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 Classification 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, or Organisation mondiale de la santé, OMS), they helped epidemiologists study global health (WHO 2018). In the USA, the WHO ICD codes have been extended for administrative purposes by the government Centers for Medicare and Medicaid Services (CMS), notably for billing and the insurance industry (CMS 2018). All over the world, ICD codes are used to record diseases and causes of death, sometimes using national variants. ICD codes are important in many kinds of medical research, e.g., for defining national health metrics (Lee *et al.* 2003); and in clinical studies in which patients are compared, or risk is calculated (e.g., Frank *et al.* 2014).

A widespread use of ICD codes is to show the differences between patient groups, often seen in the first table of published clinical studies (e.g., Tsugawa *et al.* 2016). ICD codes are grouped

ICD-10 code	Definition
I5082	Biventricular heart failure
I5083	High output heart failure
I5084	End stage heart failure
I5089	Other heart failure
I509	Heart failure, unspecified
P290	Neonatal cardiac failure

Table 1: Excerpt from an ICD-10 comorbidity map for CHF.

in categories, such as liver disease and diabetes. Patients' categories of diseases are compared instead of directly trying to compare their diverse ICD codes. These categories are known as *comorbidites*.

An unexpected use of **icd** with big national datasets was by Propublica for their Pulitzer finalist work exposing disparities in maternal health in the USA (Pulitzer 2018; Martin 2017).[1] Scientific work citing **icd** spans a wide variety of research areas, including cancer (Odisho *et al.* 2018), autism (Bishop-Fitzpatrick *et al.* 2018), cardiac surgery (Kiyota *et al.* 2017), infectious disease (Lane 2018), emergency medicine (Lane *et al.* 2019), nutrition (Hand *et al.* 2016), and rehabilition (Mundell *et al.* 2016).

icd was made to meet the need for ICD code interpretation in R, and to compute comorbidities quickly and robustly, enabling a reproducible workflow when used with big or small data, on big or small computers.

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-divided. The almost universal solution is to group these codes in a standardized way (e.g., Quan *et al.* 2005; Elixhauser *et al.* 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 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* is used in this article to refer more specifically to a defined set of diagnostic codes for a comorbidity. In Table 1, one can see the published Quan ICD-10 map (Quan *et al.* 2005) defining ICD codes for the standard Charlson comorbidities, includes the following for congestive heart failure (CHF):

[1] Use of this package discovered via personal communication with one of the authors from Propublica.

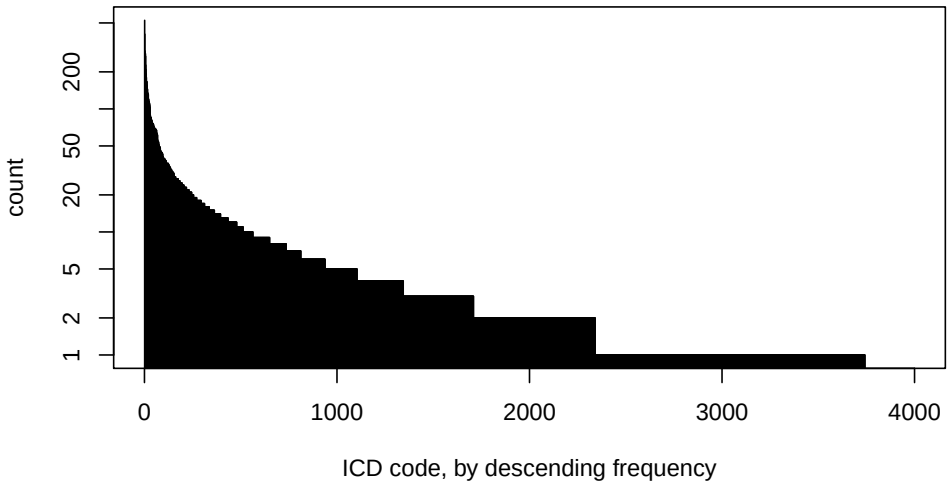


Figure 1: Frequency distribution of ICD codes in 5,000 pediatric hospital inpatients showing nearly 4,000 unique ICD codes; 1,401 of them appear only once. Low-frequency codes like these are of minimal use when statistically comparing groups, unless they are categorized into comorbidities.

Term	Description
comorbidity	broad category of disease, e.g., cancer. A set of diagnostic codes that are defined as belonging to this category of diseases.
comorbidity map	set of comorbidities, each comorbidity defined by set of 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
CHF	congestive heart failure, a comorbidity used in examples

Table 2: Terminology

1.2. Uses of comorbidities

Comparing groups

Almost every report of a medical investigation includes a table showing the characteristics of the groups comprising a study population. (For example, see Table 2 in Frank *et al.* (2014).) One purpose of this table is to show any differences between the groups under investigation. Things like age and weight can be compared numerically, whereas the tens of thousands of distinct diagnostic codes may only be compared by grouping into comorbidities. It is of central importance to retrospective studies that confounding is minimized by understanding differences in comorbidities between groups, then by risk adjustment. In randomized studies, this is achieved by recruiting enough patients and randomly assigning them to treatment: comorbidities of interest are still needed to demonstrate that recruitment was free from bias and that the sample size was large enough. It is helpful if different studies use the same criteria, and ideally the same methods, for establishing these comorbidities. One main purpose of **icd** is to enable standardized calculations in R using extremely well validated definitions of comorbidities, e.g., 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 in Table 2. 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 to enable basic understanding of global health and causes of death. There are now several major, with many countries having tailored the WHO editions to their own needs. The parent resource for ICD codes is the WHO, which recently defined ICD-11, although ICD-10 remains widely used. Examples of countries with national modifications are France, Belgium, and the USA.

The USA in particular has made extensive modifications and extensions, forming their Clinical Modification, resulting in ICD-9-CM and ICD-10-CM, with a focus on money, but with collateral benefits in research and public health. There are codes for diagnoses, procedures, and causes and circumstances of disease or injury. Many years of ICD-9 data remain in the electronic health records era, with ICD-9-CM being used from 1996 to 2011. The WHO ICD-9 codes were in use for global health from 1979 to 1998.

The Clinical Modification codes sometimes have detail beyond what seems useful for routine clinical care, whereas the WHO editions, with their basis in global health, are broader., For example, ICD-10-CM has:

```
R> library("icd")
R> explain_code("V97.33XD")

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

The US ICD Clinical Modification is much more extensive than the original WHO ICD definitions. **icd** is driven by ICD-9-CM and ICD-10-CM, because both are super-sets of the original definitions from the WHO, and they are available in the public domain. Since the codes are hierarchical, and deviations from the WHO or US versions as implemented by other countries are minor, the algorithms described below are robust to unrecognized codes. Unfortunately, the WHO exercises copyright over the international ICD-10 scheme, so **icd** (from version 4.0) prompts the user to download the French or English WHO codes when requested. Once they are downloaded, they may be used seamlessly. The same applies to code descriptions from France, Belgium, and historic years for some countries.

The core package data, with a recent ICD-9-CM and ICD-10-CM versions is included. Other data is downloaded as needed, and is cached avoiding ongoing need for an internet connection after this step. This can be done in bulk, or as needed either requirng one-off internet access for the requested data, and transparent offline usage thereafter. Users are asked for their permission before internet access or cache files are created.

```
explain_code(as.icd10who("R11"), lang = "fr")
```

```
[1] "Nausées et vomissements"
```

They may also be downloaded explicitly using `fetch_icd10who2008_fr(save_data = TRUE)` for the most recent published French edition, or `fetch_icd10who2016(save_data = TRUE)` for English. All the code definitions are downloaded at once, not on a per code basis.

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. ICD-10-CM also adds a letter in the third position for some codes. Both ICD-9 and ICD-10 codes can be presented with or without a decimal divider, thus there is potential ambiguity between ICD-9 codes without separators (e.g., 0100 and 100 are different). There is also overlap between ICD-9 and ICD-10 for some codes. The **icd** function `explain_code` is used. More detail on this can be found in Section 5.2.

```
R> explain_code(as.icd9("V10"))

[1] "Personal history of malignant neoplasm"

R> explain_code(as.icd10("V10"))

[1] "Pedal cycle rider injured in collision with pedestrian or animal"

R> explain_code("0100")

[1] "Primary tuberculous infection"

R> explain_code("100")

[1] "Leptospirosis"
```

Structure of an ICD code

The following is a breakdown of an ICD-10-CM code, chosen to illustrate the hierarchical nature of the codes, and differences between WHO ICD-10 and ICD-10-CM. The following two codes are shared by the WHO and ICD-10-CM definitions.

```
R> explain_code(c("S62", "S62.6"))

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

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

```
R> explain_code(c("S62.60", "S62.607", "S62.607S"))
```

```
[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"
```

These codes are all contained in the sub-chapter, "Injuries To The Wrist, Hand And Fingers", (S60–S69) and the chapter "Injury, poisoning and certain other consequences of external causes" (S00–T88).

Quirks of ICD codes

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

- decimal point divider: if present, this article refers to this as the long format; if absent, the short format.
- use of X as a filler
- zero-padding or lack thereof in three-digit ICD-9 codes from 000–099

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 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, which 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—if 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
```

For large n, this is an $\mathcal{O}(n)$ computation, where n is the number of patients or visits because the search element is limited to the fixed comorbidity map, thus making the map scan or search asymptotically unimportant. There are only between around ten to a few hundred codes in each comorbidity group, with only the cancer-related group having more: 5865 at present for ICD-9. Each patient has a limited number of diagnoses, and this is often capped at 15 or 30 per visit in the USA@. When there are millions of patients, with multiple visits to hospitals, the number of patients is dominant. For small numbers of patient visits, the search or scan in the above method does indeed become more significant, and much more dependent on the embedded search algorithm and hardware.

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

The main features of **icd** exposed to the user fall into three groups: those which calculate comorbidities from diagnostic codes; those which interpret and convert ICD data; and, those which use comorbidity results to calculate risk using well known scoring systems.

2.1. User-facing

- Comorbidity computations
  - compute comorbidities based on any derivates of ICD-9 or ICD-10 codes, such as ICD-10-CM
  - 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

[2]**icd** does not yet convert (also known as 'cross-walk') between ICD-9-CM and ICD-10-CM, but the foundational work was done during the 2019 NIH hackathon, for which **icd** was selected.

– Centers for Medicare & Medicaid Services (CMS) Hierarchical Condition Code (HCC) scores

2.2. Internal

icd has some important internal design choices made to make it faithful to the intention of the original comorbidity map authors, accurate, and fast, even with hundreds of millions of rows of input data.

- During the package creation, ICD definitions and comorbidity map data is extracted directly from published sources as far as possible (journal articles and, where available, SAS code) in a reproducible and verifiable manner; this is not done by the user. It is included so the process may be examined by a user to verify the source data is correctly processedl; and so changes can be tracked, which may have implications for reproducibility.
- C and C++ code and accelerated matrix algebra to give accurate results quickly with big data-sets, with an 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 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 was chosen when the source ICD codes were given as ranges, or they were numerous.

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

```
R> names(icd10_map_charlson)
```

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

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

```
R> icd10_map_charlson[1:2]
```

```
$MI
 [1] "I21"   "I210"  "I2101" "I2102" "I2109" "I211"  "I2111" "I2119" "I212"
[10] "I2121" "I2129" "I213"  "I214"  "I219"  "I21A"  "I21A1" "I21A9" "I22"
[19] "I220"  "I221"  "I222"  "I228"  "I229"  "I252"
```

```
$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"  "I508"   "I5081"  "I50810"
[33] "I50811" "I50812" "I50813" "I50814" "I5082"  "I5083"  "I5084"  "I5089"
[41] "I509"   "P290"
```

Note that these codes are of `icd10` and `character` class, and that they carry the attribute `icd_short_diag` which is set to `TRUE`, since there is no decimal divider.

3. Methods

The algorithm comprises several steps in order that an efficient matrix operation will work. The steps are summarized in Figure 2, and and detailed below, followed by examples from Section 3.2.

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. Matrix multiplication
4. Label, then restructure output when requested

Figure 2: Recap of algorithm

3.1. Algorithm for comorbidity computation

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 careful pre-processing, and some post-processing to correctly label the output matrix. A data frame or a matrix is returned, optionally re-ordering the output to match the user's order of patient-visits.

Some major simplifications are possible:

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

- ICD codes are codified as integers in step 2.
- Sparse matrix representation can be used to dramatically reduce the memory requirement of the patient-visit matrix, in step 3.
- Avoid complex restructuring of output matrix; the user can perform only if needed, in 3.1.4.

Step 1: Pre-processing user data

icd takes care of accepting a range of possible input data structures. The user may provide input data as follows:

- Enclosing structure
  - `data.frame`
  - class that inherits from a `data.frame` e.g., using **tibble** or **data.table**)
- Identifier within the data frame
  - integer vector, character vector or factor
  - one or multiple rows of data per patient-visit
  - ordered or unordered
- ICD codes
  - single or multiple columns
  - ICD-9 or ICD-10 (but should all be the same for one invocation of an **icd** comorbidity function)
  - *short* or *long* format codes (see Section 1.4.2).

The columns used for the identifier and ICD codes should be stated explicitly for reproducibility, but heuristics will select appropriate columns if they are not stated.

```
R> cmbs1 <- comorbid_ahrq(uranium_pathology)
R> cmbs2 <- comorbid_ahrq(uranium_pathology,
R+                        visit_name = "case",
R+                        icd_name = "icd10")
```

ICD-9 and ICD-10 data preparation differs. As there are fewer ICD-9 codes, and far fewer permutations permissible according to the definition, all possible ICD-9 codes are pre-calculated in ICD-9 maps, avoiding string processing in later steps. ICD-10 codes are handled differently.

ICD-9 The comorbidity maps for ICD-9 codes already contain all the syntactically valid permutations of codes in the hierarchy according to the ICD-9 and ICD-9-CM specifications. This is a super-set of the codes actually in use, and anticipates both international variations and future extensions, assuming unknown ICD-9 follow the defined syntax. Exact matching can be done in subsequent steps. The generation of the ICD-9 maps is done at package creation time, using internal functions which are included for reproducibility, as seen in: `icd:::icd10_generate_map_quan_elix`, and `icd:::icd9_parse_ahrq_sas`.

```
R> icd9_map_elix[["CHF"]]
```

```
  [1] "39891" "40211" "40291" "40411" "40413" "40491" "40493" "4280"  "42800"
 [10] "42801" "42802" "42803" "42804" "42805" "42806" "42807" "42808" "42809"
 [19] "4281"  "42810" "42811" "42812" "42813" "42814" "42815" "42816" "42817"
 [28] "42818" "42819" "4282"  "42820" "42821" "42822" "42823" "42824" "42825"
 [37] "42826" "42827" "42828" "42829" "4283"  "42830" "42831" "42832" "42833"
 [46] "42834" "42835" "42836" "42837" "42838" "42839" "4284"  "42840" "42841"
 [55] "42842" "42843" "42844" "42845" "42846" "42847" "42848" "42849" "4285"
 [64] "42850" "42851" "42852" "42853" "42854" "42855" "42856" "42857" "42858"
 [73] "42859" "4286"  "42860" "42861" "42862" "42863" "42864" "42865" "42866"
 [82] "42867" "42868" "42869" "4287"  "42870" "42871" "42872" "42873" "42874"
 [91] "42875" "42876" "42877" "42878" "42879" "4288"  "42880" "42881" "42882"
[100] "42883" "42884" "42885" "42886" "42887" "42888" "42889" "4289"  "42890"
[109] "42891" "42892" "42893" "42894" "42895" "42896" "42897" "42898" "42899"
```

ICD-10 Many more ICD-10 codes are defined than there are for ICD-9, especially so for ICD-10-CM. In addition, with a greater number of suffices and modifiers, the number of permutations of possible ICD-10 codes is huge, so pre-computing comprehensive comorbidity maps is not feasible. Partial matching to create map with relevant codes. 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. More detailed examples are shown later in this section. This computation tends to $\mathcal{O}(n)$ as, n (the number of patient-visits) increases. This is because the comorbidity map size is small, so the string matching within each set of codes in the map becomes insignificant.

The solution is to use ICD-10 comorbidity maps that only include the level of detail specified by the creators of the comorbidity maps (e.g., Quan *et al.* 2005; Elixhauser *et al.* 1998), then use partial string matching, combining steps 3.1.1 and 1 during one scan of the patient data.

The ICD-10 method can be applied to any input strings, which need not be medical codes. Any coding scheme with a lexicographically defined hierarchy will work.

For example, the Quan-Deyo ICD-10 map (Quan *et al.* 2005) contains the following for CHF:

```
R> icd10_map_quan_deyo[["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"  "I508"   "I5081"  "I50810"
[33] "I50811" "I50812" "I50813" "I50814" "I5082"  "I5083"  "I5084"  "I5089"
[41] "I509"   "P290"
```

Note that I50 appears here, but any of the following patient-visit ICD-10 codes would match.

```
R> grep(pattern = "I50", icd10cm2019[["code"]], value = TRUE)
```

```
 [1] "I50"    "I501"   "I502"   "I5020"  "I5021"  "I5022"  "I5023"  "I503"
 [9] "I5030"  "I5031"  "I5032"  "I5033"  "I504"   "I5040"  "I5041"  "I5042"
[17] "I5043"  "I508"   "I5081"  "I50810" "I50811" "I50812" "I50813" "I50814"
[25] "I5082"  "I5083"  "I5084"  "I5089"  "I509"

R> # I50 should match exactly:
R> icd10_comorbid_quan_deyo(data.frame(id = 1, code = "I50"))[1, "CHF"]

[1] TRUE

R> # I502 is a defined ICD-10-CM code
R> explain_code("I502")

[1] "Systolic (congestive) heart failure"

R> icd10_comorbid_quan_deyo(data.frame(id = 1, code = "I502"))[1, "CHF"]

[1] TRUE

R> # a fictitious code anywhere beneath I50 should match:
R> is_defined("I502X9")

[1] FALSE

R> icd10_comorbid_quan_deyo(data.frame(id = 1, code = "I502X9"))[1, "CHF"]

[1] TRUE

R> is_defined("I09")

[1] TRUE

R> explain_code("I09")

[1] "Other rheumatic heart diseases"

R> icd10_comorbid_quan_deyo(data.frame(id = 1, code = "I09"))[1, "CHF"]

[1] FALSE
```

Similarly, if a patient has a more general code, it should not match a more specific code in an ICD-10 map. I099 appears in the map above, but, if a patient was assigned I09, should not match, since it may well include diagnoses not intended by the map author.

Another example is from the Quan revision of the Elixhauser ICD-10 map. In this case, for simplicity with relatively few codes, they were transcribed from the published article (Quan *et al.* 2005) into the internal **icd** function, `icd:::icd10_generate_map_quan_elix`. An initial `list` contains the following for CHF:

```
R> CHF <- c(
R>   "I099", "I110", "I130", "I132", "I255", "I420", "I425", "I426",
R>   "I427", "I428", "I429",
R>   "I43", "I50", "P290"
R> )
```

Step 2: Reduce to matrix multiplication

This step is critical in allowing subsequent matrix computation of the problem. The key steps are elaborated below. In summary:

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 Step 3.1.1, 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.

Reduce size of problem space Reducing the total number of codes in both patient-visit and comorbidity data is performed in step 1 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 2 by using a factor,[3] 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 data 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 allows much faster binary or linear searches than repeated string pattern matching, but all this is avoided by a matrix multiplication. String matching requires transferring more memory, filling the CPU cache lines with data more quickly, then sequentially looping through characters; worse still, R stores strings in a global cache: although hashed, each string fetch is much more likely to require accessing a memory area not in the CPU cache, and R's hash lookup itself is overhead in both CPU time and memory access. Early benchmarking and profiling during development clearly showed string comparison was the bottleneck.

Although the matrix multiplication probably performs more computations than needed (loop and string-based approaches could break if or when a match is found for one patient-visit, for one comorbidity), matrix multiplication is simpler highly-optimized by multiple libraries.

[3] 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.

Sparse matrix representation In a large data set, less common codes are more likely to appear, resulting in many more almost empty columns in the patient-visit matrix. A rare disease might affect about one in a million people: suppose one patient happened to have the disease in sample of a million patients: the remaining 999,999 must be marked as not having the disease, since the rare disease has its own column.

Since this representation is used primarily to facilitate calculation of huge data sets, we ignore the fact that sparse format may be less efficient than dense for small data sets. This is done in step 3 of the algorithm.

Row-major representation is the storage of data in the computer consisting of the data in each row being stored, followed by the data in the next row, and so on. This further improves efficiency because the patient-visit is the unit of interest when analyzing the data. The memory access for a whole patient is therefore limited to a small (hopefully) contiguous chunk; with column-major matrix representation, the codes for a patient could be spread across system memory, particularly problematic when the data overflows into slow disk swap space.

The comorbidity matrix is also quite sparsely populated, but is implemented as a dense data structure in step 4 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,[4] 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 3.2, but a better idea of the bulk structure of this matrix can be seen below for a fictitious 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.

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

Step 3: Matrix multiplication

Perform matrix multiplication, which has $\mathcal{O}(R_m \cdot n)$ time complexity where R_m is the ratio of non-zero to zero elements per row.

[4] This calculation is done using a 32-bit word for each flag.

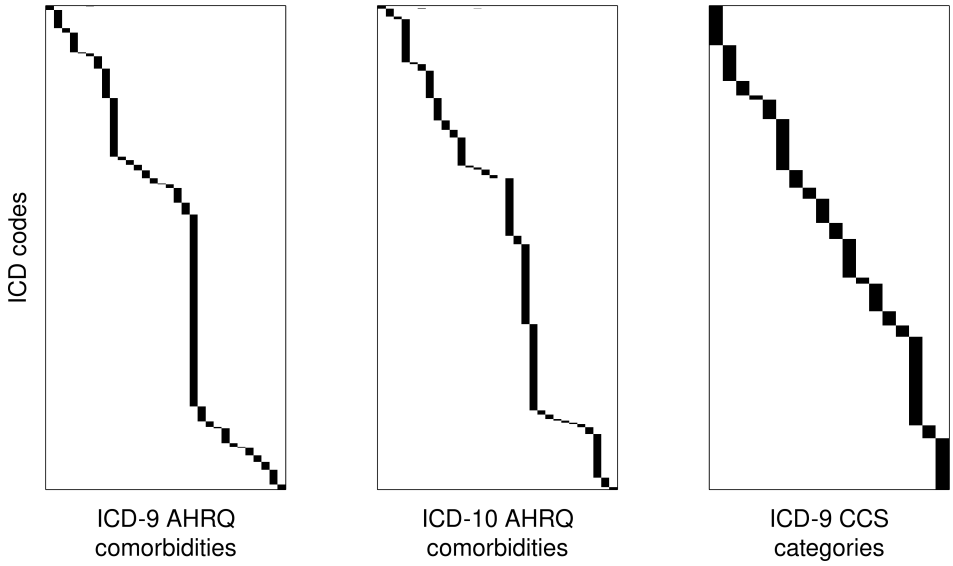


Figure 3: These are visualizations of some complete maps, black representing the appearance of a particular ICD code in a comorbidity column

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.[5]

This section first describes steps 3 and 4 of the algorithm in Section 3.1. Let

$\boldsymbol{A_{m,n}}$

represent the matrix of comorbidities associated with each patient-visit, where each row is a patient-visit, and each column represents a different code. Each cell of the matrix is therefore either unity or zero. Unity indicating that a patient is associated with a certain diagnostic code; or zero, if she is 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.

[5]Sparse matrix multiplication with dense matrices is already $\mathcal{O}(n)$ using the Compressed Row or Column Storage (CRS or CCS) scheme, where n is the number of patient-visits. Even if dense matrix multiplication could be used, it would tend towards $\mathcal{O}(n)$, 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 accounting for memory or storage bandwidth.

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

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,[6] 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.

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

Step 4: Label, and transform to user-requested structure

The matrix product may need post-processing to return data structures expected by the user. By default a logical matrix is returned, but many users prefer data frames. Although it is not difficult for a user to make the conversion, **icd** takes responsibility for restoring any factors given in the input data, labelling rows and columns of the matrix, or inserting an identifier column to data frames. The identifier may be a vector of numeric or string values, or be a factor. In addition, the order of patient-visit rows in the output is not necessarily the same as the input.

A typical user workflow involves gathering patient data, using **icd** to extract comorbidities, then merging the comorbidities with patient information such as age, height and weight. Therefore, **icd** does not by default restore the order of patients, which can be a significantly computationally expensive process with a large number of patient-visits, since merging will does not require ordered data.

Furthermore, if the patient-visit identifiers are not in *long* form

```
R> library(icd)
R> # Packaged sample Vermont data is already one row per patient, so order should
```

[6] 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 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.

```
R> # be restored in output
R> v <- comorbid_charlson(vermont_dx)
R> identical(as.integer(rownames(v)), vermont_dx[["visit_id"]])

[1] TRUE

R> vermont_shuffled <- vermont_dx[sample(seq_len(nrow(vermont_dx))), ]
R> w <- comorbid_charlson(vermont_shuffled)
R> identical(as.integer(rownames(w)), vermont_shuffled[["visit_id"]])

[1] TRUE

R> # Packaged sample Uranium Pathology data is long-format with one row per diagnostic
R> # code, multiple rows per patient. The case identifier increases monotonically.
R> u <- comorbid_charlson(uranium_pathology)
R> uranium_shuffled <- uranium_pathology[sample(seq_len(nrow(uranium_pathology))), ]
R> # avoiding restoring the order can be avoided, but the ids then differ in order
R> s1 <- comorbid_charlson(uranium_shuffled, restore_id_order = FALSE)
R> identical(as.integer(rownames(s1)), unique(uranium_shuffled[["case"]]))

[1] FALSE

R> # the default does restore the order as given:
R> s2 <- comorbid_charlson(uranium_shuffled)
R> identical(as.integer(rownames(s2)), unique(uranium_shuffled[["case"]]))

[1] TRUE

R> # and we can confirm that the output data is identical when re-ordered
R> s3 <- s1[order(as.integer(rownames(s1))), ]
R> s4 <- s2[order(as.integer(rownames(s2))), ]
R> identical(as.integer(rownames(s3)), unique(uranium_pathology[["case"]]))

[1] TRUE

R> identical(as.integer(rownames(s4)), unique(uranium_pathology[["case"]]))

[1] TRUE

R> # re-ordered results from the shuffled and unshuffled data are the same:
R> identical(s3, u)

[1] TRUE

R> identical(s4, u)
```

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 3: Four patient-visits with some ICD-10 codes in wide format

```
[1] TRUE
```

3.2. Example: using ICD-10 codes

Take four patient-visits with the following ICD-10 codes in wide format, seen in Table 3, and this simple comorbidity map:

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

There are several things to note, which represent common features in real 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 comorbidity and the code I110, which is beneath it in the hierarchy, appears in another comorbidity.
- Patient two has code I0981, but only the parent code I098 appears in the comorbidity map.

Stepping through the algorithm:

1. These are ICD-10 codes, so there is some additional character string processing to do.

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); each row represents a different patient.

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

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

Note that:

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

Table 4: Output of the example using ICD-10 codes. *Rheum* is rheumatic disease, *HTN* is hypertension, *CHF* is congestive heart failure.

$\boldsymbol{C}_{4,2} = 2$

The condition I110 is in two comorbidities, whereas we wish only to know if any of the codes were present in each comorbidity. The final result can be given as a logical matrix $\boldsymbol{C} \neq 0$

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

Table 4 shows how this can be represented to the user.

3.3. Example: comorbidities from real ICD-9 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.[7]

```
R> head(vermont_dx[1:10])
```

```
  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
```

The data is in wide format, with multiple columns of ICD codes. ICD codes have the short structure, without the decimal divider. Each row represents one of '$rnrow(vermont_dx)$' patients, with up to 20 different diagnostic codes drawn from ICD-9.

[7] 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]."

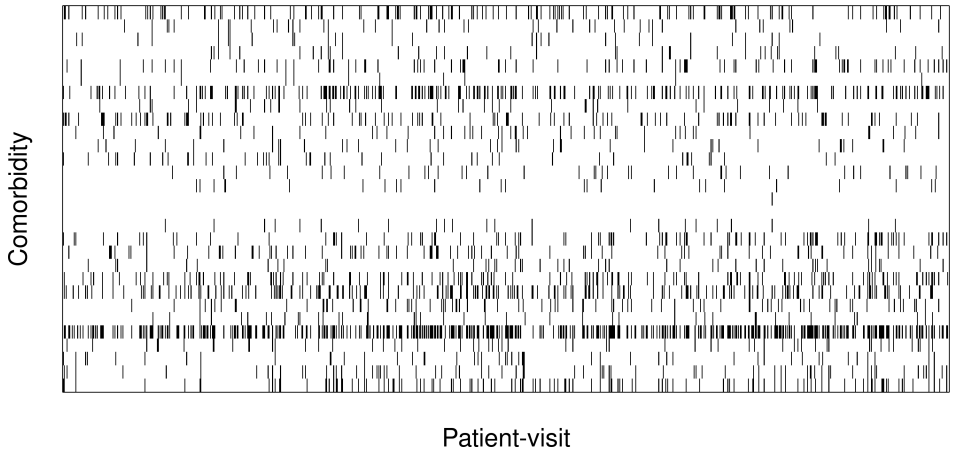


Figure 4: This visualization of the result of the comorbidity calculation shows a black cell for each positive comorbidity in one thousand patients from Vermont, USA.

```
R> v <- vermont_dx[-c(2:5)]
R> v[1:5, 1:5]

  visit_id   DX1   DX2   DX3   DX4
1        7 27801 03842 51881 41519
2       10 71526 25000 42830  4280
3       13 71535 59651 78052 27800
4       16 71535 49390 53081 27800
5       37 71536  4241  2859  2720

R> v_cmb <- comorbid_charlson(v, return_df = TRUE)

 visit_id    MI   CHF   PVD Stroke Dementia Pulmonary Rheumatic   PUD LiverMild
        7 FALSE  TRUE FALSE  FALSE    FALSE     FALSE     FALSE FALSE     FALSE
       10 FALSE  TRUE FALSE  FALSE    FALSE      TRUE     FALSE FALSE     FALSE
       13 FALSE FALSE FALSE  FALSE    FALSE     FALSE     FALSE FALSE     FALSE
       16 FALSE FALSE FALSE  FALSE    FALSE      TRUE     FALSE FALSE     FALSE
       37 FALSE FALSE  TRUE  FALSE    FALSE     FALSE     FALSE FALSE     FALSE
       41 FALSE FALSE FALSE  FALSE    FALSE     FALSE     FALSE FALSE     FALSE
    DM  DMcx Paralysis Renal Cancer LiverSevere  Mets   HIV
 FALSE FALSE     FALSE  TRUE   TRUE       FALSE FALSE FALSE
  TRUE FALSE     FALSE FALSE  FALSE       FALSE FALSE FALSE
 FALSE FALSE     FALSE FALSE  FALSE       FALSE FALSE FALSE
 FALSE FALSE     FALSE FALSE  FALSE       FALSE FALSE FALSE
 FALSE FALSE     FALSE FALSE  FALSE       FALSE FALSE FALSE
 FALSE FALSE     FALSE FALSE  FALSE       FALSE FALSE FALSE
```

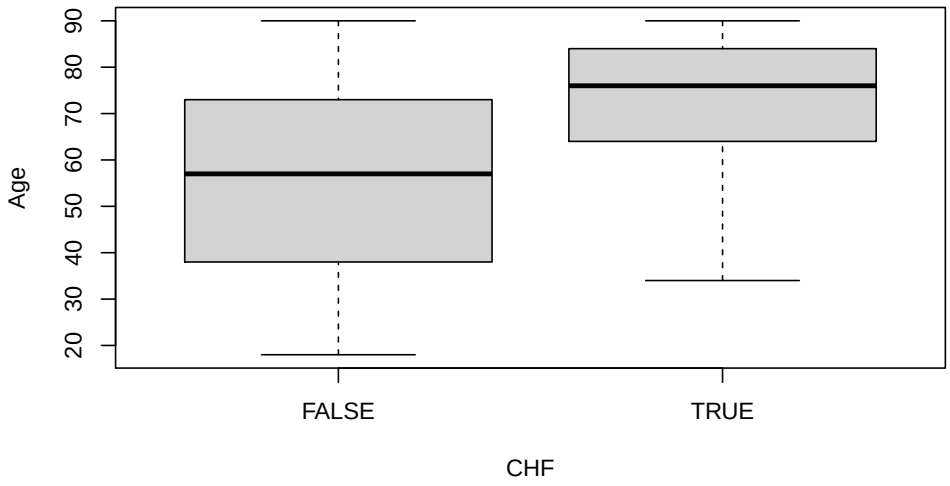
3.4. Example: National Hospital Discharge Database (NHDS)

The **nhds** package from CRAN provides anonymous 2010 data from this US-based survey. This examples show extraction of a single comorbidity of interest, and showing prevalence varying with age. There are about 130,000 adults in this database, not huge by some standards, but big enough for other algorithms to take minutes, whereas **icd** takes less than a second, which can be seen from the benchmarks in Section 4.

```
R> # start with all the adults
R> adults2010 <- nhds::nhds_adult()
R> # count patient-visits:
R> nrow(adults2010)

[1] 129242

R> # use icd to determine which has congestive heart failure:
R> adults2010[["CHF"]] <- comorbid_ahrq(adults2010)[, "CHF"]
R> boxplot(age_years ~ CHF,
R+         data = adults2010,
R+         outline = FALSE,
R+         ylab = "Age")
```



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 pseudocode in Section 1.5.1.

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; twenty per patient results in more

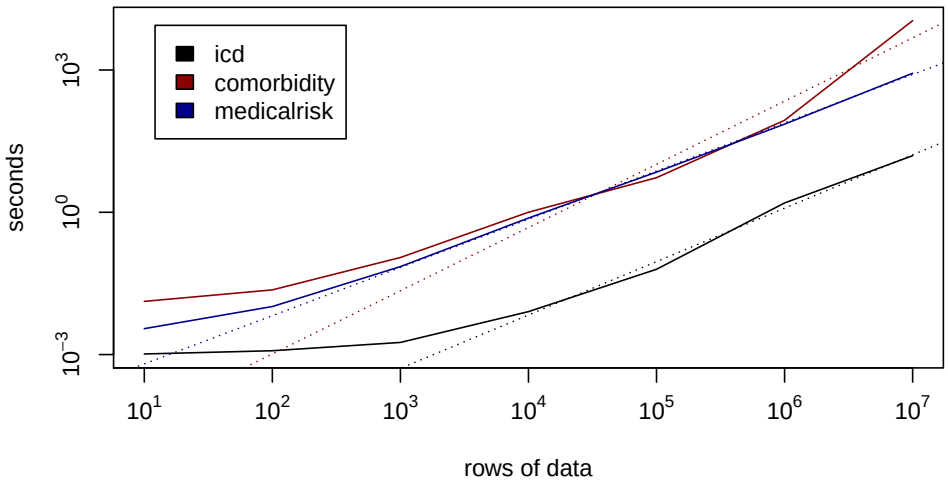


Figure 5: Performance comparison of comorbidity packages up to 10,000,000 rows, with 500,000 patient-visits and 20 comorbidities per visit. Models are fitted where the log-log relationship becomes linear, where rows > 1,000. Using a four-core eight-thread 3.40GHz CPU, 32GB RAM R 3.5.1 using Linux, kernel 4.15. **comorbidity** was run with and without parallel option, and the best strategy was chosen for each number of iterations.

work at each level: more code searches must be done, which emphasises the core problem solved by each algorithm.

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 fundamental 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.* (2011) provide such SAS source code, whereas other maps are only available in the form of tabulated data in journal articles.

Parsing original SAS code

For example, Quan *et al.* (2011) offer the following code for pulmonary disease in the ICD-9 Charlson map:

```
%LET DC6=%STR('4168','4169',
'490','491','492','493','494','495','496',
```

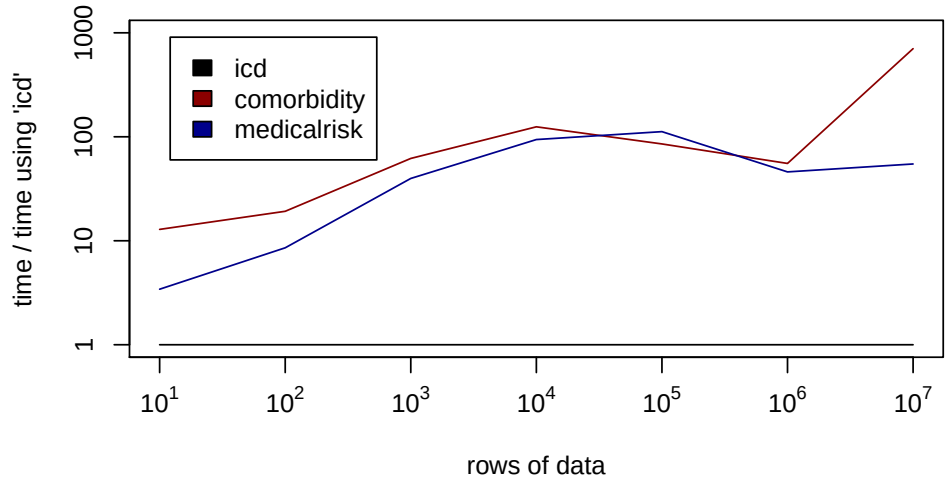


Figure 6: Relative speed-up using **icd** compared to the alternatives, using the same numbers of patient-visits and comorbidities as in Figure 5. The variation in the ratio of improvement **icd** offers is likely due to the varying bottlenecks experienced by each package as the problem scales.

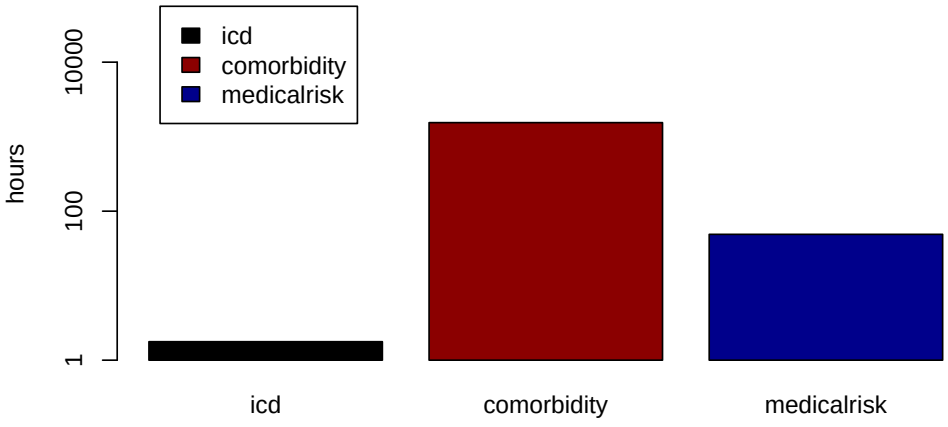


Figure 7: Predicted duration of computation for one hundred-million patient-visits, with twenty diagnoses per patient

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

```
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][[1]]

start   end
"490" "496"

R> icd9_sub_chapters[sc][[2]]

start   end
"500" "508"
```

497 is not a valid code itself. It could be a typo for any of the other combinations, of which four are completely different, yet valid codes:

```
R> explain_code(c("497", "479", "947", "974", "749", "794"), warn = FALSE)

[1] "Cleft palate and cleft lip"
[2] "Nonspecific abnormal results of function studies"
[3] "Burn of internal organs"
[4] "Poisoning by water, mineral, and uric acid metabolism drugs"
```

There are also many other ways the coder may have made the error other than a permutation of the intended code. 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> match("497", icd9_map_charlson)

[1] NA
```

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 codes in the map. This can be seen here, where two fictitious 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:

```
R> alice <- data.frame(id = "alice", icd9 = "49699")
R> comorbid_charlson(alice, return_df = TRUE)[["Pulmonary"]]
```

```
[1] TRUE
```

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 diseases 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:

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

```
[1] "243"    "243.0"  "243.00" "243.01" "243.02" "243.03"
```

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

```
[1] FALSE
```

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

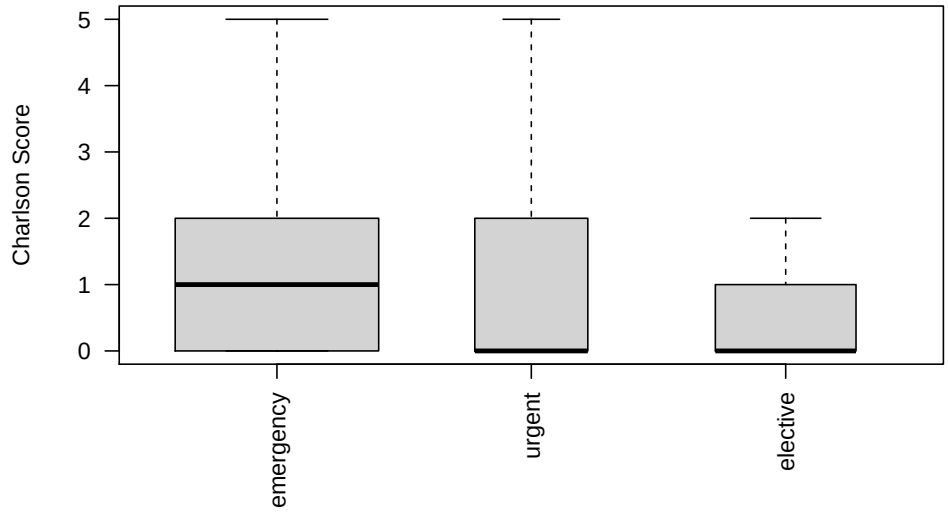
5.2. Other functions

Risk scores

Scoring systems to estimate risk are commonly used in the medical literature, one example being the Charlson score. It is calculated by adding up comorbidities for a patient, with more for particular diseases, and avoiding double-counting (e.g., diabetes, and diabetes with complications). It is a non-negative integer.

Example: Charlson score Extending the NHDS example, a user can show the relative risk of death on hospital admission in each category of patients:

```
R> adults2010_charl <- adults2010
R> adults2010_charl["charlson"] <- charlson(adults2010)
R> adults2010_charl["adm_type"] <- factor(adults2010_charl[["adm_type"]],
R+                                        c("emergency", "urgent", "elective"))
R> boxplot(charlson ~ adm_type,
R+         data = adults2010_charl,
R+         las = 2,
R+         varwidth = TRUE,
R+         outline = FALSE,
R+         xlab = NULL,
R+         ylab = "Charlson Score")
```



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.

```
R> is_valid(c("441", "441.0", "441.01", "XXX"))

[1]  TRUE  TRUE  TRUE FALSE

R> is_leaf(c("441", "441.0", "441.01", "XXX"))

[1] FALSE FALSE  TRUE FALSE

R> is_billable(c("441", "441.0", "441.01", "XXX"))
```

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

```
R> head(
R+   data.frame(code = children("441"),
R+              billable = is_billable(children("441"))))
```

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

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:

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

```
 [1] "441"   "4410"  "44100" "44101" "44102" "44103" "4411"  "4412"  "4413"
[10] "4414"  "4415"  "4416"  "4417"  "4419"
```

Several five-digit codes begin with 4410. `explain_code` condenses all these children to their common parent, before interpreting the ICD code:

```
R> explain_code(children("4410"))
```

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

What does each one mean?

```
R> explain_code(children("4410"), condense = FALSE)
```

```
[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"
```

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, and et al 2017)

, 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 multi-threaded solution. Once this was done, 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 possible 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), analogous 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. **icd** now includes the ICD definitions for France, Belgium, and the WHO in both English and French. Australian codes have been requested by several users, and are expected to be implemented next. Each new country requires careful evaluation of copyright claims over the data. The effect of using nationally or internationally coded ICD data on comorbidity results has not yet been evaluated. For ICD-10 calculations, at least, the strategy of working backwards from detailed to broad codes is likely to capture comorbidities reasonably well. ICD-9 codes in particular will require attention. A notable example is the minimal detail in coding HIV/AIDS in the USA, whereas the WHO and other countries define in more detail. The design of **icd** has always been to tolerate and do the best to anticipate the intended disease when finding the comorbidity it belongs to, and the international use of **icd** today has shown this has at least been helpful in practice, even if not formally validated.

Conversion between ICD-9 and ICD-10 is not direct, although General Equivalence Mappings exist for the USA's ICD-9-CM and ICD-10-CM schemes, which give best-effort conversions between them. The US National Institutes of Health (NIH) sponsored a Hackathon and the participants chose to work adding such a conversion, which will be included in a future release.

Eigen provided the best optimization of the specific sparse-dense matrix problem available at the time. Future work may examine replacing **Eigen** with a call to R BLAS, which may be configured to use GPU or alternative standard libraries which may use GPU processing, with

possible speed improvements at the expense of configuration complexity, and the cost of this abstraction.

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 across comorbidities. This was done informally by JW, with little impact on benchmark results.

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 work-flow.

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: Aaron Masino, Allan Simpao, Jorge Galvez and Jonathan Tan. Thanks also to the growing number of people around the world who have contributed code, including William Murphy, Anobel Odisho, Vitaly Druker, Patrick McCormick, the NIH 'hackathon' team, and many others.

References

Agency for Healthcare Research and Quality (2012). "Clinical Classifications Software for ICD-10 Data."

Agency for Healthcare Research and Quality (2018). "Elixhauser Comorbidity Software for ICD-10-CM Healthcare Cost and Utilization Project."

Bishop-Fitzpatrick L, Movaghar A, Greenberg J, Page D, DaWalt L, Brilliant M, Mailick M (2018). "Using Machine Learning to Identify Patterns of Lifetime Health Problems in Decedents with Autism Spectrum Disorder." *Autism Research*, **11**(8), 1120–1128. doi:10.1002/aur.1960.

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/c4rqj6.

CMS (2018). "Centers for Medicare and Medicaid Services: Research, Statistics, Data & Systems."

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/gdskrv.

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/ggkqvq.

Eddelbuettel D, Francois R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8). doi:10/gc3hqm.

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. doi:10/dg4f8d.

Evans MA (2011). "Evaluation of the CMS-HCC Risk Adjustment Model." *Technical report*, Center for Medicaid and Medicare Services.

Frank SM, Wick EC, Dezern AE, Ness PM, Wasey JO, Pippa AC, Dackiw E, Resar LM (2014). "Risk-Adjusted Clinical Outcomes in Patients Enrolled in a Bloodless Program." *Transfusion*, **54**(10), 2668–2677. ISSN 1537-2995. doi:10/ggv67x.

Free Software Foundation (2007). "GNU General Public License."

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

Guennebaud G, Jacob B, et al (2017). "Eigen."

Hand RK, Murphy WJ, Field LB, Lee JA, Parrott JS, Ferguson M, Skipper A, Steiber AL (2016). "Validation of the Academy/A.S.P.E.N. Malnutrition Clinical Characteristics." *Journal of the Academy of Nutrition and Dietetics*, **116**(5), 856–864. ISSN 2212-2672. doi:10.1016/j.jand.2016.01.018.

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/cxtb96.

Kiyota Y, Corte AD, Vieira VM, Habchi K, Huang CC, Ratta EED, Sundt TM, Shekar P, Muehlschlegel JD, Body SC (2017). "Risk and Outcomes of Aortic Valve Endocarditis among Patients with Bicuspid and Tricuspid Aortic Valves." *Open Heart*, **4**(1), openhrt–2016–000545. ISSN 2053-3624. doi:10.1136/openhrt-2016-000545.

Lane D (2018). *The Identification and Management of Sepsis in the Prehospital Setting*. Thesis.

Lane DJ, Lin S, Scales DC (2019). "Classification versus Prediction of Mortality Risk Using the SIRS and qSOFA Scores in Patients with Infection Transported by Paramedics." *Prehospital emergency care: official journal of the National Association of EMS Physicians and the*

National Association of State EMS Directors, pp. 1–8. ISSN 1545-0066. doi:10.1080/10903127.2019.1624901.

Lee DS, Austin PC, Rouleau JL, Liu PP, Naimark D, Tu JV (2003). "Predicting Mortality among Patients Hospitalized for Heart Failure: Derivation and Validation of a Clinical Model." *JAMA*, **290**(19), 2581–2587. ISSN 0098-7484. doi:10/b89frv.

Martin N (2017). "Lost Mothers."

McCormick P, Joseph T (2016). *medicalrisk: Medical Risk and Comorbidity Tools for ICD-9-CM Data*. R package version 12.

Mundell B, Kremers HM, Visscher S, Hoppe KM, Kaufman KR (2016). "Predictors of Receiving a Prosthesis for Adults with Above-Knee Amputations in a Well-Defined Population." *PM & R : the journal of injury, function, and rehabilitation*, **8**(8), 730–737. ISSN 1934-1482. doi:10.1016/j.pmrj.2015.11.012.

Odisho AY, Etzioni R, Gore JL (2018). "Beyond Classic Risk Adjustment: Socioeconomic Status and Hospital Performance in Urologic Oncology Surgery." *Cancer*, **124**(16), 3372–3380. ISSN 0008-543X. doi:10.1002/cncr.31587.

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.

Pulitzer (2018). "Finalist: Staff of ProPublica."

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/d5mv98.

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. doi:10/drnncx.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Simpao AF, Ahumada LM, Rehman MA (2015). "Big Data and Visual Analytics in Anaesthesia and Health Care." *British Journal of Anaesthesia*, p. 552. ISSN 0007-0912, 1471-6771. doi:10/f7qk5r.

Tsugawa Y, Jena AB, Figueroa JF, Orav EJ, Blumenthal DM, Jha AK (2016). "Comparison of Hospital Mortality and Readmission Rates for Medicare Patients Treated by Male vs Female Physicians." *JAMA Internal Medicine*. doi:10.1001/jamainternmed.2016.7875.

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/djgn5z.

Vermont Department of Health (2016). "Vermont Hospital Discharge Data."

WHO (2018). "International Classification of Diseases."

Affiliation:

Jack O. Wasey
Children's Hospital of Philadelphia
3401 Civic Center Blvd. Philadelphia, PA 19104, USA
E-mail: 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
E-mail: 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
E-mail: 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