

LobbyView: Firm-level Lobbying & Congressional Bills Database*

In Song Kim[†]

August 30, 2018

Abstract

A vast literature demonstrates the significance for policymaking of lobbying by special interest groups. Yet, empirical studies of political representation have been limited by the difficulty of observing a direct connection between politicians and interest groups. This article introduces **LobbyView**, a comprehensive lobbying database that is based on the universe of lobbying reports filed under the Lobbying Disclosure Act of 1995. **LobbyView** bridges two distinct observable political behaviors with regard to congressional bills: (1) sponsorship by politicians, and (2) reported lobbying by interest groups. It also allows researchers to identify political actors and their lobbying activities based on standardized firm- and industry-level identifiers to facilitate systemic research on lobbying and merging with external datasets. Finally, we develop an API (Application Programming Interface) that enables researchers to bulk download the massive amount of unstructured data in standard formats so that they can conduct further analyses using preferred statistical software.

*Financial support from the National Science Foundation is acknowledged (SES-1264090 and SES-1725235).

[†]Associate Professor, Department of Political Science, Massachusetts Institute of Technology, Cambridge, MA, 02139. Email: insong@mit.EDU, URL: <http://web.mit.edu/insong/www/>

1 Introduction

This document currently contains only some technical details of the database. We will soon update this paper to provide a full description of the methods used for constructing the database. Specifically, we will show 1) how we disambiguate interest group names using natural language processing (NLP) as well as collaborative filtering, 2) how the complex relational structure is stored using PostgreSQL and Elasticsearch, and 3) the scalability of our methods for bill number and bill title matching described in Section 2. Moreover, we will conduct several descriptive and statistical analyses to demonstrate the scope and quality of the lobbying database. Until then, we refer readers to the following two papers for an introduction to **LobbyView**.

References

- Kim, In Song. 2017. “Political Cleavages within Industry: Firm-level Lobbying for Trade Liberalization.” *American Political Science Review* 111 (1): 1–20.
- Kim, In Song, and Dmitriy Kunisky. 2018. “Mapping Political Communities: A Statistical Analysis of Lobbying Networks in Legislative Politics.” Working paper available at <http://web.mit.edu/insong/www/pdf/network.pdf>.

2 Identifying Bills and Missing Congress Numbers

Identifying congressional bills in lobbying reports is difficult because bill numbers are repeated across Congresses, and often do not appear directly annotated with Congress numbers in lobbying reports. Using the report filing year to guess the Congress often leads to erroneous matches, because reports filed at the beginning of a new Congress tend to include disclosures of lobbying activities from the previous year (and therefore, if a new Congress has begun recently, from the previous Congress as well). For example, consider the following lobbying report filed by **GOOGLE, INC.** in 2013. It reads:

Monitor legislation regarding online privacy including Safe Data Act (H.R. 2577, S. 1207) and Do not track proposals (H.R. 654). Monitor any Congressional or Administration efforts to impose privacy laws on search engines. Monitor Spectrum acts (S. 911, H.R. 2482).

Figure 1: First Quarter Report by **GOOGLE, INC.** in 2013

A naive guess would be that the bill H.R. 2577 refers to a bill from the 113th Congress, because the report was filed in 2013. However, it is clear from the report that this is a bill from the 112th Congress, the “SAFE Data Act.” We use the following strategies to mitigate this problem and correctly identify Congress session numbers under various circumstances.

1. **Bill Number Search:** We first identify bill numbers (e.g., H.R. 2577 above) using regular expression search in the report text. In the above example in Figure 1, our algorithm would identify bill numbers H.R. 2577, S. 1207, H.R. 654, S. 911, and H.R. 2482. Note that all of these bills are from the 112th Congress rather than the 113th.
2. **Congress Identification:** Given a bill number found in a specific issue text (a section of the lobbying report), we attempt to identify the most likely Congress to which that bill would belong using other text around the bill number. We consider a range of candidate Congresses extending backwards from the Congress containing the year that the lobbying report was filed. By default, we consider the three preceding Congresses; in the above example, therefore, we would consider the 113th, 112th, and 111th Congresses. We then retrieve the bills having the same number as the given bill from each of these Congresses (omitting the Congresses that do not have a bill of that number), and compute a bag-of-words representation (after a tokenization and stopword filtering pipeline) of each of those bills, producing vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ representing the n candidate bills. We also compute the same representation of the text around the mention of the bill number in the lobbying report, producing a vector \mathbf{w} representing that text. We then choose the Congress number by maximizing the cosine similarity between the \mathbf{v}_i and \mathbf{w} , choosing bill i^* with index given by

$$i^* = \operatorname{argmax}_{1 \leq i \leq n} \frac{\mathbf{v}_i^\top \mathbf{w}}{\|\mathbf{v}_i\| \|\mathbf{w}\|}. \quad (1)$$

If no bill having the same number exists in the entire range of Congresses we consider, we simply guess that the bill comes from the Congress of the year the lobbying report was filed.

3. **Congress Propagation:** If we successfully find a match for a Congress, it may be propagated to the other bills mentioned in the lobbying report, since, being scheduled on a quarterly basis, lobbying report will almost always only mention legislation from a single Congress. If different bills in a lobbying report disagree on the best-matching Congress, a majority vote may be taken, but this rarely occurs in practice.
4. **Bill Title Search:** Bills are sometimes only referred to by titles or alternate names. To account for this, we clean and tokenize the specific issue sections of the lobbying report, and perform a text matching operation against a table of bill titles. For instance, this operation would identify “Safe Data Act” in our previous example, even if the bill number H.R. 2577 were not mentioned.
5. **Bill Range Expansion:** It is also common for bills with nearby numbers to be related, and for lobbying reports to refer to ranges of bills when lobbying all of them at once. For

instance, a lobbying report filed by MATTEL, INC. in 2002 contains the following text:

“H.R.3009, Trade Act of 2002. Certain miscellaneous tariff bills to suspend the rates of duty on certain toy-related articles (H.R.4182-4186; S.2099-2103). WTO market access negotiations for non-agricultural products Port and border security measures”

Figure 2: Midyear Report by MATTEL, INC. in 2002

Therefore, if we find two bill numbers that are close (by default, we take this to mean that they share the same prefix and their numbers differ by at most 10), then we consider all other bills with numbers in between as also being lobbied in the same report. For instance, the pattern “H.R. 4182-4186” in the excerpt shown in Figure 2 would be expanded into bills H.R. 4182, H.R. 4183, H.R. 4184, H.R. 4185, and H.R. 4186, all of which we would consider lobbied on by MATTEL, INC.

3 Technical Details

This document concerns the code in the `/trade/code/database` directory of our repository, which sets up and provides access to a system of databases (running on `SQLite` and the `Whoosh` text indexing library) that store relationships between bills, their lobbyists, and various other related pieces of data.

3.1 Dependencies

The table below is a summary of the packages on which the database depends, along with a short summary of the functionalities that they provide (further information and documentation can be easily found on their PyPI (Python Package Index) pages). The packages are roughly grouped by their function (database, parsing, etc). For the purposes of actual deployment, the file that manages these packages is `database/REQUIREMENTS`.

Package	Description
<code>SQLAlchemy</code>	Basic Python bindings and model representations for SQL-type databases.
<code>Elixir</code>	Higher-level abstractions for dealing with SQL-type databases, extending the functionality that <code>SQLAlchemy</code> makes available.
<code>Whoosh</code>	A library for creating full-text index databases that are searchable with reasonable efficiency (if in the future better efficiency here is needed, there are non-Python packages that can do a better job). SQL is not good, generally speaking, for full-text search, hence the necessity of this for the bill CRS summaries and lobbying report specific issue texts.
<code>BeautifulSoup</code>	Convenient library for parsing XML and HTML data into Python objects, although when speed becomes an issue there are faster but less convenient (and more code-verbose) alternatives, such as <code>xml.etree</code> in the core Python library.
<code>nltk</code>	The “natural language toolkit” for Python, providing tools for tokenizing and statistically analyzing English-language texts.
<code>path.py</code>	Convenience tools to make dealing with the filesystem easier from Python.
<code>python-dateutil</code>	Convenience tools for dealing with datetimes and time ranges.

Before going any further, you will need to install all of these packages, which can be conveniently done through the `REQUIREMENTS` file, by running the following command in the shell (you will need the `pip` utility for Python package management):

```
> pip install -r REQUIREMENTS
```

Also, there are a few subpackages to install for the `nltk` package. In a Python interpreter, run the following:

```
>>> import nltk
>>> nltk.download()
```

That will open up an interface for downloading extras/packages for `nltk`. Then, you should go into the Corpora tab and install the packages `stopwords` and `wordnet`. If no errors arise during this installation procedure, it is safe to proceed to the next steps.

4 Getting Started

4.1 Configuration

All of the necessary configuration for the database system can be done by modifying the variables in `general.py`. The variables are listed below along with the effect that they have on the database creation process. Realistically speaking, to get things working locally you should just change `DATA_DIR` to something that is not Della-specific. Everything else should be (more or less) good to go, assuming no large repository reorganizations have happened.

Variable	Description
<code>TEST_MODE</code>	Intended as a testing mode for bill detection and related algorithms, but this is not yet complete, so avoid setting this to <code>True</code> before taking a look at the testing code.
<code>CONGRESSES</code>	The range of congresses that all processes will be concerned with (note that in Python, the <code>range(x, y)</code> syntax gives the numbers $x, x + 1, \dots, y - 1$, not including y).
<code>OUTPUT_DIR</code>	The directory for generic outputs of analytics scripts.
<code>ROOT_DIR</code>	The <code>database</code> directory (these directories are given relative to the <code>trade/code</code> directory).
<code>DATA_DIR</code>	The directory used for storing databases, which can be totally separate from the code directories. For instance, on Della it is useful to put this under <code>/tigris</code> since it requires lots of storage space.
<code>LOBBY_REPO_DIR</code>	The location of the lobby repository (parallel to the <code>trade</code> repository in the current setup).
<code>CLIENT_NAME_MATCHES_FILE</code>	The file containing the client name filtering matches (i.e. the output of Josh's script).

4.2 Initialization

Installing the databases is (or should be) very easy, just go to the `trade/code` directory and run the command `sudo python -m database.setup`. This will probably take a very long time to run from scratch—possibly up to several days.

5 Working with the Database

5.1 Starting and Ending Sessions

In order to get started with a database session, navigate to the `trade/code` directory, and run the following sequence of commands in the Python interpreter:

```
>>> import database.general
>>> from database.bills.models import *
>>> from database.lda.models import *
>>> from database.firms.models import *
>>> database.general.init_db()
```

When finished, the following command will safely close the database without accidentally permanently writing any changes that may have been made to the data:

```
>>> database.general.close_db(write=False)
```

If you are in fact correcting errors in the database or otherwise performing operations that cause changes that you would like permanently registered, then just change the above to instead pass the argument `write=True`.

5.2 Basic Database Objects and Relationships

To see what objects are stored in the various databases, look at the `models.py` files in the directories `bills`, `lda`, and `firms` (the imports described above are what give you access to all of these classes). Each class has some fields, where are available to access on any object of the class. The system is best clarified with an example: consider the case of bill objects, which are represented by the `Bill` class. This class, like any other, has an `id` field that is its *primary key*, i.e. the value of this field uniquely identifies a `Bill` object. For bills, the `id` is a string of the form `110_HR7311`, where `110` is the number of the congress to which this bill belongs, and `HR7311` is the bill number. To retrieve a particular bill by its `id`, use the following snippet:

```
>>> b = Bill.query.get('110_HR7311')
```

Once this command completes, the object `b` will have all of the fields listed under `Bill` in the file `bills/models.py`. So, for instance, we can get the date that the bill was introduced with `b.introduced`, get its CRS summary text with `b.summary`, and so forth. Any field inside `Bill` that is initialized as `Field(ABC)` where `ABC` is some text (example possible values are `Integer` for integer fields, `Unicode(L)` for a string field of maximum length `L`, or `DateTime` for a date/time field) is accessed in this straightforward way.

Other fields are registered as `ManyToMany(ABC)`, `ManyToOne(ABC)`, or `OneToMany(ABC)`, where `ABC` is now the name of some other model. These fields contain references to one or more instances of some other model. For instance, in `Bill`, the field definition

```
titles = OneToMany('BillTitle')
```

indicates that each bill has one or more (hence `Many`) associated objects of class `BillTitle`, which are its titles. In `BillTitle`, we see the field giving the reverse relation,

```
bill = ManyToOne('Bill')
```

which indicates that many `BillTitle` objects can share the same `Bill` object (for some intuition, think of a `OneToMany` field as a “my children” relation, and of a `ManyToOne` field as a “my parent” relation).

Thus, if `b` is a `Bill` object, then `b.titles` will give an iterable (effectively a Python list, for all basic purposes) containing all the titles of `b`. Conversely, if `t` is a `BillTitle` object, then `t.bill` is the `Bill` object to which the title belongs.

The last possibility of these more complex relationships is a `ManyToMany` field, which as its name suggests creates a generic relation between two object types (where neither object plays the “child” or “parent” role). For example, we see in `Bill`,

```
terms = ManyToMany('Term')
```

and in `Term`,

```
bills = ManyToMany('Bill')
```

which means that for a bill `b`, looking at `b.terms` gives all of the terms that that bill is classified under, and for a term `t`, looking at `t.bills` gives all bills under that term.

5.3 Filtering Operations

The more sophisticated and interesting sorts of queries that are possible are those that involve not just fetching particular bills or other objects and examining their relationships, but also involve filtering sets of objects by useful criteria.

Example: filter by columns of each Model

```
>>> reports = LobbyingReport.query.\
        filter_by(year=2011)
```


returns all lobbying reports filed in 2011

Example: filter by membership in at least one ManyToMany related table

```
>>> trade = LobbyingReport.query.\
    filter(LobbyingReport.issues.any(LobbyingIssue.code.\
    in_(['TRADE (DOMESTIC/FOREIGN)'])))
>>> trade.count()
52418
```

returns all lobbying reports that at least has 'TRADE (DOMESTIC/FOREIGN)' as one of issues lobbied.

5.4 Full-Text Indices

Two types of data items are duplicated in a separate full-text index database to facilitate more efficient searching: the CRS summary text of each bill, and the text of each lobbying report specific issue. The code concerning the creation and access of these indices is found in the files `bills/ix_utils.py` and `lda/ix_utils.py`, respectively.

The primary useful methods, in turn, for accessing these indices are `summary_search` and `issue_search`, in the above two files respectively. These both take one required argument, called `queries_list`, which is a list of the queries (as strings) to make to the full-text index. They also have two optional boolean arguments, `return_objects` and `make_phrase`, which default to `False`. Setting `return_objects` to `True` will return a collection of `Bill` or `LobbyingSpecificIssue` Python objects rather than just their `id` values. Setting `make_phrase` to `True` will make each query into a phrase that is searched for a single unit, rather than separately searching for each word (as in the difference when searching Google for *red cat running* versus “*red cat running*” in quotes).

In `lda/ix_utils.py`, there is an additional method exposed for using the index that is called `get_bill_specific_issues_by_titles`, which is a simple special case of `issue_search` that searches for all of the titles of a particular bill in the specific issues, used in the database construction process to find the bills mentioned by title in specific issues.

A simple example of using these indices to find bills pertaining to a particular textually-distinguished subject (trade-related bills in our case) can be found in `analytics/lobbied_bills_data.py`. We define a list of queries on our bills in the following way:

```
from database.analytics.bill_utils import *
bill_queries = [
    u'trade barrier',
```

```

    u'tariff barrier',
    ...
    u'uruguay round',
    u'harmonized tariff schedule'
]

```

Then, to get the id's of the bills that contain one of these phrases, we do this:

```

query_bill_ids = database.bills.ix_utils.summary_search(
    bill_queries,
    make_phrase=True,
    return_objects=False
)

```

This returns a list of id's as strings. If we wanted the corresponding `Bill` objects instead, we could instead pass the argument `return_objects=True`. Note that here it is important that we use `make_phrase=True`, since otherwise the query `'uruguay round'` would match all bills that contain both the word `uruguay` and the word `round`, not necessarily together, which is not what we want.

A simple example of using these indices to find lobbying reports that contain a particular phrase,

```

from database.analytics.lda_utils import *
reports = lda_issue_search(
    ['Free trade agreements with South Korea']
)

```

5.5 Calculating Herfindahl Indices for Industry Clients Belong to

We provide a tool to measure the size of each firm in relation to the industry. Herfindahl index measures the levels of competition among firms (clients) within the same industry.

5.5.1 `herfindahl.py` (in `lobby/code/hfcc`)

Given lobbying database containing firm, LDA, and bill information,

1. Pulls firm-level financial and LDA report information from lobbying database
2. Computes Herfindahl indices
3. Outputs rows with firm information (sorted by industry as identified by NAICS2), industry Herfindahl index, and lobbying information for firm.

To run: From lobby/code, type "python -m hfcc.herfindahl" No additional parameters needed.

5.5.2 herfindadd.py (in lobby/code/hfcc)

Given output (csv) files generated by herfindahl.py,

1. Adds indicator showing whether firm lobbied on at least one trade issue
2. Adds firm-level compustat financial data
3. Generates (in addition) new csv files with industry-level information in rows

To run: Ensure output files from herfindahl.py are in lobby/code From lobby/code, type "python -m hfcc.herfindadd" On-screen documentation will detail additional parameters that are needed.

Example

```
python -m hfcc.herfindadd namerica naics -s 1996 -e 2011
```

runs the script for the North America files, using NAICS (rather than SIC), starting from 1996 and ending in 2011 (herfindahl.py generates one file per year per classification system (NAICS / SIC).)