-HCC Model., Risk Adjustment of Medicare Capitation Payments Using the CMS-HCC Model." *Health care financing review, Health Care Financing Review*, **25, 25**(4, 4), 119, 119–141. ISSN 0195-8631.

Quan H, Li B, Couris CM, Fushimi K, Graham P, Hider P, Januel JM, Sundararajan V (2011). "Updating and Validating the Charlson Comorbidity Index and Score for Risk Adjustment in Hospital Discharge Abstracts Using Data From 6 Countries." *American Journal of Epidemiology*, **173**(6), 676–682. ISSN 0002-9262, 1476-6256. doi:10.1093/aje/kwq433.

Quan H, Sundararajan V, Halfon P, Fong A, Burnand B, Luthi JC, Saunders LD, Beck CA, Feasby TE, Ghali WA (2005). "Coding Algorithms for Defining Comorbidities in ICD-9-CM and ICD-10 Administrative Data." *Medical Care*, **43**(11), 1130–1139. ISSN 0025-7079.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Simpao AF, Ahumada LM, Rehman MA (2015). "Big Data and Visual Analytics in Anaesthesia and Health Care." *British Journal of Anaesthesia*, p. aeu552. ISSN 0007-0912, 1471-6771. doi:10.1093/bja/aeu552.

van Walraven C, Austin PC, Jennings A, Quan H, Forster AJ (2009). "A Modification of the Elixhauser Comorbidity Measures into a Point System for Hospital Death Using Administrative Data." *Medical Care*, **47**(6), 626–633. ISSN 1537-1948. doi:10.1097/MLR.0b013e31819432e5.

Vermont Department of Health (2016). "Vermont Hospital Discharge Data." http://www.healthvermont.gov/health-statistics-vital-records/health-care-systems-reporting/hospital-discharge-data. Online; accessed 5-May-2018.

Affiliation:

Jack O. Wasey
Children"s Hospital of Philadelphia
3401 Civic Center Blvd. Philadelphia, PA 19102, USA email: waseyj@chop.edu url: http://www.chop.edu/doctors/wasey-jack

Steven M. Frank
Johns Hopkins Medical Institutions
1800 Orleans St. Baltimore, MD 21287, USA email: sfrank3@jhmi.edu url: https://www.hopkinsmedicine.org/profiles/results/directory/profile/1819685/steven-frank

Mohamed A. Rehman
Johns Hopkins All Children's Hospital
501 6th Ave S. St Petersburg, FL 33701, USA email: rehman@jhmi.edu url: http://www.jhmi.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd