Linking & Exploring Open Government Data: Business Investment and Grants

COMP3003 Report

William Vigolo da Silva
BSc Hons Computer Science with Year in Industry
psywv@nottingham.ac.uk
14279225

May 4, 2020

Contents

1 Introduction

Many governments and their institutions regularly publish information due to legal requirements of freedom of information and policies regarding open data. The purposes of such policies are often to improve and maintain transparency in government.

Transparency is part of the principle of open government (Lathrop and Ruma 2010), and is considered to strengthen democracy, regulate government behaviour, and promote government efficiency (Schauer 2011).
Open government aims to encourage citizen participation in overseeing the actions of government and their officials - a tradition that can be traced back to ancient Greece (Dornum 1997).
This idea that data should be pro-actively published and freely available is known as the principle of *open data* ("Open Definition 2.1" 2019). This transfers some of the responsibility of oversight; governments must make available facts that could reveal misbehaviour or areas that could be improved.

Open government data often includes topics such as election statistics, department budgets & expenditures, and information about significant individuals within organisations. These topics are useful in identifying undesirable behaviour like corruption and nepotism.
Other areas such as statistics in crime, justice, and healthcare could be used to check that a government is acting in the best interests of its citizens.

There are some unfortunate limitations in the way that governments publish open data. They do not properly follow the principles of open data, failing to use open formats and formats that are not machine-readable.
This makes the data harder to use by both citizens interested in finding out facts about their government, and by researchers looking to perform quantitative research on the data available.

Consequentially, analysing open government data requires a few extra steps, including:

- **Collecting** datasets from multiple sources.
- **Cleaning** data that is invalid or incomplete
- **Linking** data from the different sources, i.e. combining the available information.

2 Motivation

This project's goal is to explore the value of open data and government transparency.
It discovers how existing open data and existing government data publishing platforms can used to perform specific research and produce useful findings.

My intention is to apply the full process needed to conduct in-depth data analysis and address the challenges that one faces while working with open data. Outcomes of this effort will include insights into ways to improve government publishing of open data in order to enable a range of use scenarios and potential applications.
The focus is on open data published by the government of the United Kingdom.

3 Related work and resources

3.1 Papers

Shadbolt et al. (2012): Open data from the UK government was collected, migrated, and linked into a 'semantic web'. Use of standard technologies like the Resource Description Framework data

format and Sparql query language enabled a variety of interesting analyses and visualisations, including exploring data point variations over regions and time. A user interface was created that allowed less technical users to explore data using only their spreadsheet skills.
This paper is a valuable reference into the process needed for transforming a collection of datasets into a database that can be analysed using data analysis methods and to which machine learning methods can be applied.

Winkler (2006): An overview of existing methods and research in the area of record linkage. Records from different datasets need to be linked together in order for the relationships between entities to be explored, and to provide extra information about individual entities, such as businesses and grants.
The methods described in the paper were helpful through the process of linking records in the dataset, particularly in describing algorithms and heuristics for record comparison.

Elmagarmid, Ipeirotis, and Verykios (2006): Techniques for detecting duplicate records in a database are explored and explained in detail.
Among other things, duplicate records in the datasets used in this project need to be merged in order for the data analysis to produce high quality and accurate results. This paper provides a valuable reference for methods used during this process, including Q-gram matching.

3.2 Data sources

UK Research and Innovation[1] (UKRI): A non-governmental organisation that funds research and innovation in the UK through grants. It is an umbrella organisation for several 'research councils' in several industries and areas. Data on funding from each council is made publicly available through the Gateway to Research[2] database.
This provides details on each project, organisations & individuals involved, and catalogues research outcomes as summarised by the researchers. Relevant research papers produced during the project are also listed.
This is the main dataset that will be used throughout this project, as it includes most information relevant to the intended research.

Data.gov.uk[3], European Data Portal[4], UK Transparency and FOI Releases[5], Data.gov[6]: catalogues datasets released by government institutions of the United Kingdom, European Union, and United States of America respectively.
These could be used to provide additional context to the UKRI dataset, if needed. Should time allow, research data from other regions such as the United States may be explored, in a similar manner to that of the United Kingdom.

3.3 Software

PostgreSQL[7]: A database management system. As I have prior experience with it, I intend to use it throughout this project to manage any databases created throughout the project.

Stanford Natural Language Processor (Manning et al. 2014): A toolkit for natural language processing. This software could be used to extract data points from textual documents, or

[1] https://www.ukri.org
[2] https://gtr.ukri.org
[3] https://data.gov.uk
[4] https://europeandataportal.eu
[5] https://gov.uk/search/transparency-and-freedom-of-information-releases
[6] https://data.gov
[7] https://www.postgresql.org

during analysis of grant descriptions, for example.

WorldMap[8]: An open-source platform for overlaying data on a map, which could be used to visualise data points over different regions.

NodeXL[9] and Gephi[10] (Bastian, Heymann, and Jacomy 2009): graph visualisation and analysis tools. These tools could be used to explore the relationship between entities in collected datasets. Gephi supports attaching to a database management system, such as PostgreSQL. This makes it easier to use with datasets stored in a single database, as is the case in this project.

[8] https://worldmap.harvard.edu
[9] https://nodexl.com
[10] https://gephi.org

4 Description of work

The aim is to leverage computational techniques and open government data about investment in research & innovation and businesses in different development stages, to determine the relationship between levels of investment and the impact on businesses within various sectors. Such analysis is of interest to companies that analyse markets, e.g. to support business decision making. This includes The Decision Project[11], who expressed an interest in being involved in the project.

4.1 Research questions

The following questions were identified as potential interesting analysis from preliminary research and initial insights gained by exploring the available data.

- What is the structure of the ecosystem of publicly funded research - how does it change over time?
- What are the significant factors that influence collaboration between organisations?

4.2 Technical approach

This project explores open government data from the United Kingdom relating to grants and investment in research and development. Specifically, it uses the Gateway to Research database from UK Research and Innovation.

This dataset is aggregated, normalised, and linked so that all available information can be used during the project's research. This involves translating the schema of this dataset into a new schema which encapsulates both existing and new data. This schema is then implemented using the PostgreSQL database management system.

Focus is placed on applying existing data analysis and machine learning techniques on these datasets which contain both structured and unstructured data.
Network analysis methods are used to identify the relationships between entities in the datasets.
Correlation analysis is performed to answer questions based on historical data.
Visualisations are created to show the identified relationships and other findings.
Machine learning methods such as decision trees and artificial neural networks are applied to perform predictive analyses required by some research questions. This requires identifying the most significant attributes within the dataset that contribute best to answering the questions.
Each method applied is evaluated to identify each performs with respect to the accuracy or usefulness of findings or predictions. For example, a predictive model for the popularity of a subject over time could be tested against historical data.

[11] https://www.decisionplatform.io

5 Methodology

(1) **Data aggregation**
   (i) Datasets exported from the UKRI Gateway to Research database
   (ii) Data points extracted and normalised where necessary
   (iii) Design a database schema to accommodate storage of data attributes and metadata
   (iv) Import all records into a single relational database

   In order to link and ultimately analyse the data, each dataset was parsed and exported from their original format into a single database. Individual data points are be extracted to normalise the data, such that it can be stored in a relational database.
   A database schema is needed so that records from all datasets are available in a single relational database. This enables relationship analysis and other analyses.
   Focus was placed on data that is well-formatted and machine-readable. Besides numeric and categorical data, text was also processed.

(2) **Linking datasets**
   (i) Discard low-quality or invalid records
   (ii) Research appropriate linking & de-duplication methods
   (iii) Merge duplicate records
   Some records refer to non-existent entities, such as unnamed businesses listed as participants in research projects. These records are unusable for the purposes of this project, so they need to be identified and removed. These records are identifiable by their similar names and lack of other details.
   Datasets can refer to the same business or grant by different names, causing duplicate entries in the database. In such cases similarity comparison algorithms is be applied to automatically identify the single entity which is being referred to. Some manual work was performed to clean up remaining duplicates.
   The data was sampled randomly to test the efficacy of linking & de-duplication methods.

(3) **Analysing data**
   (i) Research and test useful tools
   (ii) Perform network analysis to identify relationships
   (iii) Analyse data to will to answer the desired questions
   (iv) Develop software to perform analysis, visualisation, and learning
   (v) Create visualisations to show relationships and other results

   Research was performed to identify tools that could be useful in analysis or visualisation. Software was developed to apply existing analysis and visualisation tools to the collected data.
   An initial network analysis was performed on the linked data to identify and visualise some interesting relationships between the entities.
   Several algorithms were then applied to explore these relationships. For network analysis, this included degree distributions and clustering coefficients. Some statistical analysis of relationships were also applied.

(4) **Evaluating results**
   (i) Evaluate outcomes of analysis using metrics appropriate for each method

(ii) Make conclusions to answer the desired questions
Relationships identified through network analysis were evaluated to explore how well they answer the research questions.

6 Data acquisition and preparation

6.1 Data aggregation

The UKRI Gateway to Research (GtR) service is available online[12]. The service also provides APIs that allow programmatic access to the database, which are documented on the Gateway to Research website[13].
A shell script was written that uses the GtR-2 API. As the API restricts downloading to 100 records per query, the script downloads all available pages into individual XML files. In order to avoid overtaxing the service, the script pauses for some time after each page is downloaded. Initially developed for downloading only organisation records, once fully functioning the script was extended to support downloading all types of records available through the API.

For example, the first page of organisations stored in the GtR database can be accessed at the URL https://gtr.ukri.org/gtr/api/organisations. Though one might have to 'View page source' in an internet browser, one can see that a series of records have been returned in XML format, for example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ns2:organisations ...>
    <ns2:organisation ns1:created="2020-01-28T17:07:16Z" ...>
        <ns1:links>
            <ns1:link ns1:href="https://gtr.ukri.org:443/gtr/api/projects/..."
                ns1:rel="PROJECT"/>
            <!-- More project links -->
            <ns1:link ns1:href="https://gtr.ukri.org:443/gtr/api/persons/..."
                ns1:rel="EMPLOYEE"/>
            <!-- More employee links -->
        </ns1:links>
        <ns2:name>Tsinghua University</ns2:name>
        <ns2:addresses>
            <ns1:address
                ns1:created="2020-01-28T17:07:15Z"
                ns1:id="41B25E49-A52C-46D5-A83B-8CF1723184C7">
                <ns1:line1>Tsinghua</ns1:line1>
                <ns1:line2>Hai Dian District</ns1:line2>
                <ns1:postCode>100084</ns1:postCode>
                <ns1:region>Unknown</ns1:region>
                <ns1:type>MAIN_ADDRESS</ns1:type>
            </ns1:address>
        </ns2:addresses>
    </ns2:organisation>
</ns2:organisations>
```

Downloading all records involves parsing this file to identify how many pages of records are available, then performing a query to download each one in turn. This process needs to be repeated for each type of record: organisations, individuals, projects, and project outcomes.

The shell script was written in Bourne Again Shell (Bash)[14] for the GNU/Linux operating

[12] https://gtr.ukri.org
[13] https://gtr.ukri.org./resources/api.html
[14] https://www.gnu.org/software/bash

system. Additional command-line tools were used, including: xmllint from libxml2[15] for XML manipulation, GNU Awk[16] for text processing, and curl[17] for downloading web pages.
Since these software packages are free, open source, and commonly used, the research can be relatively easily reproduced and results verified. The tools support automation of the main process downloading a series of files from the internet.

In order to merge the downloaded XML files an auxiliary program was created.
Each page of downloaded records is an individual XML file, with records contained in a single wrapping XML element. As an example, pages of organisation records are in the following structure:

```xml
<?xml version="1.0" encoding="utf8"?>
<ns0:organisations xmlns:ns0="...">
    <ns0:organisation ns1:created="...">
        <!-- Record information -->
    </ns0:organisation>
    <!-- repeat ns0:organisation -->
</ns0:organisations>
```

In order to combine the pages, the program collects the inner records, e.g. `ns0:organisations`, into the outer `ns0:organisations` of the first record.
This could not be performed by the main script itself, as the XML tool being used, xmllint, did not support this combination of records. Python was used instead, as I am similarly familiar with it, it is cross-platform, free and open-source, and it comes with a fully-featured library for manipulating XML data in that manner that was needed.

The source code of the main download program is provided in the appendix section ukriDownload, and the auxiliary combing script in mergeXML.

6.2 Data importing

Aggregated data is imported into a single relational database management system, which will enable analysis through queries composed in the Structured Query Language (SQL).
The schema of the database is designed to be *normalised*, such that each atomic piece of information is stored in an individual field. This minimises data duplication - particularly helpful when handling large databases such as in this project - as well as enabling analysis of individual components of records.

All aggregated data is kept in the database so that it can be used throughout the project. As a result some information is duplicated, as records exist in both the original dataset and the cleaned dataset. In order to differentiate between the sources of data, tables containing data from the Gateway to Research database are prefixed with 'gtr'. All other tables contain data formed from this project.

The design for data from the Gateway to Research database is based partially on the contents of the GtR API manual (*Gateway to Research Api 2* 2018), which explains the contents of the records returned by the API.
In occasions where the manual provided inaccurate or incomplete information, the XML schema of records and some exported records were visually inspected to identify how best to normalise

[15] http://xmlsoft.org
[16] https://www.gnu.org/software/gawk
[17] https://curl.haxx.se

each type of record. For example, the XML schema for individuals is available through the REST API[18].

The final database design is visualised figure 16, with each table and field annotated in the appendix section Database schema manual. It was applied to the PostgreSQL database management system, and is implemented in the SQL file provided in appendix section setup.sql.

PostgreSQL has good support for importing structured XML data, though the structure of the XML records must be mapped to the relational and normalised structure of the database schema.
An SQL procedure was produced for the importing of each record type: organisations, individuals, projects, and project outcomes. These are provided in the appendix: importGtrOrgs.sql, importGtrPersons.sql, importGtrProjects.sql, and importGtrOutcomes.sql.

The SQL procedures extract individual data points from records through XPath queries against the XML data ("XML Path Language (Xpath) 3.1" 2017). Similarly to the design of the database schema, these were developed through a combination of inspecting the XML schema as well as some extracted records.
Once again, visual inspection during this process resulted in additional adjustments being made to the database design, including adding new fields to existing tables and creating new ones where necessary, e.g. the `gtrTopics` table.

Some difficulties encountered during the development of these procedures included:

- Very high memory usage during importing.
  The exported XML files for project records total to over 700 megabytes at the time they were last exported. While creating the import procedures, I found an that PostgreSQL would use large amount of the system's memory, dramatically slowing down both the system and the import process.
  I noticed that memory usage increased dramatically each time I implemented an additional sub-record to be imported. For example, the list of organisations associated with a project is specified in the XML records for projects. Initially, these were extracted using an XPath query, then iterated over to insert each sub-record, in this case into `gtrProjectOrgs`. This iteration is performed using the FOR-IN syntax[19], which has the following structure:

  ```
  FOR identifier IN
    -- QUERY
  LOOP
    -- STATEMENTS
  END LOOP
  ```

  I found that each unique identifier used within a for loop resulted in greater memory usage.
  My solution was to simply use a single variable for these sub-loops, when necessary, sacrificing some clarity in the meaning of the loop variables for more efficient use of memory. Most of the queries were instead adjusted to use a SELECT query within the INSERT statement[20], eliminating the need for temporary variables.
  The project importing procedure remains the most memory-intensive, but the PostgreSQL process peaks at just under 6 gigabytes during its execution, which is feasible for most computers to handle.

[18] https://gtr.ukri.org/gtr/api/person

[19] https://www.postgresql.org/docs/12/plpgsql-control-structures.html#PLPGSQL-RECORDS-ITERATING

[20] https://www.postgresql.org/docs/current/sql-insert.html

Memory usage could be further reduced by using temporary tables for sub-records, and additional research into minimising memory usage in PostgreSQL procedures.

- XML entities are not replaced automatically.
  An XPath query in PostgreSQL that targets the text of an element does not automatically decode any entities that reside within the text.
  XML data will normally encode specific characters as *entities* to avoid conflicting with the syntax. Among these are ampersands, `&`: in XML data these are encoded as `&`.
  Once this was noticed, manual text replacement was added for fields where it was noticed.

- Money values being imported incorrectly.
  Initially monetary values, such as the funding organisations provided to projects, were being imported by casting the text of the fields directly to the PostgreSQL type `money`, which can store currency values.
  For example, an exported project record will contain a list of participating organisations and the value of their contribution as decimal number:

```xml
<ns2:project>
  <!-- ... -->
  <ns2:participantValues>
      <ns2:participant>
      <ns2:organisationId>
          AAC0E6BF-AC85-4577-AA5A-2E8725FFAF7B
      </ns2:organisationId>
      <ns2:organisationName>Wsp UK Limited</ns2:organisationName>
      <ns2:role>LEAD_PARTICIPANT</ns2:role>
      <ns2:projectCost>100000.0</ns2:projectCost>
      <ns2:grantOffer>100000.0</ns2:grantOffer>
      </ns2:participant>
  </ns2:participantValues>
  <!-- ... -->
</ns2:project>
```

  However, once these values become sufficiently large, the exported records will store these decimal values in scientific form, which PostgreSQL cannot cast to a monetary type directly. Attempting to do so results in errors such as:
  `ERROR:  invalid input syntax for type money: "3.4442927E7"`
  The solution was to cast to an intermediary type, `numeric`, before then casting to `money`:
  `(xpath('//pro:projectCost/text()', x, nss))[1]::text::numeric::money AS cost`
  instead of simply
  `(xpath('//pro:projectCost/text()', x, nss))[1]::text::money AS cost`.

- Conflicts occurring upon repeating the import process
  Testing the import procedures involved running them repeatedly to test new adjustments. An expected result of the primary key constraints implemented in the database design is that no two records may share the same primary key. As a result, these repeated executions would result in unique constraint violations.
  In order to work around these, I used PostgreSQL's `ON CONFLICT` clause[21] to handle these conflicts. In cases where new fields were added, the `DO UPDATE SET field = value` action was used to set the value of this new field for already existing records. These

[21] https://www.postgresql.org/docs/12/sql-insert.html#SQL-ON-CONFLICT

were eventually replaced with the `DO NOTHING` action to simply skip over already existing records.

6.3 Data cleaning

Data cleaning is required to ensure that the data used during analysis is of high quality, i.e. free of errors and sufficient for the planned analysis. The use of low quality data negatively impacts analysis, resulting in inaccuracies and less useful outcomes (Rahm and Do (2000)).

The Gateway to Research data is entered by researchers and individuals involved in the grant funding process. As a result, human error can result in erroneous data and duplicate records. These problems are classified as 'single-source' problems, as they occur even a single database is being considered. The introduction of multiple databases then introduce 'multi-source problems', such as different encodings for the roles of organisations, or different monetary systems being used for currency values.

In order to make the development and further steps of this project simpler, the processes undertaken to clean data avoided removing or modifying any data imported from the Gateway to Research database. Instead, new database tables and/or fields were created to store information during the cleaning process, as well as storing the outcomes of cleaning: merged records.

6.3.1 Eliminating invalid records

The first step taken during data cleaning was to identify records that contain no useful information and/or do not refer to a real entity. For example, an organisation record with the name 'Unknown'[22] doesn't represent a real organisation, limiting how its relationship with other entities can be explored. Such issues exist at the *record* scope of problems (as categorised by Rahm), due to this problem applying to the record as a whole.

Initial exploration of the data was undertaken, revealing a number of other organisation and person records with the name 'Unknown' (with varying capitalisation). By querying the database for organisation records with the name specified as 'Unknown', I found that a dozen other records used this name, each providing no information in the fields other than the name.
By querying the database for most popular names used by organisation records (`SELECT name, COUNT(name) FROM gtrOrgs GROUP BY name ORDER BY count desc;`), I found a similar pattern with the name 'Unlisted': over 100 records used the name (in various forms of capitalisation), each also lacking any other information.

In order to indicate these records were not to be used during analysis, I created a table `junkGtrOrgs`, which contains the UUIDs of such organisation records. All records with the name 'Unknown' or 'Unlisted' were inserted into this table.

From the name frequency analysis, I also noticed a significant number of organisation records shared what appeared to be names of organisation departments - 'Research', 'Economics', 'Psychology', etc.. Similarly, these records contained other information but the name.
An example of such a record is the Economics Department of Queen Mary University of London (QMUL)[23]. On the web portal, the associated projects for this record all indicate that these projects are/were led by QMUL - the same is indicated by my records in `gtrProjectOrgs`.
I inferred that organisation entries are automatically created for organisation departments when the 'lead organisation department' is specified for a project, for example in an associated project:

```
<ns2:project ... ns1:id="86616057-4857-4DAC-A1AE-3D354EB9C34A">
    <ns1:links>
        <ns1:link
            ns1:href=".../organisations/D5337A10-AC8A-402A-8164-C5F9CC6B0140"
```

[22]https://gtr.ukri.org/organisation/39BBE949-0333-428F-864F-C3B196D3D92D
[23]https://gtr.ukri.org/organisation/3A5E126D-C175-4730-9B7B-E6D8CF447F83

```xml
            ns1:rel="LEAD_ORG"
        />
        <!-- ... -->
    </ns1:links>
    <ns2:title>The marriage premium revisited: the case of baseball</ns2:title>
    <ns2:leadOrganisationDepartment>Economics</ns2:leadOrganisationDepartment>
    <!-- ... -->
</ns2:project>
```

The only linked organisation in this project record is QMUL. However, if we inspect organisation records for the Economics department of QMUL, we see this project is linked.
During the database importing process explained in [Importing data], only the links specified in project records are imported. Hence, organisation records which are actually departments will have no entries for projects that they were associated with.

These records do not refer to a real organisation entity, and are isolated from other records due to their links not being imported, hence relationships between these entities cannot be analysed. For these reasons, I decided to add organisation records with no linked projects to the invalid records collection.

As a result, 5213 of 47822 (10.9%) organisation records were marked as invalid records.

6.3.2 Merging organisation roles enumerations

Through visual inspection, the enumerations that specify an organisation's role in a project (in the `gtrProjectOrgs` table) appeared to some duplicates. This is likely a result of the merging of several databases without considering the meanings of these enumerations.
In order to solve this, I created a new database type, `gtrOrgRole`, and added a utility function that mapped the enumerations within the GtR database to this new type:

GtR enumeration	`gtrOrgRole` mapping
`LEAD_PARTICIPANT`	`Lead`
`PARTICIPANT`	`Participant`
`LEAD_ORG`	`Lead`
`COLLAB_ORG`	`Collaborating`
`FELLOW_ORG`	`Fellow`
`PP_ORG`	`Project Partner`
`FUNDER`	`Funder`
`COFUND_ORG`	`Co-Funder`
`PARTICIPANT_ORG`	`Participant`
`STUDENT_PP_ORG`	`Student Project Partner`

These decisions were made by investigating the GtR API documentation, manually gauging the semantics of each enumeration, and determining which refer to the same role.
For example, the first two enumerations, `LEAD_PARTICIPANT` and `PARTICIPANT` are specified to only be used by the subsidiary Innovate UK ((*GtR Data Dictionary*, n.d.)). Both `LEAD_PARTICIPANT` and `LEAD_ORG` both refer to "[an] organisation receiving project funding which is accountable for ensuring that the planned outcomes for the project are achieved…". Hence, these enumerations can be merged into a single role. The `PARTICIPANT` enumeration is undocumented, but since it is named almost identically to `PARTICIPANT_ORG`, I decided that this pair were semantically identical and thus mapped them to the same value.

Mapping is performed by `ParseOrgRole` in importGtrProjects.sql.

6.4 Merging duplicate records

As explored by Winkler (2006) and Elmagarmid, Ipeirotis, and Verykios (2006) there are many existing methods for identifying duplicate records, most of which involve comparing the similarity of text within records.

In order to track the de-duplication process, additional database tables were created for each entity being de-duplicated.
One set of tables stores the similarity of records: e.g., the `similarGtrOrgs` table tracks pairs of similar organisations within the Gateway to Research database, including metrics measuring the level of similarity, and the result of any manual checking.
Another set of tables stores primary keys of pairs of records that have been determined to be duplicates of each other.
Finally, when appropriate, another table stores records with the merged information of all duplicates. This is the purpose of the `orgs` table, which has all the fields that the main `gtrOrgs` table does, associated with the primary organisation UUID. New records will merge each field from pairs of its duplicates using the `COALESCE` function, which will take the first non-null value. Thus, records in the `orgs` table will contain as much data as the Gateway to Research database contains for each individual entity.

Data normalisation involves splitting data into its minimal atomic components, and enforcing well-structured relations between entities stored in the database. Doing so provides confidence that the stored data is free of anomalies and always correct. One part of this process involves eliminating fields whose values can be calculated from the values of other fields - if the record is updated these values can become outdated, resulting in anomalous information being stored (Lee (1995)).
Storing similarity metrics for pairs of records breaches this principle. However, I believe this decision is justified for several reasons:

1. Original data needs to be maintained so that the changes made during data cleaning can be identified and statistics can be reported within this report.

2. It takes a significant amount of time to calculate these metrics:
   As of the date of data aggregation, the Gateway to Research database contained just under 48,000 records of organisations, and just over 82,000 records of people. Performing unique pairwise comparisons for each type of record is thus rather computationally intensive: in total, the triangle number of $n - 1$, (where n is the number of records) comparisons must be made. Given 48,000 organisations, that results in $\frac{n(n-1)}{2} = 1151976000$ comparisons.
   Re-calculating each time these values are required would be a great inconvenience during the development of the project, as these values are required regularly during de-duplication and analysis.
   The similarity tables instead act as a cache that can be referred back to when needed to speed up processes throughout the project.

3. The database is not regularly being updated:
   Data from Gateway to Research was exported only twice during the project, and the project does not aim to keep up-to-date with the current state of the Gateway to Research database, so it is appropriate to calculate metrics once so that they can be reused.

Database tables were created to store pairs of similar records and potentially useful values for each pair. `similarGtrOrgs` stores similar pairs from `gtrOrgs`, and `similarGtrPeople` stores

similar pairs from `gtrPeople`.
For each pair in `similarGtrOrgs`, the calculated trigram similarity of the names and addresses are stored, as well as a boolean field for the result of any manual checking: `NULL` indicates no manual check has been performed, `TRUE` that the pair were confirmed to be duplicates, and `FALSE` that the pair were confirmed to be unique entities.

6.4.1 Entity name analysis

Manually exploring records revealed small variations with how organisation names were entered into the database, as expected of data entered by untrained users.
For example, private companies incorporated as 'limited' often contain a variation of 'Limited', 'Ltd', or 'Ltd.' in their names. The names of duplicate records will vary in how this is specified in the name. The same is true of 'Corporation' ('Corp.'), 'Company' ('Co.'), ampersands ('and'), and 'University' ('Uni.').
Names sometimes included 'The', other times it was omitted - e.g. 'The Institute of ...' would sometimes be entered as 'Institute of ...'.

Another notable feature was the significant number organisation records with postcodes or any form of address omitted: 42.5% provided no postcode, and 32.2% provided neither postcode nor any other part of an address.
As addresses enable two organisations to be clearly distinguished from each other, this means it is not possible to rely on the address information to de-duplicate organisations.

As for records on people, another common pattern is users entering nicknames in the place of first names, such as 'Tom' instead of 'Thomas'. This complicates the de-duplication process, as if one record uses a nickname and the other the full name, these will have a lower similarity metric despite both referring to the same name.

To compensate for this, the similarity comparison metrics were adjusted to ensure that variations of the same words are replaced with a single variation. This resulted in comparisons of records that used these variations having a higher trigram similarity than they did before these replacements.

6.4.2 Q-Gram matching

One approach for detecting duplicates is the q-gram matching method, which compares the number of shared substrings of length q within two bodies of text.
PostgreSQL has built-in support for performing this comparison through the pg_trgm module[24]. This module performs q-gram comparison with q as three, hence comparing 3-character sequences. These are known as *trigrams*.

As an example, the word 'trigram' is broken down into the following trigrams:
`tri`, `rig`, `igr`, `gra`, and `ram`.
In PostgreSQL's implementation, a "string is considered to have two spaces prefixed and one space suffixed", hence these additional trigrams are generated (the symbol '␣' indicates a space): `␣␣t`, `␣tr`, and `am␣`.
These additional trigrams add additional weight to the beginnings and ends of individual words.

I chose to use this method due to existing support in the PostgreSQL database management system, as well as its particular affinity to catching typographical errors in data entered by users.

Compared to some other similarity comparison methods, q-grams are not as sensitive to the

[24] https://www.postgresql.org/docs/12/pgtrgm.html

order of words within a strings. This is particularly important for data entry by users, as names or addresses of organisations may be entered in slightly different variations.
For example the Levenshtein distance (or edit distance) "between two strings [. . .] is the minimum number of edit operations of single characters needed to transform" one string to another. The trigram similarity of the strings "The University of Nottingham" and "Nottingham University" is 75.9% (3sf), whereas the Levenshtein distance is 23. A significant number of characters must be changed to transform one string to the other, despite both strings sharing two entire words.

Similarity was calculated and recorded through SQL procedures, similarGtrOrgs.sql and similarGtrPeople.sql.
The calculated values for trigram similarity were stored for names of organisations (`similarGtrOrgs.simTrigramName`) and people (`similarGtrPeople.simTrigramName`).

An initial attempt at running these scripts involved calculating the similarity of every pair of records, but this failed after a significant amount of time, once it had completely filled the remaining 50GB or so available on the storage device the database was being stored.
In order to avoid this, I filtered pairs by requiring that they share at least 50% of trigrams in their names. This allows the amount of data generated from this procedure to be significantly reduced, down to 243363 from 1151976000 rows for organisations.

An initial attempt to compute the pairwise similarity of records proved to be computational infeasible. It required over 30 minutes and a storage space of more than 50GB before the procedure failed due to running out of disk space. In order to address this issue, record pairs were filtered to include those that share at least 50% of trigrams in their names. This reduced the amount of data generated down to 243363 from the original 1151976000 rows for organisations.

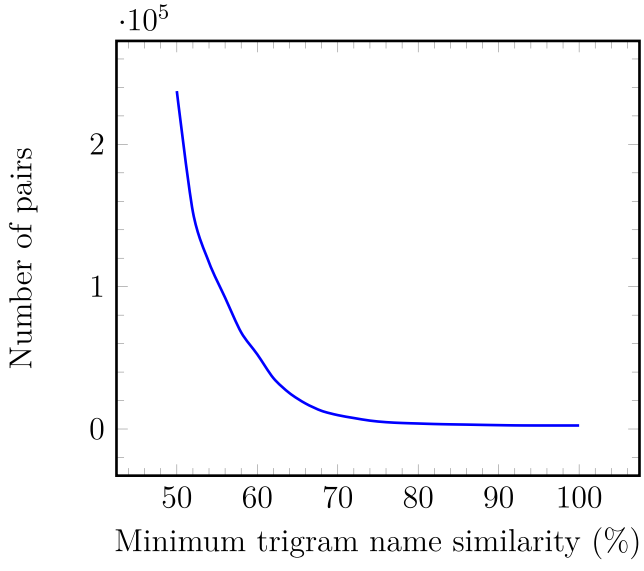

Figure 1: Similarity distribution of similarly-named organisation records. Generated with `OrgSimDist` from stats.sql

6.4.2.1 Organisations Once all these similarity records were created for organisations, I applied an hypothetical heuristic: assume two records are duplicates if their names share at least 90% of trigrams. This results in 2710 or 5.6% organisations being duplicates.
I then chose 100 random records and manually verified whether the pair of organisations were indeed duplicates, which was determined if one of the following conditions held:

1. Both organisation records listed addresses, and the addresses matched
2. Both organisation records listed addresses, and there existed public records of a single organisation being registered at both addresses throughout its history (Using the Companies House[25] service)
3. Both organisations records worked on an identical project

If one of the records did not specify any form of address, then the pair was skipped.

A script was created to present the user with each pair, who can then specify the result of manual checking and update the database records accordingly.
My reasons for choosing Python for this utility are similar to those explained in [Aggregating Data] - Python additionally has support for performing PostgreSQL queries through the psycopg2 module[26].
This necessitated adding an additional field to `similarGtrOrgs` named `manualResult`. The field is a boolean flag indicating the result of manual checking, where a true value indicates the records refer to the same entity. Should this field be `NULL`, this indicates no manual checking has been performed.

The results are shown in the following plot:

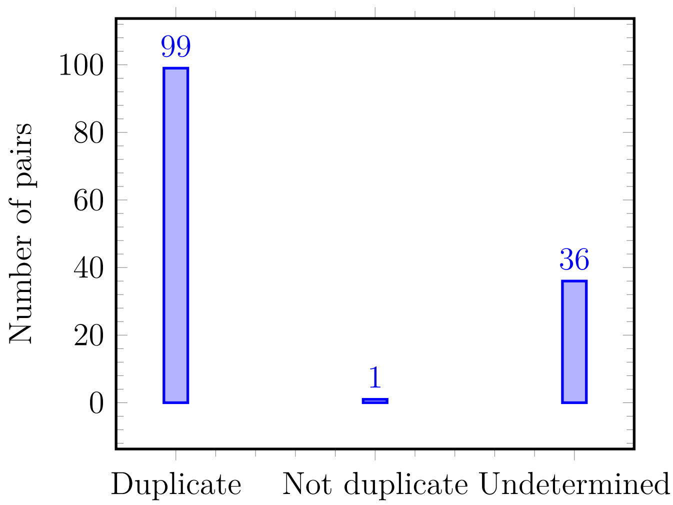

Figure 2: Results of manual verification of duplicate organisations whose names have a trigram similarity of 90% or above

A significant portion number of pairs contained an organisation that had no address specified and lacked sufficient information to determine whether the pair were indeed duplicates. This was expected, as a significant portion of organisation records lack address information.

However, of those pairs that both contained addresses, 99 were confirmed to refer to the same entity, normally through sharing an identical address. A smaller number of those were confirmed by checking the historical addresses for a real organisation, and finding that it has previously been registered in both. Less than 10 were determined to be duplicates through sharing similar partners on projects, or similar topics for projects.

The single pair that was confirmed to not be a duplicate was the University of Los Lagos (in Chile) and the University of Lagos (in Nigeria), with a trigram name similarity of 91%.

[25] https://beta.companieshouse.gov.uk
[26] https://psycopg.org

This accuracy is acceptable enough for the purposes of this project, and so I decided to apply the heuristic of 90% trigram similarity during the de-duplication process, merging records where this property applies.

Additional confidence could be provided to the de-duplication of organisations by automatically querying an API for records of these organisations, such as the Companies House API[27]. The addresses of records in the Gateway to Research database could be matched against historical names and addresses automatically, similarly to the manual process I performed.
Data from the Companies House database could also be exported and linked with that of the Gateway to Research database.

6.4.2.2 People Trigram similarity was calculated for pairs of records on people, taking the average trigram similarity of both the first name and surname. Where both of the pair specified the 'other names' field, the similarity was calculated as the average trigram similarity of the first name, surname, and other names.

First names were filtered to replace nicknames with possible full names before comparison, to minimise the negative impact of users providing nicknames in the place of full names.

6.4.3 Manual inspection

6.4.3.1 Organisations Through manually inspecting organisation records with high name similarity, I identified that the specified postcodes can be used to reliably identify whether similarly named records are indeed duplicates.
Hence, I decided to merge organisation records if the following conditions hold:

1. Neither of the pair is listed as an invalid record
2. The pair share at least 90% of trigrams in their names, or
3. The pair share at least 50% of trigrams in the names, both have specified postcodes, and the postcodes match

As explored in Q-Gram matching, using a threshold of 90% in name similarity is a rather effective heuristic, and hence I have applied it when deciding to merge organisation records.
Matching postcodes provides enough additional confidence that a pair of organisation records are duplicates, hence I apply a much lower trigram similarity threshold for merging them.
The accuracy of the de-duplication is not paramount to the purposes of this project, so 100% accuracy is not necessary. The number of records and proportion of those that lack much information also makes it infeasible to complete this process manually.

[27] https://developer.companieshouse.gov.uk/api/docs

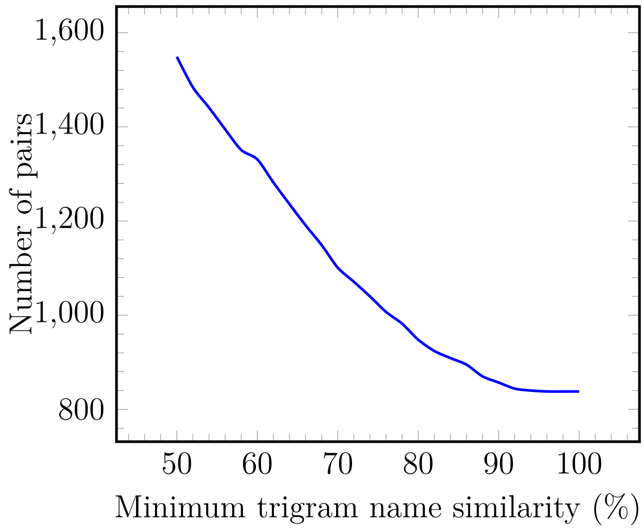

Figure 3: Similarity distribution of similarly-named organisation records, who also share the same postcode. Generated with `OrgSimDistPostcode` from stats.sql

The mergeGtrOrgs.sql procedure performs the record merging, storing the resulting records in the `orgs` table. The `COALESCE` function on each duplicate pair's fields so that the resulting record will take all information available on a single entity.

As a result, 3091 of 47822 (6.5%) organisation records were marked as duplicates.

6.4.3.2 People I identified that 617 of 14456 (4.3%) of similar person records shared the employer, which could be used as an additional heuristic in determining whether the pair are duplicates. By linking the employers against the duplicate organisation list in `duplicateGtrOrgs` as generated in the previous section, this number rises to 625.
Using this employer information in conjunction with name similarity results in much more confidence when determining if two records indeed refer to the same individual.

From inspecting the most similar records, I noted that despite having similar names, most had different employers. While it's possible for the same individual to be employed by different organisations over time, the records lack sufficient personally-identifiable information to reliably link them to a single real individual.

As individuals' names are significantly less unique than that of organisations, relying on name similarity alone in de-duplication appears infeasible. Hence, I decided that person records should only be merged if the following conditions hold:

1. Both records list the same employer
2. The name of both records share at least 90% of trigrams

The mergeGtrPeople.sql procedure performs the record merging, storing the resulting records in the `people` table. The `COALESCE` function on each duplicate pair's fields so that the resulting record will take all information available on a single entity.

As a result, 499 of 82015 (0.61%) of records on people were marked as duplicates.

6.4.4 Summary

Name trigram similarity is an effective heuristic for detecting duplicate organisations, as these tend to use more unique names to distinguish themselves. The same is not true of people: due to the wide re-use of names within populations, names alone cannot be used to uniquely identify a real individual. In these cases, additional contextual information must be used to ascertain duplication, such as an individual's employer.

Using name comparison is more effective when common variations of the same words or names are substituted for a single variation before comparison. However, this approach is limited by the requirement of domain knowledge being applied or a significant amount visual inspection being done.

Combining name similarity with the comparison of other information available - such as the addresses of organisations - results in more records being linked. 4871 of 47822 (10.2%) organisation records are detected as duplicates when the heuristic is that they share 90% or more of trigrams in their name. By checking whether postcodes match, this number rises to 5935 (12.4%).
Manual testing of this method found 99 of 100 linked organisation records were indeed duplicates (though an additional 36 could not be determined). This suggests combining such contextual information is more effective than simply using name similarity.

Limitations to these heuristics include:

- Incomplete records: Many omitted address information, meaning solely name comparison could be performed.
- User error: Postcodes being specified in the incorrect field or differing by a couple of characters.
- Limited amount of information: Analysing sets of organisations collaborated with is less reliable for records with few projects associated with them.

Public government records could be queried to search previous names and addresses of organisations. These combinations of names and addresses could be compared between pairs of records to provide additional information to the de-duplication process.
The dates that these records were created is also provided and could be used to identify what an organisation was named and what address they were using at a certain point in time.
This information could provide more confidence in the decisions made during the de-duplication process.

Another possible heuristic is to compare the set of organisations that a pair of organisations have previously collaborated with, or to compare the research topics of projects they have been involved in.

7 Data analysis and visualisations

7.1 Grouping organisations by type

A variety of organisations exist in the database, from universities, local councils, medical institutions, to private companies.
For the purposes of analysis, they have been grouped into four categories:

- **Academic**: For academic institutions like universities, colleges, and other schools
- **Medical**: Medical institutions like hospitals, medical research institutions, and other healthcare providers
- **Private**: Privately owned corporations
- **Public**: Government bodies and publicly funded institutions

Organizations are placed in each category based on pattern matching of the organization names. More precisely, organisation records (and records that are listed as duplicates) are searched for keywords that indicate their type:

Type	Exemplary keywords
Academic	University, College, Academy, Academic
Medical	Hospital, NHS, Medical, Health
Private	Limited/LTD, Corporation, Company, Incorporated
Public	Council, Government, Governorate

The organisation type is stored as an additional field within the organisations database table (`orgs`), which contains records which have been de-duplicated and cleansed of invalid records. Grouping is performed in the procedure [classifyOrgs.sql].
After applying this procedure, 21694 of 39578 organisation records (54.8%) were given a type.

The procedure was manually enhanced by refining the keyword list and controlling the scope of pattern matching. For example, abbreviations, such as LTD, LLP, and PLC are also used for identifying private limited companies alongside the 'Limited' keyword. The procedure also considers the optional period at the end of these abbreviations, such as "Uni." for "University". Another issue is that of keyword overlap: e.g. is Albany Medical College a medical institution or an academic one? Through manual research we can discover that it is indeed an academic one, but as the procedure sets the group of medical organisations after academic ones, these are categorised as medical.
An improvement to this procedure could be to consider combinations of words that appear in the outliers, ordered to prioritise these patterns over single keywords.

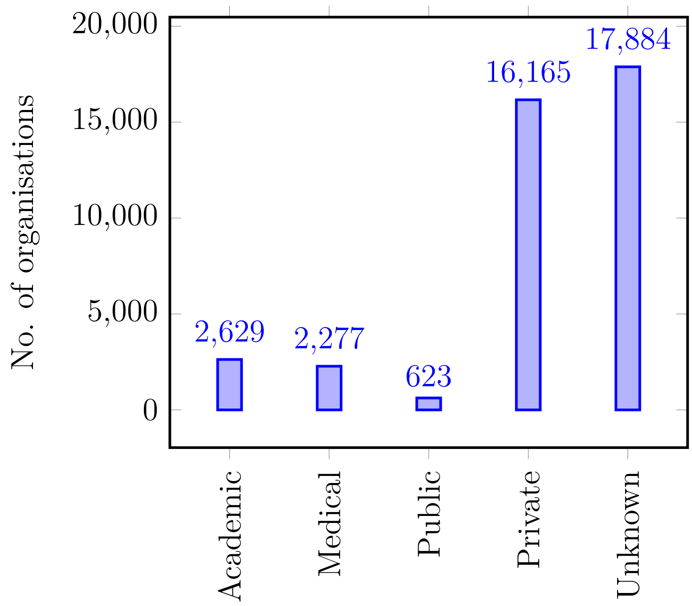

Figure 4: Distribution of organisations within the UKRI dataset, grouped by type. The type is determined using keywords contained in their names, such as 'Limited', or 'University'. 54.8% of records were grouped, with the remainder given the 'Unknown' type.

Due to the limitation of time, I have not endeavour to collect more information about individual

organizations. For that purpose, a public database such as the UK's Companies House[28] service could be used to determine both the legal classification and the nature of business of an organisation. The nature of business is reported by the company themselves, and so may not be entirely reliable: e.g., the Nottingham University Hospitals Trust Charity[29] lists their nature of business as "hospital activities" despite also being an educational institution.
The categories used by Companies House are provided on their website: ("Nature of Business: Standard Industrial Classification (Sic) Codes," n.d.).

7.2 East Midlands network analysis and visualisation with Gephi

In order to explore the structure of research ecosystems, network analysis was employed. A subset of the network was chosen so that patterns could be explored at a smaller scale.
Organisations were selected if they had 'East Midlands' specified as their region and had an active projected within the years 2002 and 2008.

Gephi[30] is a Java-based graph analysis tool, and was used to plot the network of organisations based in the East Midlands region or Nottingham.
This software was chosen as it is free and open source, and so it costs nothing to use. Documentation also suggested it was easy to use, which would enable me to quickly get to grips with the application in order to use it in this project. Through research I also found that it includes several features that enable network analysis and visualisation, the possibilities of which I was keen to apply to explore.

Limiting the dataset used resulted in analysis being much faster to perform, as only 238 of 39578 organisations match these criteria. Fewer records mean less computational power is needed to manipulate the dataset in Gephi.
While Gephi is able to load the entire dataset if the Java virtual machine is allowed to use more system memory (e.g. by setting `_JAVA_OPTIONS="-Xms1024m -Xmx10000m"`), adjusting the layout, filtering, or calculating statistics still took a significant amount of time.
I also found that limiting the dataset significantly increased the legibility of the resulting graph, as having fewer nodes and edges means less overlap. With all nodes and edges visible, it was impossible to distinguish individual edges due to the massive number of them - 283197 in total. In the filtered dataset this number is reduced to a much more manageable 940.

While Gephi supports importing data directly from a database through queries, I encountered difficulties when attempting to use this feature with long queries and large amounts of data.
I instead opted to create procedures that exported the needed data to Comma Separated Values (CSV) files, a common plaintext format, and then imported these files manually into Gephi.

Data was exported from the database using [scripts/eastMidlandsGraphGephi.sql], then imported into Gephi through the import spreadsheet[31] wizard. This includes organisation records (used as the nodes) and projects that pairs of those organisations were both involved in (used as the edges). Organisation records were inspected for duplicates by sorting by name, and some were manually merged within Gephi.

7.2.1 Analysis

Gephi allows some graph properties to be analysed, including the degree distribution of nodes and (weighted) clustering coefficient.

[28] https://beta.companieshouse.gov.uk
[29] https://beta.companieshouse.gov.uk/company/09978675
[30] https://gephi.org
[31] https://github.com/gephi/gephi/wiki/Import-CSV-Data

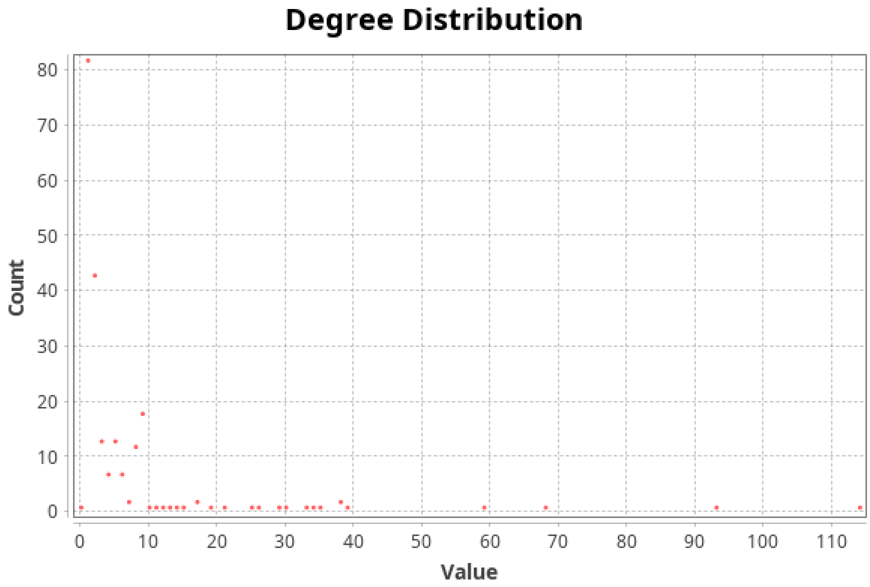

Figure 5: Degree distribution of the East Midlands funding network (2002 to 2008), where nodes are organisations and edges are projects those organisations are collaborating on.

The degree distribution shown in figure 5 is similar to that of the network as a whole (see Amount of research), with the vast majority of nodes (i.e. organisations) having a low number of connections, indicating that they are involved in only a few collaborations.
Frequencies drop dramatically past degree 10, and very few organisations have the highest degrees, and these are particularly sparse within the distribution. This supports the expectation that there are a few organisations that make up the most of research and collaboration, and that most are only involved in a few projects.
As the distribution is similar to that of the database as a whole, this suggests that publicly funded research in the East Midlands region does not stand out significantly from other regions in terms of amount of research and its distribution.

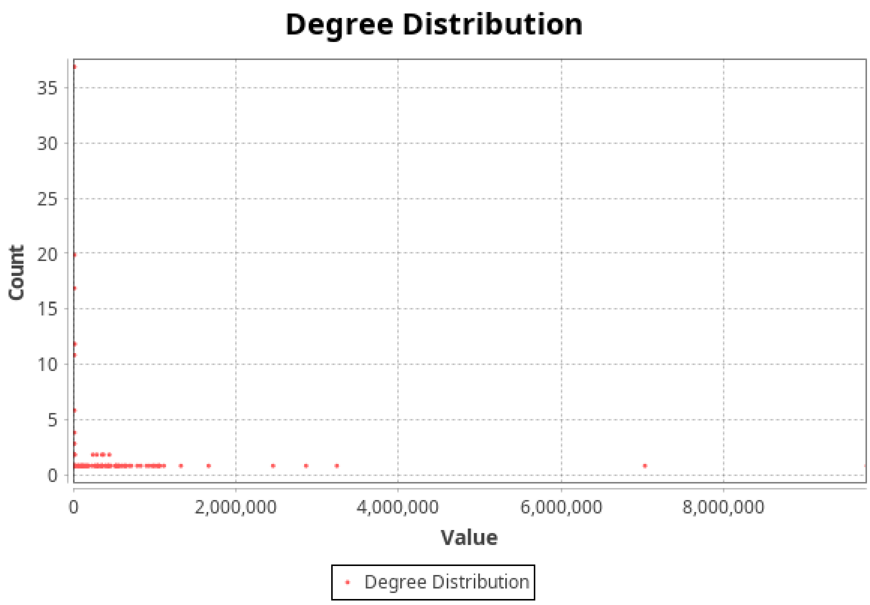

Figure 6: Weighted degree distribution of the East Midlands funding network (2002-2008), where nodes are organisations, edges are projects those organisations are collaborating on, and the weight of edges is the total spent on the project between both organisations.

The weighted degree distribution (where each edge is multiplied by its weight) for the graph is shown in figure 6, where the weight is the cost of the project between the two organisations. A similar pattern is shown to the degree distribution; lower values have significantly greater frequencies, and higher costs becoming rarer as indicated by the sparsity at higher values.

Figure 7: Clustering coefficient distribution of the East Midlands funding network (2002 to 2008), where nodes are organisations and edges are projects those organisations are collaborating on. It shows that organisations are much more likely to only be involved in a few collaborations, and organisations with high amounts of collaboration are few and far between.

The clustering coefficient is a measure of "the degree to which nodes tend to cluster together" (Opsahl and Panzarasa (2009)), originally attempted by Luce and Perry (1949).
When considering a single node, its coefficient indicates how close that node's neighbours are to being a fully-connected graph: a lower value indicates less connection, higher indicates more. Gephi calculates this for triplets (referred to as 'triangles') of nodes in the graph.
Gephi was used to calculate the distribution of clustering coefficients for all nodes in the network, with the results shown in figure 7. We can see that a large number of triangles have a clustering coefficient of 0, and slightly fewer have a coefficient of 1. This indicates that there are a similar number of triplets of organisations that have collaborated with each other, as there are those that have never collaborated with each other. The remaining node triangles are distributed between these extremes 0 and 1, with coefficients higher than 0.5 being slightly more frequent. These results show that most of the network is highly connected, suggesting there are many organisations in the East Midlands that engage in research with many other organisations in this region. There is also a significant portion that does not: these organisations are likely involved in few projects with few collaborators.

7.2.2 Visualisation

Nodes were filtered to those with at least one connection, then Yifan Yu's proportional graph layout algorithm (Hu (n.d.)) was applied. This layout algorithm clusters related nodes while minimising the amount of edge crossing. Clustering is a result of edges causing connected vertices to attract one another while all vertices repel each other.
Edges were scaled according to the amount spent on projects between those organisations.

Nodes were scaled by their (unfiltered) total number of connections and coloured according to their type:

- Purple: Academic
- Blue: Private
- Green: Medical
- Gray: Unknown

Nodes with no connections were removed, minor repositioning was performed to make the graph visualisation more compact, and some nodes were adjusted to fix overlapping labels.

The resulting visualisation can be found in figure 18.

The network shows academic organisations being involved in the majority of projects in this period & area, having many connections to other organisations through projects. Among these are the University of Nottingham, Loughborough University, and Loughborough College.
Medical organisations are also heavily involved in research, including the University Hospitals of Nottingham and Leicester. These have formed large hubs of connected organisations, showing the amount of research and variety of organisations they are involved in.
A few private and other classes of organisations stand out including PERA Innovation[32], a research association for the manufacturing sector, and Experian[33], a credit reporting company. Both have a significant number of connections and spend a lot on research.

Almost all organisations are connected to a single contiguous graph, but there are a few outliers that reside in their own disjoint networks, shown in the bottom left of the graph. These include Sun Chemical[34] who produce "printing inks, coatings and supplies".
This suggests these organisations are in niche industries or involved in research that does not benefit most organisations.

7.3 East Midlands network analysis and visualisation with NodeXL

NodeXL[35] is an add-on for Microsoft Office Excel by the Social Media Research Foundation, originally developed to explore and analyse social media networks. It can be used to explore any network, however. NodeXL was recommended by my supervisor, who had previously used it in her research, and kindly provided a training session on using it.
As the University provides complimentary access to the Microsoft Office suite, and a free version is provided, I decided to try applying it to this project.

Similarly to the Gephi network, I filtered organisations to the East Midlands region, for the same reasons as explained in [East Midlands network with Gephi].
Microsoft Excel performed much worse compared to Gephi when I attempted to handle the entire dataset. The application regularly became unresponsive when manipulating the dataset or when NodeXL was generating graphs.
I decided to explore a different year range for projects than was explored in the Gephi visualisation, from 2005 to 2010, so the differences in the networks can be explored.
This resulted in 327 nodes and 806 edges being selected for this network.

The procedure for exporting this data is provided in eastMidlandsGraphNodeXL.sql.

[32] https://www.perainternational.com/about
[33] https://www.experian.com
[34] https://www.sunchemical.com
[35] https://www.smrfoundation.org/nodexl

7.3.1 Analysis

In the free version of NodeXL, the available analyses are greatly limited. While it supports several algorithms Gephi also supports (such as clustering coefficients and eigenvector centrality), these are limited to the paid version. One available analysis available is degree distribution, which is shown in figure 8.

Just as in the 2005 to 2008 network, the vast majority of organisations are involved in only a few collaborations, with higher amounts of collaboration being increasingly rare. Degrees above 20 seem to have become rarer in the 2005 to 2010 time range, but the top degree is 172 compared to 2002 to 2008's 114. This suggests a lower amount of collaboration from most organisations in this later time period, but the top organisations are involved in more.

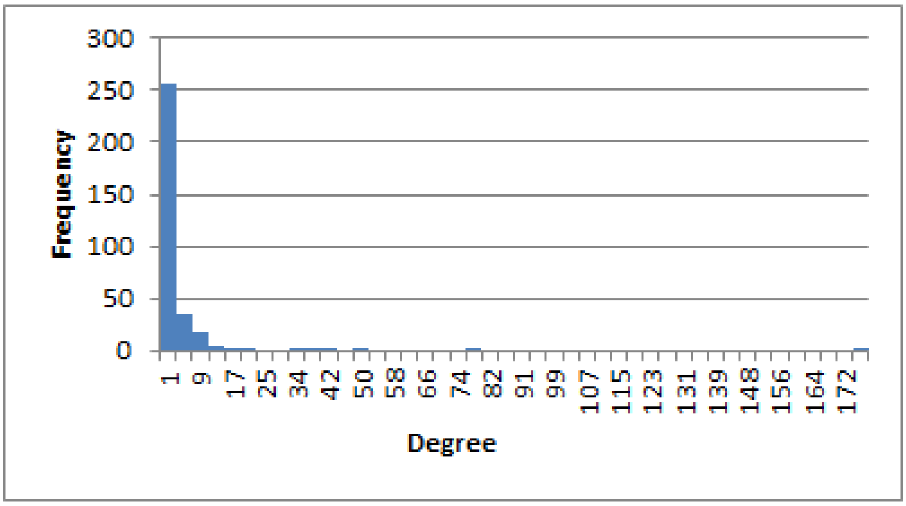

Figure 8: Degree distribution of the East Midlands funding network (2005-2010), where nodes are organisations and edges are projects those organisations are collaborating on. It shows that organisations are much more likely to only be involved in a few collaborations, and organisations with high amounts of collaboration are few and far between.

7.3.2 Visualisation

Nodes were scaled by the number of project connections they had within that time period, and edges scaled by how much was spent by either organisation on that project.

I applied the Fruchterman-Reingold layout algorithm (Fruchterman and Reingold (1991)), which considers edges in positioning. By adjusting the configuration of the repulsive force to 25.0 and number of iterations to 25, this resulted in nodes with greater degrees being positioned towards the centre of the graph, and other nodes being placed towards the periphery.

The layout algorithm was also configured to position small networks in the bottom-left of the graph.

Labels were added for nodes with the highest degrees (greater than 13), and some other ones close to the centre of the graph. Minor repositioning was performed to reduce overlap of these labels.

Nodes were coloured according to their type:

- Dark blue: Academic
- Red: Private
- Light green: Medical
- Dark green: Unknown
- Light blue: Public

The resulting visualisation can be found in figure 19.

Academic and Medical institutions are shown towards the centre of the graph and with high degrees, indicating they are heavily involved in collaborative research, as one might expect. This pattern is also visible in the 2002 to 2008 network. This graph also shows that most academic organisations have higher degrees than most, hence are involved in more collaboration.
With the exception of PERA Innovation, all the organisations mentioned in the analysis of the 2002 to 2008 network also appear as nodes with high degree in this new graph. This suggests the remaining organisations have maintained the large amount of research they perform, whereas PERA Innovation is less involved in publicly funded research within the East Midlands region.

Some particularly notable edges appear near the centre of the graph and travel downwards to the bottom.
These both represent a project that a significant amount of money was invested into to develop hydrogen fuel cell systems[36], led by Intelligent Energy Limited[37] and partnered with Frost Electronics Limited.

7.4 Important factors in collaboration

7.4.1 Amount of research

We can take the number of projects an organisation has been involved in to be an indicator of the amount of research activity they are involved in. If we explore organisations by the number of projects they are involved in, we can see that the top organisations are involved in significantly more projects than lower organisations. Below the 90th percentile, these organisations are involved in less than 10 distinct projects each, whereas this number becomes exponential past the 90th percentile:

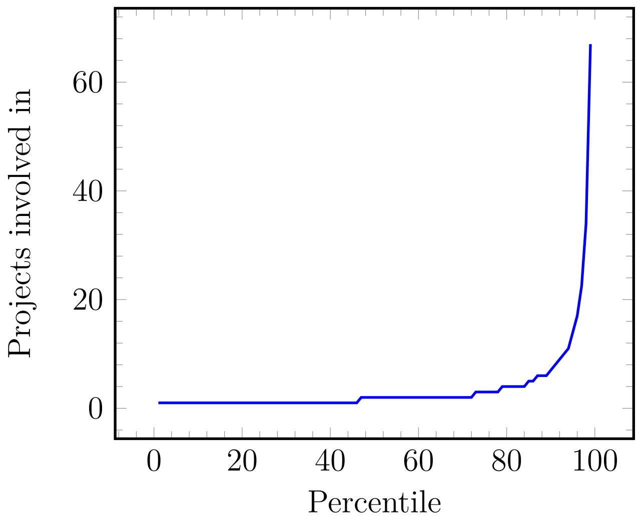

Figure 9: Percentiles for the number of projects an organisation is involved in, across the entire UKRI database. Generated with `OrgProjectPercentiles` in stats.sql

Visualising the distribution of organisations within these percentiles shows a similar pattern: the vast majority are involved with only one or two projects:

[36] https://gtr.ukri.org/projects?ref=113057
[37] https://www.intelligent-energy.com

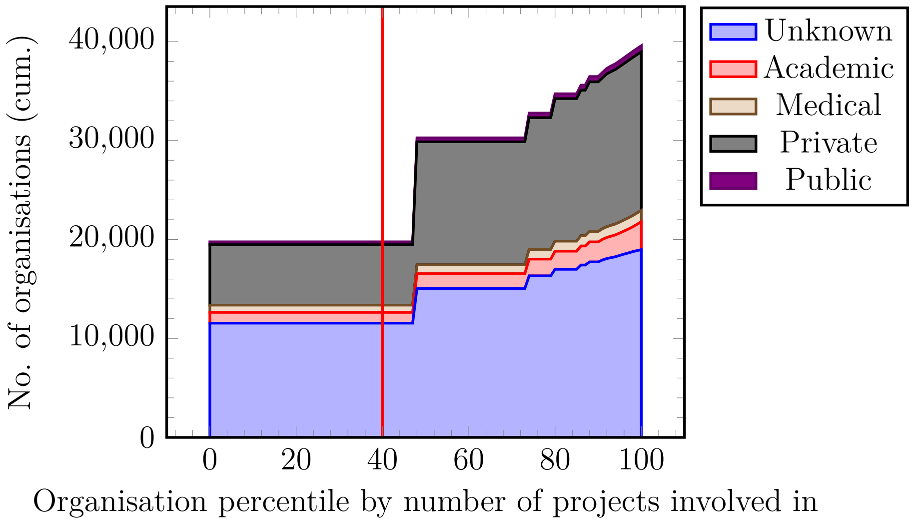

Figure 10: Cumulative distribution of all organisations in the UKRI database by percentile, ranked by the total number of projects they are involved in. The organisation counts are broken down by their type. E.g. the red vertical line indicates just under 20,000 organisations are ranked at the 40th percentile or below. Generated with `OrgProjectPercentileDist` in stats.sql

By exploring the relationship between project involvement percentiles and collaboration, we can see that the majority of organisations have collaborated with organisations of lower percentile ranks:

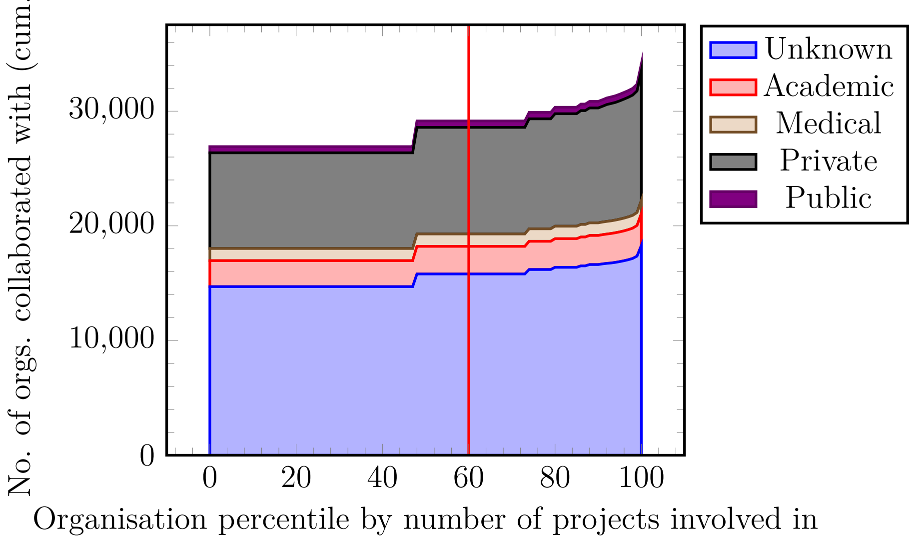

Figure 11: Cumulative distribution of all organisations in the UKRI database, filtered by ones that have collaborated with at least one other organisation at a certain percentile. Organisations are ranked by the total number of projects they are involved in. E.g. the red vertical line indicates that there are just under 30,000 organisations that have collaborated with at least one other organisation at or below the 60th percentile, when ranked by the number of projects this organisation has been involved in. Generated with `OrgProjectPercentileCollab` in stats.sql

While there are many organisations that have collaborated with lower ranked organisations, we still see a sharp increase past the 90th percentile. This indicates a similar proportion of organisations have only collaborated with the top rank researchers, suggesting that the number of projects one organisation has taken part in is a factor in whether another organisation decides to collaborate with them, though not a very important one.

7.4.2 Amount of funding

I expect that the amount of funding an organisation has previously received is a strong indicator of the likelihood that another organisations will collaborate with them.
Hypothetically, the UKRI would provide more funding to organisations who are the leading experts in their area compared to those who are less reputable. Organisations would be more interested in collaborating with experts and those who would bring more funding to projects, as this could result in the outcomes of the project being more valuable.

Of projects that reported any offered funding, the mean funded amount was £471,995, with a large (population) standard deviation of £5,535,964. This indicates that there is a great variety in the amount of funding received by projects.

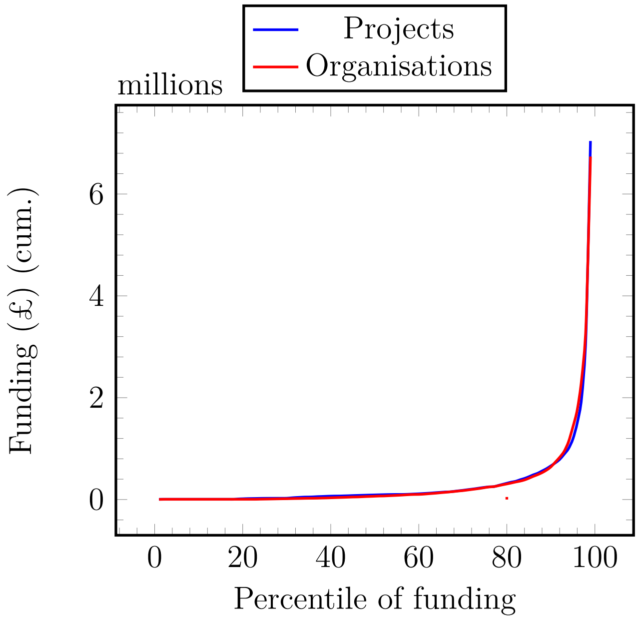

Figure 12: Percentiles of public funding that organisations have received for the projects the are involved in, as well as the total funding received by projects. E.g. the red line indicates that when ranked by the amount of funding received, the top 20least £300,000 in total funding. Generated with `ProjectFundingPercentiles` and `OrgFundingPercentiles` in stats.sql

The percentile distribution is very similar for the total funding received by both projects and organisations. Both curves show that the top 20% of organisations and projects account for the majority of funding received from UKRI.

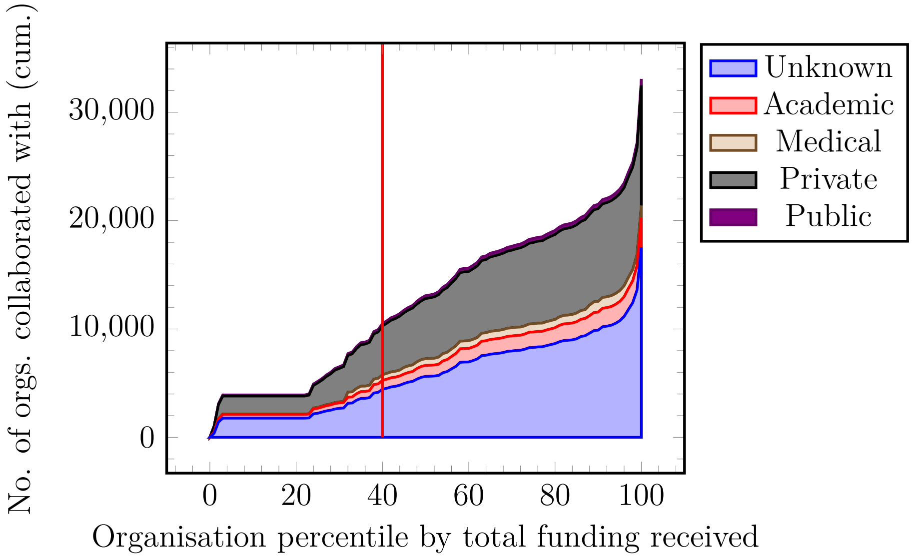

Figure 13: Cumulative distribution of all organisations in the UKRI database, filtered by ones that have collaborated with at least one other organisation at a certain percentile. Organisations are ranked by the total amount of funding they have received. E.g. the red vertical line indicates that there are just over 10,000 organisations that have collaborated with at least one other organisation at or below the 40th percentile, when ranked by the total amount of funding this organisation has received. Generated with `OrgFundingPercentileCollab` in stats.sql

By exploring the relationship between funding percentile and the number of organisations that have been collaborated with, one can see a rather positively linear relationship for the majority of the percentile distribution. This indicates that for organisations within the 20th to 90th percentile, the funding percentile does not have much influence in the likelihood that another organisation will collaborate with them.

However, from the 90th percentile and above, the number of organisations collaborated with dramatically increases in a more exponential manner. This suggests that a significant portion of organisations in the Gateway to Research database have only collaborated with the top 10% most funded organisations.

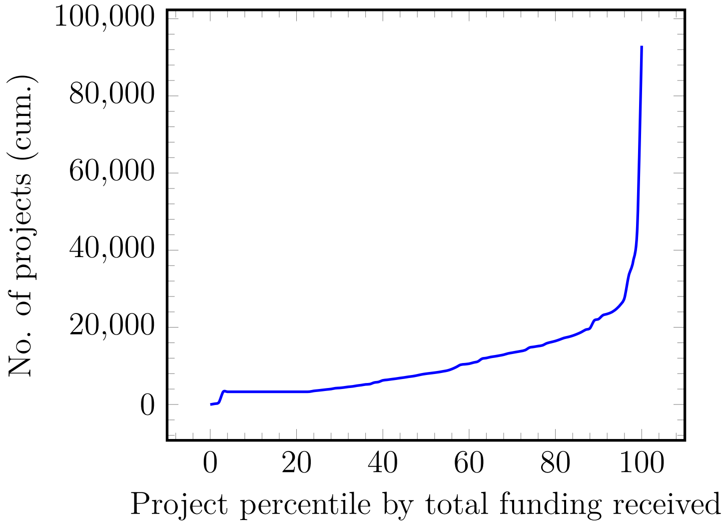

Figure 14: Cumulative distribution of projects by percentile, across the entire UKRI database. Projects are ranked by how much total funding they received. Generated with `OrgFundingPercentileProjects` in stats.sql

If we instead explore the total number of projects that involve at least one organisation at certain funding percentiles, it is trivial to identify a strong positive correlation between the funding percentile and number of projects these organisations are involved in.
A significant majority of projects involve organisations above the 90th percentile, suggesting that these organisations are also the most prolific in their involvement in publicly funded research.

These analyses show that organisations are more likely to have collaborated with only the most funded organisations, and that there are many projects involving only the most funded organisations.
While it is impossible to prove causality from this, the results support the theory that, when an organisation is deciding whether to collaborate with another, the total research funding which that other organisation has received is a factor.

8 Summary and Reflections

8.1 Conclusions

- What is the structure of the ecosystem of publicly funded research - how does it change over time?

Through aggregate analysis and specific analysis of the East Midlands network, a common pattern appears within the ecosystem wherein a few organisations (4252, or 9.92%) and projects (9723, or 10.0%) receive the vast majority of funding from UKRI and its subsidiaries. This pattern applies to all types of organisation, though academic and privately-owned organisations are likelier to be among these top researchers.
Private organisations represent the overwhelming majority of all those taking part in this research.

Analysis of the East Midlands network reveals that the top researching organisations form hubs of research, wherein many organisations collaborate with solely this top organisation. Within the 2002-2008 period, such hubs include:

- The University of Nottingham: 114 collaborations

- Loughborough University: 93
- University Hospitals of Leicester: 68
- Nottingham University Hospitals: 59
- PERA Innovation: 38
- Experian: 17

Comparing two different time periods (2002-2008, and 2005-2010) in this network there reveals not much change within the ecosystem in terms of distribution of collaboration among the involved organisations. However, while the top academic and medical organisations remain at the top, the private organisations involved vary over time. This suggests private organisations do not always maintain the amount of research they are involved with as consistently as these other types of organisations.

- What are the significant factors that influence collaboration between organisations?

Further aggregate analysis revealed that organisations are more likely to collaborate with the top organisation when ranked by either amount of funding received or the number of projects involved.
This suggests a relationship between these attributes and one organisation's decision to collaborate with these top organisations, perhaps due to this indicating the quality or value of the research they engage in.

8.2 Contributions

The project succeeded in going through the entire process of data warehousing and analysis for a set of open data, as originally intended.
Research did not bring up any other works that focused entirely on UKRI's Gateway to Research database and explored the dataset in its entirety. Previous research involved creating systems around such open data, including EnAKTing (Shadbolt et al. (2012)) and Dbpedia (Auer et al. (2007)). While this project did not go as far as creating an entire platform for users to perform their own analysis, I believe some useful analyses were performed and that it provides a valuable starting point from which to create such a platform.

By applying existing computational methods to an entire dataset in this project, I believe it provides a useful case study that supports future research that applies the chosen methods on open datasets. This includes difficulties encountered in applying them, methods for circumventing or alleviating these difficulties, and a data point from which to gauge their efficacy or usefulness.

I also believe the projects provides additional insights into possible improvements to open data platforms that would make similar future research easier. Details provided about de-duplication and schema merging performed during data cleaning could be applied by these open data platforms.
E.g. by showing similar existing records in data entry forms in order to avoid the creation of duplicate records. The project found overlapping semantics in the schema used by the Gateway to Research database, which may be a result of it encapsulating data from several subsidiary organisations. UKRI could apply these findings by reviewing the existing schema and applying changes to reduce this overlap.
Lack of documentation about the Gateway to Research's schema required manual work in exploring the structure of the dataset in order to convert it into a relational schema that could be implemented by standard relational databases. Similarly, documentation about the contents of the database tended to be outdated or lacking in detail. In order to support use of their database by researchers, improving this documentation is a step that UKRI could take.

8.3 Reflections

What the project achieved differs from the original plan, mostly due to the time constrictions of this project.
While the data warehousing tasks were achieved in some capacity, the originally planned analyses had to be cut down for the project to be completed on time. This includes using correlation analysis and machine learning for predicting possible collaboration, identifying technologies being researched, and estimating the market readiness of those technologies.

Time spent on individual tasks was typically longer than planned (with the plan laid out in the Gantt chart below). This was both due to the allocated time being insufficient, and due to less focus being put on the project than it required during the earlier periods of the project. This is main reason for reducing the amount of analysis within the project.
Initially The Decision Project (TDP), a business consultancy, intended to have a more active part in the project by providing additional contextual data about organisations within their own datasets. This would have provided more possibilities for analysing the impact of research on businesses, but required more work be done to incorporate these datasets into the database formed in this project.

While I planned to perform tasks in the order specified in the Methodology, I found myself continually going back to adjust previously done work, such as adjusting the schema or deduplication procedures. This made keeping track of the pace of the project to be difficult, resulting in tasks and priorities continually shifting as the project progressed.
I ceased updating the work plan shown in the Gantt chart, as well not tracking tasks with Taskwarrior[38] as originally planned - I found having to do this regularly as plans shifted both unproductive and demoralising. Instead of setting fixed dates by which to accomplish certain things, I would have preferred to use a more agile project management methodology such as Kanban Ahmad, Markkula, and Oivo (2013). In such a method, the tasks that compose the project could be laid out individually, prioritised, and their dependencies linked. This would allow the project plan to be flexibly adjusted when tasks take a different amount of time to complete than predicted.

I also regret not using version control software from the beginning of the project. I believe it would've provided additional help in managing the pace of work and could have been paired nicely with a Kanban system - where logged changes work towards completing individual tasks. A version control system was applied to the project very late, where most of the benefits were in the ability to tracking changes in documentation.

[38] https://taskwarrior.org

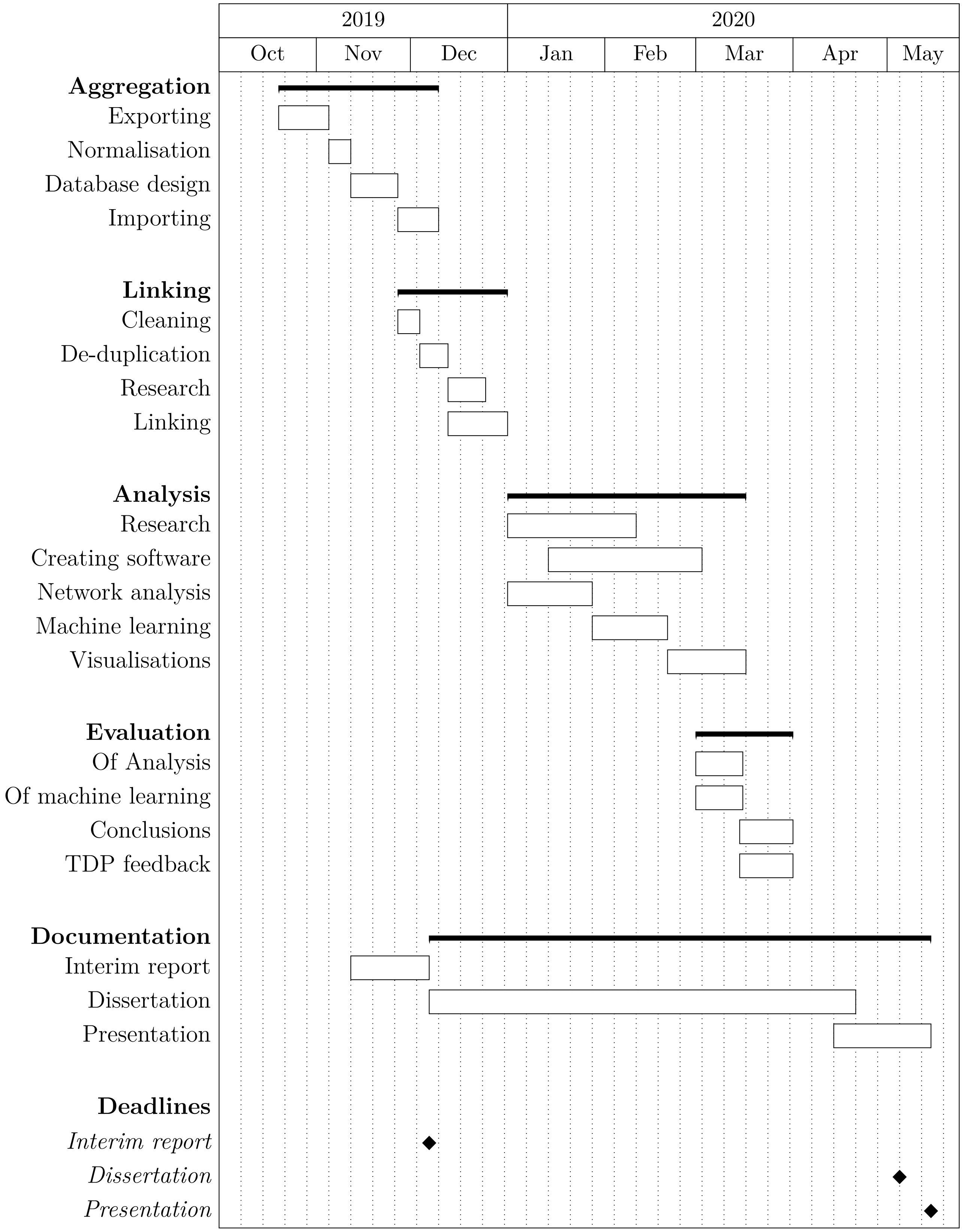

Figure 15: Gannt chart visualising the work plan for the project.

9 Bibliography

Ahmad, Muhammad Ovais, Jouni Markkula, and Markku Oivo. 2013. "Kanban in Software Development: A Systematic Literature Review." In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, 9–16. IEEE.

Auer, Sören, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. "Dbpedia: A Nucleus for a Web of Open Data." In *The Semantic Web*, 722–35. Springer.

Bastian, Mathieu, Sebastien Heymann, and Mathieu Jacomy. 2009. "Gephi: An Open Source Software for Exploring and Manipulating Networks." http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154.

Dornum, Deirdre Dionysia von. 1997. "The Straight and the Crooked: Legal Accountability in Ancient Greece." *Columbia Law Review*, 1483–1518.

Elmagarmid, Ahmed K, Panagiotis G Ipeirotis, and Vassilios S Verykios. 2006. "Duplicate Record Detection: A Survey." *IEEE Transactions on Knowledge and Data Engineering* 19 (1): 1–16.

Fruchterman, Thomas MJ, and Edward M Reingold. 1991. "Graph Drawing by Force-Directed Placement." *Software: Practice and Experience* 21 (11): 1129–64.

Gateway to Research Api 2. 2018. Version 1.7.4. UK Research and Innovation. https://gtr.ukri.org/resources/GtR-2-API-v1.7.4.pdf.

GtR Data Dictionary. n.d. Version 8.0. UK Research and Innovation. https://gtr.ukri.org/resources/GtRDataDictionary.pdf.

Hu, Yifan. n.d. "Efficient and High Quality Force-Directed Graph Drawing." Wolfram Research Inc. http://yifanhu.net/PUB/graph_draw_small.pdf.

Lathrop, D., and L. Ruma. 2010. Theory in Practice. O'Reilly Media.

Lee, Heeseok. 1995. "Justifying Database Normalization: A Cost/Benefit Model." *Information Processing & Management* 31 (1): 59–67.

Luce, R Duncan, and Albert D Perry. 1949. "A Method of Matrix Analysis of Group Structure." *Psychometrika* 14 (2): 95–116.

Manning, Christopher, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. "The Stanford Corenlp Natural Language Processing Toolkit." In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 55–60.

"Nature of Business: Standard Industrial Classification (Sic) Codes." n.d. Companies House. https://resources.companieshouse.gov.uk/sic.

"Open Definition 2.1." 2019. Open Knowledge Foundation. 2019. https://opendefinition.org/od/2.1/en/.

Opsahl, Tore, and Pietro Panzarasa. 2009. "Clustering in Weighted Networks." *Social Networks* 31 (2): 155–63.

Rahm, Erhard, and Hong Hai Do. 2000. "Data Cleaning: Problems and Current Approaches." *IEEE Data Eng. Bull.* 23 (4): 3–13.

Schauer, Frederick. 2011. "Transparency in Three Dimensions." *University of Illinois Law Review*, 1339–58.

Shadbolt, Nigel, Kieron O'Hara, Tim Berners-Lee, Nicholas Gibbins, Hugh Glaser, Wendy Hall, and others. 2012. "Linked Open Government Data: Lessons from Data.gov.uk." *IEEE Intelligent Systems* 27 (3): 16–24.

Winkler, William E. 2006. "Overview of Record Linkage and Current Research Directions." In *Bureau of the Census*. Citeseer.

"XML Path Language (Xpath) 3.1." 2017. Standard. World Wide Web Consortium. https://www.w3.org/TR/2017/REC-xpath-31-20170321.

10 Appendix

10.1 ukriDownload

```
#!/usr/bin/env bash
URL='https://gtr.ukri.org/gtr/api/'
PAUSE=4
NUM_PAGES_XPATH='string(/*/@*[local-name()="totalPages"])'
OUTPUT='.'

# Parse options: https://stackoverflow.com/a/14203146
TYPES=()
while [[ $# -gt 0 ]]; do
    key="$1"

    case $key in
        -o|--output)      OUTPUT="$2";          shift 2;;
        -u|--url)         URL="$2";             shift 2;;
        -p|--pause)       PAUSE="$2";           shift 2;;
        -x|--pages-xpath) NUM_PAGES_XPATH="$2"; shift 2;;
        *)                TYPES+=("$1");        shift 1;;
    esac
done

set -- "${TYPES[@]}"

if [ "${#TYPES[@]}" == 0 ]; then
    echo "No record types specified"
    exit 1
fi

for typ in "$TYPES"; do
    case "$typ" in
        or*) recordType='organisations';;
        ou*) recordType='outcomes';;
        pe*) recordType='persons';;
        pr*) recordType='projects';;
        *) (>&2 echo "Unknown record type '$typ' - skipping"); continue;;
    esac

    url="$URL$recordType"
    mkdir -p "$OUTPUT/$recordType"

    function getPageURL {
        echo "$url?p=$1&s=100"
    }

    function getPageFile {
        echo "$OUTPUT/$recordType/$(printf '%04d' "$1").xml"
    }

    function xmlCount {
        awk '{s+=$1} END {print s}' <(xmllint --xpath 'count(/*/*)' $*)
    }

    file1="$(getPageFile 1)"
    curl "$(getPageURL 1)" -so "$file1"
    numPages=$(xmllint --xpath "$NUM_PAGES_XPATH" "$file1")

    echo "Total pages: $numPages"
```

```

    for i in $(seq 2 $numPages); do
        file="$(getPageFile $i)"
        [ -f "$file" ] && continue
        sleep "$PAUSE"
        echo "Downloading page $i of $numPages"
        curl "$(getPageURL $i)" -so "$file"
        [ $? ] || break
    done

    merged="$OUTPUT/$recordType.xml"
    pages="$OUTPUT/$recordType/*.xml"
    if ! [ -f "$merged" ] || \
       ! [ "$(xmlCount "$merged")" = "$(xmlCount "$pages")" ]; then
        echo "Merging into a single file..."
        "$(basename $0)"/mergeXML "$pages" >"$merged"
    else
        echo "Merged file contains all records: $merged"
    fi
done
```

10.2 mergeXML

```
#!/usr/bin/env python
# Adapted from https://stackoverflow.com/a/11315257
import sys

from xml.etree import ElementTree

def run(files):
    first = None

    for filename in files:
        data = ElementTree.parse(filename).getroot()

        if first is None:
            first = data
        else:
            first.extend(data)

    if first is not None:
        print(ElementTree.tostring(first, encoding='utf8').decode('utf-8'))

if __name__ == "__main__":
    run(sys.argv[1:])
```

10.3 ukriXmlToCsv

```
#!/usr/bin/env python3
'''Convert a UKRI XML dataset to CSV.
Expected format is:
<records>
    <record></record>
    ...
</records>

Children of records will be flattened, with only the first sub-child being kept.
```

```
'''

import argparse, csv, re, io, sys, os, xml.etree.ElementTree as ET
from typing import List, Optional
import os.path as path

usage_example = '''example: \
    ukriXmlToCsv \
        --fields name line1 line2 line3 line4 line5 postCode country \
        --output . \
        organisations.xml'''

parser = argparse.ArgumentParser(
    description='Convert UKRI XML record collections to CSV',
    epilog=usage_example)
parser.add_argument('--fields', '-f',
    nargs='+', metavar='pattern', action='extend', type=str,
    help='patterns of fields to include in the output')
parser.add_argument('--output', '-o',
    default='.',
    help='output directory for the converted files')
parser.add_argument('files',
    metavar='file', nargs='+',
    help='XML files to convert')

args = parser.parse_args()

os.makedirs(args.output, exist_ok=True)

def strip_namespace(field_tag: str) -> str:
    '''Remove the namespace from a field's tag'''
    i = field_tag.find('}')
    if i >= 0:
        field_tag = field_tag[i + 1:]
    return field_tag

def match_field_name(field: ET.Element) -> Optional[str]:
    '''Return whether a field with a specific tag should be filtered, according
    to the ignored field filters'''
    field_name = strip_namespace(field.tag)
    for pattern in args.fields:
        if re.match(pattern, field_name):
            return pattern
    return None

def filter_fields(fields: List[ET.Element]):
    return list(filter(lambda f: match_field_name(f) != None, fields))

def xml_to_csv(fname: str) -> str:
    output = io.StringIO()
    tree = ET.parse(fname, parser=ET.XMLParser(encoding='utf-8'))

    writer = csv.DictWriter(output, fieldnames=args.fields, delimiter=',')
    writer.writeheader()

    for record in tree.getroot():
        record_dict: dict[(str, str)] = dict()
        fields = filter_fields(list(record))

        # Flatten fields
```

```python
        for field in record:
            children = list(field)
            if children:
                if len(children) > 1:
                    print('Warning: more than one child in flattened field ' \
                          f"{field.tag}", file=sys.stderr)
                fields.extend(filter_fields(field[0]))

        for field in fields:
            field_name = match_field_name(field)
            if field_name:
                record_dict[field_name] = field.text.strip()

        writer.writerow(record_dict)

    return output.getvalue()

for f in args.files:
    csv = xml_to_csv(f)
    csv_name = path.join(args.output,
        path.splitext(path.basename(f))[0] + '.csv')

    with open(csv_name, 'w', encoding='utf-8') as out:
        out.write(csv)
```

10.4 setup.sql

```sql
-- Set up the database schema
-- Execute using:
-- $ psql -U <username> -d <database> -f setup.sql

-- Provides UUID data types
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Provides similarity comparisons
CREATE EXTENSION IF NOT EXISTS "pg_trgm";

-- Provides crosstab()
CREATE EXTENSION IF NOT EXISTS "tablefunc";

-- GtR organisations
-- based on https://gtr.ukri.org/gtr/api
-- and https://gtr.ukri.org/gtr/api/organisation

DO $$ BEGIN
    CREATE TYPE gtrRegion as ENUM(
        'Channel Islands/Isle of Man',
        'East Midlands',
        'East of England',
        'London',
        'North East',
        'Northern Ireland',
        'North West',
        'Outside UK',
        'Scotland',
        'South East',
        'South West',
        'Unknown',
        'Wales',
```

```sql
        'West Midlands',
        'Yorkshire and The Humber'
    );
EXCEPTION
    WHEN duplicate_object THEN null;
END $$; -- BEGIN

DO $$ BEGIN
    CREATE TYPE gtrSector as ENUM(
        'Academic/University',
        'Charity/Non-profit',
        'Public',
        'Private'
    );
EXCEPTION
    WHEN duplicate_object THEN null;
END $$; -- BEGIN

CREATE TABLE IF NOT EXISTS gtrOrgs(
    orgUuid uuid PRIMARY KEY NOT NULL,
    name text not null,
    address1 text,
    address2 text,
    address3 text,
    address4 text,
    address5 text,
    postCode text,
    city text,
    region gtrRegion not null default 'Unknown',
    country text,
    recorded date
);

-- GtR projects
-- based on https://gtr.ukri.org/gtr/api/project

DO $$ BEGIN
    CREATE TYPE gtrProjectStatus as ENUM(
        'Active',
        'Closed'
    );
EXCEPTION
    WHEN duplicate_object THEN null;
END $$; -- BEGIN

DO $$ BEGIN
    CREATE TYPE gtrGrantCategory as ENUM(
        'BIS-Funded Programmes',
        'Centres',
        'Collaborative R&D',
        'CR&D Bilateral',
        'EU-Funded',
        'European Enterprise Network',
        'Fast Track',
        'Feasibility Studies',
        'Fellowship',
        'GRD Development of Prototype',
        'GRD Proof of Concept',
        'GRD Proof of Market',
        'Intramural',
```

```
        'Knowledge Transfer Network',
        'Knowledge Transfer Partnership',
        'Large Project',
        'Launchpad',
        'Legacy Department of Trade & Industry',
        'Legacy RDA Collaborative R&D',
        'Legacy RDA Grant for R&D',
        'Missions',
        'Other Grant',
        'Procurement',
        'Research Grant',
        'Small Business Research Initiative',
        'SME Support',
        'Special Interest Group',
        'Studentship',
        'Study',
        'Third Party Grant',
        'Training Grant',
        'Unknown',
        'Vouchers'
    );
EXCEPTION
    WHEN duplicate_object THEN null;
END $$; -- BEGIN

DO $$ BEGIN
    CREATE TYPE gtrFunder as ENUM(
        'AHRC',
        'BBSRC',
        'EPSRC',
        'ESRC',
        'Innovate UK',
        'MRC',
        'NC3Rs',
        'NERC',
        'STFC',
        'UKRI'
    );
EXCEPTION
    WHEN duplicate_object THEN null;
END $$; -- BEGIN

CREATE TABLE IF NOT EXISTS gtrProjects(
    projectUuid uuid PRIMARY KEY NOT NULL,
    title text not null,
    status gtrProjectStatus,
    category gtrGrantCategory,
    leadFunder gtrFunder,
    abstract text,
    techAbstract text,
    potentialImpact text,
    startDate date,
    endDate date,
    recorded date
);

CREATE TABLE IF NOT EXISTS gtrSubjects(
    subjectUuid uuid PRIMARY KEY NOT NULL,
    name text
);
```

```sql

CREATE TABLE IF NOT EXISTS gtrProjectSubjects(
    subjectUuid uuid REFERENCES gtrSubjects,
    projectUuid uuid REFERENCES gtrProjects,
    percent numeric,
    PRIMARY KEY (subjectUuid, projectUuid)
);

CREATE TABLE IF NOT EXISTS gtrTopics(
    topicUuid uuid PRIMARY KEY NOT NULL,
    name text
);

CREATE TABLE IF NOT EXISTS gtrProjectTopics(
    topicUuid uuid REFERENCES gtrTopics,
    projectUuid uuid REFERENCES gtrProjects,
    percent numeric,
    PRIMARY KEY (topicUuid, projectUuid)
);

-- GtR outcomes
-- based on https://gtr.ukri.org/gtr/api/outcome

CREATE TABLE IF NOT EXISTS gtrDisseminations(
    disUuid uuid PRIMARY KEY NOT NULL,
    title text,
    description text,
    form text,
    primaryAudience text,
    yearsOfDissemination text,
    results text,
    impact text,
    typeOfPresentation text,
    geographicReach text,
    partOfOfficialScheme boolean,
    supportingUrl text
);

CREATE TABLE IF NOT EXISTS gtrCollaborations(
    collabUuid uuid PRIMARY KEY NOT NULL,
    description text,
    parentOrganisation text,
    childOrganisation text,
    principalInvestigatorContribution text,
    partnerContribution text,
    startDate date,
    endDate date,
    sector gtrSector,
    country text,
    impact text,
    supportingUrl text
);

CREATE TABLE IF NOT EXISTS gtrKeyFindings(
    keyFindingUuid uuid PRIMARY KEY NOT NULL,
    description text,
    nonAcademicUses text,
    exploitationPathways text,
    sectors text,
    supportingUrl text
```

```sql
);

CREATE TABLE IF NOT EXISTS gtrFurtherFundings(
    furtherFundingUuid uuid PRIMARY KEY NOT NULL,
    title text,
    description text,
    narrative text,
    amount money,
    organisation text,
    department text,
    fundingId text,
    startDate date,
    endDate date,
    sector gtrSector,
    country text
);

CREATE TABLE IF NOT EXISTS gtrImpactSummaries(
    impactSummaryUuid uuid PRIMARY KEY NOT NULL,
    title text,
    description text,
    impactTypes text,
    summary text,
    beneficiaries text,
    contributionMethod text,
    sector gtrSector,
    firstYearOfImpact int
);

CREATE TABLE IF NOT EXISTS gtrPolicyInfluences(
    policyInfluenceUuid uuid PRIMARY KEY NOT NULL,
    influence text,
    type text,
    guidelineTitle text,
    impact text,
    methods text,
    areas text,
    geographicReach text,
    supportingUrl text
);

CREATE TABLE IF NOT EXISTS gtrResearchMaterials(
    researchMatUuid uuid PRIMARY KEY NOT NULL,
    title text,
    description text,
    type text,
    impact text,
    softwareDeveloped boolean,
    softwareOpenSourced boolean,
    providedToOthers boolean,
    yearFirstProvided int,
    supportingUrl text
);

-- GtR persons
-- based on https://gtr.ukri.org/gtr/api/person

CREATE TABLE IF NOT EXISTS gtrPeople(
    personUuid uuid PRIMARY KEY NOT NULL,
    firstName text,
```

```sql
    otherNames text,
    surname text,
    email text,
    orcId text
);

-- Similarity indices

CREATE INDEX IF NOT EXISTS gtrOrgs_name_gist ON gtrOrgs USING gist(
    name gist_trgm_ops
);

CREATE INDEX IF NOT EXISTS gtrPeople_name_gist ON gtrPeople USING gist(
    firstName gist_trgm_ops,
    surname gist_trgm_ops
);

-- GtR links

DO $$ BEGIN
    CREATE TYPE gtrPersonRole as ENUM(
        'Principal Investigator',
        'Co-Investigator',
        'Project Manager',
        'Fellow',
        'Training Grant Holder',
        'Primary Supervisor',
        -- These do not appear in the API reference, so may be deprecated
        'Researcher Co-Investigator',
        'Researcher'
    );
EXCEPTION
    WHEN duplicate_object THEN null;
END $$; -- BEGIN

DO $$ BEGIN
    CREATE TYPE gtrOrgRole as ENUM(
        'Lead',
        'Collaborating',
        'Fellow',
        'Project Partner',
        'Funder',
        'Co-Funder',
        'Participant',
        'Student Project Partner'
    );
EXCEPTION
    WHEN duplicate_object THEN null;
END $$; -- BEGIN

DO $$ BEGIN
    CREATE TYPE gtrProjectRelation as ENUM(
         'TRANSFER',
         'STUDENTSHIP_FROM',
         'TRANSFER_FROM',
         'STUDENTSHIP'
    );
EXCEPTION
    WHEN duplicate_object THEN null;
END $$; -- BEGIN
```

```sql

CREATE TABLE IF NOT EXISTS gtrOrgPeople(
    personUuid uuid REFERENCES gtrPeople,
    orgUuid uuid REFERENCES gtrOrgs,
    PRIMARY KEY (personUuid, orgUuid)
);

CREATE TABLE IF NOT EXISTS gtrProjectPeople(
    projectUuid uuid REFERENCES gtrProjects,
    personUuid uuid REFERENCES gtrPeople,
    role gtrPersonRole,
    PRIMARY KEY (projectUuid, personUuid, role)
);

CREATE TABLE IF NOT EXISTS gtrProjectOrgs(
    projectUuid uuid REFERENCES gtrProjects,
    orgUuid uuid REFERENCES gtrOrgs,
    role gtrOrgRole NOT NULL,
    startDate date,
    endDate date,
    cost money,
    offer money,
    PRIMARY KEY (projectUuid, orgUuid, role)
);

CREATE TABLE IF NOT EXISTS gtrRelatedProjects(
    projectUuid1 uuid REFERENCES gtrProjects,
    projectUuid2 uuid REFERENCES gtrProjects,
    relation gtrProjectRelation NOT NULL,
    startDate date,
    endDate date,
    PRIMARY KEY (projectUuid1, projectUuid2, relation)
);

-- GtR cleaning

CREATE TABLE IF NOT EXISTS junkGtrOrgs(
    orgUuid uuid REFERENCES gtrOrgs PRIMARY KEY
);

CREATE TABLE IF NOT EXISTS similarGtrOrgs(
    orgUuid1 uuid REFERENCES gtrOrgs,
    orgUuid2 uuid REFERENCES gtrOrgs,
    simTrigramName float,
    simTrigramAddress float,
    manualResult bool,
    PRIMARY KEY (orgUuid1, orgUuid2)
);

CREATE TABLE IF NOT EXISTS similarGtrPeople(
    personUuid1 uuid REFERENCES gtrPeople,
    personUuid2 uuid REFERENCES gtrPeople,
    simTrigram float,
    manualResult bool,
    PRIMARY KEY (personUuid1, personUuid2)
);

-- Merged organisations

CREATE TABLE IF NOT EXISTS duplicateGtrOrgs(
```

```
    orgUuid uuid REFERENCES gtrOrgs,
    duplicateUuid uuid UNIQUE REFERENCES gtrOrgs,
    PRIMARY KEY (orgUuid, duplicateUuid)
);

DO $$ BEGIN
    CREATE TYPE orgType as ENUM(
        'Academic',
        'Medical',
        'Private',
        'Public',
        'Unknown'
    );
EXCEPTION
    WHEN duplicate_object THEN null;
END $$; -- BEGIN

CREATE TABLE IF NOT EXISTS orgs(
    gtrOrgUuid uuid PRIMARY KEY NOT NULL,
    name text not null,
    address1 text,
    address2 text,
    address3 text,
    address4 text,
    address5 text,
    postCode text,
    city text,
    region gtrRegion NOT NULL DEFAULT 'Unknown',
    country text,
    type orgType DEFAULT 'Unknown'
);

-- Merged people

CREATE TABLE IF NOT EXISTS duplicateGtrPeople(
    personUuid uuid REFERENCES gtrPeople,
    duplicateUuid uuid UNIQUE REFERENCES gtrPeople,
    PRIMARY KEY (personUuid, duplicateUuid)
);

CREATE TABLE IF NOT EXISTS people(
    gtrPersonUuid uuid PRIMARY KEY NOT NULL,
    firstName text,
    surname text,
    otherNames text
);

-- Utility functions

-- Remove all whitespace in some text
CREATE OR REPLACE
FUNCTION STRIP(txt text)
RETURNS text AS $$
BEGIN
    RETURN regexp_replace(txt, '\s', '');
END; $$ -- FUNCTION
LANGUAGE plpgsql IMMUTABLE;

-- Generate a list of percentile fractions in the specified range
CREATE OR REPLACE
```

```sql
FUNCTION PERCENTILE_FRACTIONS(
    lowerBound int,
    upperBound int,
    step int DEFAULT 1
)
RETURNS numeric[] AS $$
BEGIN
    RETURN (SELECT ARRAY(
        SELECT (a.n::numeric) / 100
        FROM GENERATE_SERIES(lowerBound, upperBound, step) AS a(n)
    ));
END; $$ -- FUNCTION
LANGUAGE plpgsql IMMUTABLE;

CREATE OR REPLACE FUNCTION CoalesceAll(state anyelement, new anyelement)
RETURNS anyelement
AS $$
BEGIN
    RETURN COALESCE(state, new);
END; $$ --FUNCTION
LANGUAGE plpgsql IMMUTABLE;

CREATE OR REPLACE AGGREGATE Merge(anyelement) (
    sfunc = CoalesceAll,
    stype = anyelement
);
```

10.5 importGtrOrgs.sql

```sql
-- Run with
-- $ psql -U <username> -d <database> -f importGtrOrgs.sql
-- Ensure the organisations.xml file exists in the PostgreSQL data directory
DO $$
DECLARE
    file text = 'organisations.xml';
    xmlRecords xml;
    org record;
    nss text[][2] := array[
        array['api', 'http://gtr.rcuk.ac.uk/gtr/api'],
        array['org', 'http://gtr.rcuk.ac.uk/gtr/api/organisation']
    ];
BEGIN

xmlRecords := XMLPARSE(
    DOCUMENT convert_from(pg_read_binary_file(file), 'UTF8'));

DROP TABLE IF EXISTS xmlImport;
CREATE TEMP TABLE xmlImport AS
SELECT
    (xpath('//@api:id', x, nss))[1]::text AS uuid,
    (xpath('//@api:created', x, nss))[1]::text::date AS recorded,
    (xpath('//org:name/text()', x, nss))[1]::text AS name,
    (xpath('//org:addresses[1]/api:address/api:line1/text()',    x, nss))[1]::text AS line1,
    (xpath('//org:addresses[1]/api:address/api:line2/text()',    x, nss))[1]::text AS line2,
    (xpath('//org:addresses[1]/api:address/api:line3/text()',    x, nss))[1]::text AS line3,
    (xpath('//org:addresses[1]/api:address/api:line4/text()',    x, nss))[1]::text AS line4,
    (xpath('//org:addresses[1]/api:address/api:line5/text()',    x, nss))[1]::text AS line5,
    (xpath('//org:addresses[1]/api:address/api:postCode/text()', x, nss))[1]::text AS
    ↪  postCode,
```

```
    (xpath('//org:addresses[1]/api:address/api:city/text()',     x, nss))[1]::text AS city,
    (xpath('//org:addresses[1]/api:address/api:region/text()',   x, nss))[1]::text AS
    ↪ region,
    (xpath('//org:addresses[1]/api:address/api:country/text()',  x, nss))[1]::text AS
    ↪ country
FROM unnest(xpath('//org:organisations/org:organisation', xmlRecords, nss)) x;

FOR org IN
    SELECT * FROM xmlImport
LOOP
    INSERT INTO gtrOrgs VALUES(
        uuid(org.uuid),
        replace(org.name, '&amp;', '&'),
        org.line1,
        org.line2,
        org.line3,
        org.line4,
        org.line5,
        org.postCode,
        org.city,
        COALESCE(gtrRegion(org.region), gtrRegion('Unknown')),
        org.country,
        org.recorded
    ) ON CONFLICT DO UPDATE SET
        recorded = org.recorded;
END LOOP;

END$$;

DROP TABLE IF EXISTS xmlImport;
```

10.6 importGtrPersons.sql

```
-- Run with
-- $ psql -U <username> -d <database> -f importGtrPersons.sql
-- Ensure the persons.xml file exists in the PostgreSQL data directory
DO $$
DECLARE
    file text := 'persons.xml';
    xmlRecords xml;
    per record;
    link record;
    nss text[][2] := array[
        array['api', 'http://gtr.rcuk.ac.uk/gtr/api'],
        array['per', 'http://gtr.rcuk.ac.uk/gtr/api/person']
    ];
BEGIN

xmlRecords := XMLPARSE(
    DOCUMENT convert_from(pg_read_binary_file(file), 'UTF8'));

DROP TABLE IF EXISTS xmlImport;
CREATE TEMP TABLE xmlImport AS
SELECT
    (xpath('//@api:id', x, nss))[1]::text::uuid AS uuid,
    (xpath('//per:firstName/text()', x, nss))[1]::text AS firstName,
    (xpath('//per:otherNames/text()', x, nss))[1]::text AS otherNames,
    (xpath('//per:surname/text()', x, nss))[1]::text AS surname,
    (xpath('//per:email/text()', x, nss))[1]::text AS email,
```

```
    (xpath('//per:orcidId/text()', x, nss))[1]::text AS orcId,
    (xpath('//api:links/api:link[@api:rel="EMPLOYED"]/@api:href', x, nss))[1]::text AS
    ↪  employerLinks,
    (xpath('//api:links', x, nss))[1] AS links
FROM unnest(xpath('//per:persons/per:person', xmlRecords, nss)) x;

FOR per IN
    SELECT * FROM xmlImport
LOOP
    INSERT INTO gtrPeople VALUES(
        per.uuid,
        per.firstName,
        per.otherNames,
        per.surname,
        per.email,
        per.orcId
    ) ON CONFLICT (personUuid) DO UPDATE SET
        firstName  = per.firstName,
        otherNames = per.otherNames,
        surname    = per.surname,
        email      = per.email,
        orcId      = per.orcId;

    IF per.employerLinks IS NOT NULL THEN
        INSERT INTO gtrOrgPeople VALUES(
            per.uuid,
            -- Extract organisation UUID from HREF
            substring(per.employerLinks, '/organisations/(.*)$')::uuid
        ) ON CONFLICT DO NOTHING;
    END IF;

    FOR link IN
        SELECT
            (xpath('//@api:rel', x, nss))[1]::text AS rel,
            (xpath('//@api:href', x, nss))[1]::text AS href
        FROM unnest(xpath('//api:link', per.links, nss)) x
    LOOP
        IF EXISTS (SELECT projectUuid FROM gtrProjects WHERE
            projectUuid = substring(link.href, '/projects/(.*)$')::uuid)
        THEN
            INSERT INTO gtrProjectPeople VALUES(
                substring(link.href, '/projects/(.*)$')::uuid,
                per.uuid,
                (CASE
                    WHEN link.rel = 'PI_PER'           THEN 'Principal Investigator'
                    WHEN link.rel = 'COI_PER'          THEN 'Co-Investigator'
                    WHEN link.rel = 'PM_PER'           THEN 'Project Manager'
                    WHEN link.rel = 'FELLOW_PER'       THEN 'Fellow'
                    WHEN link.rel = 'TGH_PER'          THEN 'Training Grant Holder'
                    WHEN link.rel = 'SUPER_PER'        THEN 'Primary Supervisor'
                    WHEN link.rel = 'RESERACH_COI_PER' THEN 'Researcher Co-Investigator'
                    WHEN link.rel = 'RESERACH_PER'     THEN 'Researcher'
                    ELSE link.rel::gtrPersonRole
                END)::gtrPersonRole
            ) ON CONFLICT DO NOTHING;
        END IF;
    END LOOP;
END LOOP;

END$$;
```

10.7 importGtrProjects.sql

```sql
-- Run with(xmlImport,(xmlImport,
-- $ psql -U <username> -d <database> -f importGtrProjects.sql
-- Ensure the projects.xml file exists in the PostgreSQL data directory

CREATE OR REPLACE
FUNCTION ParseOrgRole(txt text)
RETURNS gtrOrgRole AS $$
BEGIN
    RETURN (CASE
        -- Project participants
        WHEN txt = 'LEAD_PARTICIPANT' THEN 'Lead'
        WHEN txt = 'PARTICIPANT'      THEN 'Participant'
        -- Project links
        WHEN txt = 'LEAD_ORG'         THEN 'Lead'
        WHEN txt = 'COLLAB_ORG'       THEN 'Collaborating'
        WHEN txt = 'FELLOW_ORG'       THEN 'Fellow'
        WHEN txt = 'PP_ORG'           THEN 'Project Partner'
        WHEN txt = 'FUNDER'           THEN 'Funder'
        WHEN txt = 'COFUND_ORG'       THEN 'Co-Funder'
        WHEN txt = 'PARTICIPANT_ORG'  THEN 'Participant'
        WHEN txt = 'STUDENT_PP_ORG'   THEN 'Student Project Partner'
        ELSE txt
    END)::gtrOrgRole;
END; $$ -- FUNCTION
LANGUAGE plpgsql IMMUTABLE;

DO $$
DECLARE
    file text := 'projects.xml';
    xmlRecords xml;
    rec record;
    nss text[][2] := array[
        array['api', 'http://gtr.rcuk.ac.uk/gtr/api'],
        array['pro', 'http://gtr.rcuk.ac.uk/gtr/api/project']
    ];
BEGIN

xmlRecords := XMLPARSE(
    DOCUMENT convert_from(pg_read_binary_file(file), 'UTF8'));

CREATE TEMP TABLE IF NOT EXISTS xmlImport AS
SELECT
    (xpath('//@api:id', x, nss))[1]::text::uuid AS uuid,
    (xpath('//@api:created', x, nss))[1]::text::date AS recorded,
    (xpath('//pro:title/text()', x, nss))[1]::text AS title,
    (xpath('//pro:status/text()', x, nss))[1]::text AS status,
    (xpath('//pro:grantCategory/text()', x, nss))[1]::text AS category,
    (xpath('//pro:leadFunder/text()', x, nss))[1]::text AS funder,
    (xpath('//pro:abstract/text()', x, nss))[1]::text AS abstract,
    (xpath('//pro:techAbstract/text()', x, nss))[1]::text AS techAbstract,
    (xpath('//pro:potentialImpact/text()', x, nss))[1]::text AS potentialImpact,
    (xpath('//api:links/api:link[@api:rel="FUND"]/@api:start', x, nss))[1]::text::date AS startDate,
    (xpath('//api:links/api:link[@api:rel="FUND"]/@api:end', x, nss))[1]::text::date AS endDate,
    (xpath('//api:links', x, nss))[1] AS links,
    (xpath('//pro:participantValues', x, nss))[1] AS participants,
    (xpath('//pro:researchSubjects', x, nss))[1] AS subjects,
```

```
    (xpath('//pro:researchTopics', x, nss))[1] AS topics
FROM unnest(xpath('//pro:projects/pro:project', xmlRecords, nss)) x;

-- Projects

INSERT INTO gtrProjects
SELECT
    uuid,
    REPLACE(title, '&amp;', '&'),
    status::gtrProjectStatus,
    -- XML entities aren't parsed, so manually convert ampersands
    REPLACE(category, '&amp;', '&')::gtrGrantCategory,
    funder::gtrFunder,
    abstract,
    techAbstract,
    potentialImpact,
    startDate,
    endDate,
    recorded
FROM xmlImport
ON CONFLICT (projectUuid) DO NOTHING;

-- Project organisations

INSERT INTO gtrProjectOrgs(
    projectUuid,
    orgUuid,
    role,
    cost,
    offer
)
SELECT
    proUuid,
    (xpath('//pro:organisationId/text()', x, nss))[1]::text::uuid AS orgUuid,
    ParseOrgRole((xpath('//pro:role/text()', x, nss))[1]::text) AS role,
    (xpath('//pro:projectCost/text()', x, nss))[1]::text::numeric::money AS cost,
    (xpath('//pro:grantOffer/text()', x, nss))[1]::text::numeric::money AS offer
FROM (
    SELECT
        uuid AS proUuid,
        unnest(xpath('//pro:participant', participants, nss)) AS x
    FROM xmlImport
) q
ON CONFLICT DO NOTHING;

-- Subjects

FOR rec IN
    SELECT
        proUuid,
        (xpath('//pro:id/text()', x, nss))[1]::text::uuid AS subUuid,
        (xpath('//pro:text/text()', x, nss))[1]::text AS name,
        (xpath('//pro:percentage/text()', x, nss))[1]::text::numeric AS percent
    FROM (
        SELECT
            uuid AS proUuid,
            unnest(xpath('//pro:researchSubject', subjects, nss)) x
        FROM xmlImport
    ) q
LOOP
```

```
    INSERT INTO gtrSubjects VALUES(
        rec.subUuid,
        rec.name
    ) ON CONFLICT DO NOTHING;

    INSERT INTO gtrProjectSubjects VALUES(
        rec.subUuid,
        rec.proUuid,
        rec.percent
    ) ON CONFLICT DO NOTHING;
END LOOP;

-- Topics

FOR rec IN
    SELECT
        proUuid,
        (xpath('//pro:id/text()', x, nss))[1]::text::uuid AS topUuid,
        (xpath('//pro:text/text()', x, nss))[1]::text AS name,
        (xpath('//pro:percentage/text()', x, nss))[1]::text::numeric AS percent
    FROM (
        SELECT
            uuid AS proUuid,
            unnest(xpath('//pro:researchTopic', topics, nss)) x
        FROM xmlImport
    ) q
LOOP
    INSERT INTO gtrTopics VALUES(
        rec.topUuid,
        rec.name
    ) ON CONFLICT DO NOTHING;

    INSERT INTO gtrProjectTopics VALUES(
        rec.topUuid,
        rec.proUuid,
        rec.percent
    ) ON CONFLICT DO NOTHING;
END LOOP;

-- Links

FOR rec IN
    SELECT
        proUuid,
        (xpath('//@api:rel', x, nss))[1]::text AS rel,
        href,
        (xpath('//@api:start', x, nss))[1]::text::date AS startDate,
        (xpath('//@api:end', x, nss))[1]::text::date AS endDate,
        (SELECT orgUuid FROM gtrOrgs
            WHERE orgUuid = substring(href, '/organisations/(.*)$')::uuid
            LIMIT 1) AS orgUuid,
        (SELECT proUuid FROM gtrProjects
            WHERE projectUuid = substring(href, '/projects/(.*)$')::uuid
            LIMIT 1) AS proUuid2
    FROM (
        SELECT
            *,
            (xpath('//@api:href', x, nss))[1]::text AS href
        FROM (
            SELECT
```

```
                uuid AS proUuid,
                unnest(xpath('//api:link', links, nss)) x
            FROM xmlImport
        ) q2
    ) q
LOOP
    -- Linked organisations
    IF rec.orgUuid IS NOT NULL THEN
        INSERT INTO gtrProjectOrgs VALUES(
            rec.proUuid,
            rec.orgUuid,
            ParseOrgRole(rec.rel),
            rec.startDate,
            rec.endDate,
            NULL,
            NULL
        ) ON CONFLICT DO NOTHING;

    -- Related projects
    ELSIF rec.proUuid2 IS NOT NULL THEN
        INSERT INTO gtrRelatedProjects VALUES(
            rec.proUuid,
            rec.proUuid2,
            rec.rel::gtrProjectRelation,
            rec.startDate,
            rec.endDate
        ) ON CONFLICT DO NOTHING;
    END IF;
END LOOP;

END$$;
```

10.8 importGtrOutcomes.sql

```
-- Run with
-- $ psql -U <username> -d <database> -f importGtrOutcomes.sql
-- Ensure the outcomes.xml file exists in the PostgreSQL data directory
DO $$
DECLARE
    file text := 'outcomes.xml';
    xmlRecords xml;
    outcome record;
    nss text[][2] := array[
        array['api', 'http://gtr.rcuk.ac.uk/gtr/api'],
        array['out', 'http://gtr.rcuk.ac.uk/gtr/api/project/outcome']
    ];
BEGIN

xmlRecords := XMLPARSE(
    DOCUMENT convert_from(pg_read_binary_file(file), 'UTF8'));

-- Disseminations

DROP TABLE IF EXISTS xmlImport;
CREATE TEMP TABLE xmlImport AS
SELECT
    (xpath('//@api:id', x, nss))[1]::text AS uuid,
    (xpath('//out:title/text()', x, nss))[1]::text AS title,
    (xpath('//out:description/text()', x, nss))[1]::text AS description,
```

```
    (xpath('//out:form/text()', x, nss))[1]::text AS form,
    (xpath('//out:primaryAudience/text()', x, nss))[1]::text AS primaryAudience,
    (xpath('//out:yearsOfDissemination/text()', x, nss))[1]::text AS yearsOfDissemination,
    (xpath('//out:results/text()', x, nss))[1]::text AS results,
    (xpath('//out:impact/text()', x, nss))[1]::text AS impact,
    (xpath('//out:typeOfPresentation/text()', x, nss))[1]::text AS typeOfPresentation,
    (xpath('//out:geographicReach/text()', x, nss))[1]::text AS geographicReach,
    (xpath('//out:partOfOfficialScheme/text()', x, nss))[1]::text AS partOfOfficialScheme,
    (xpath('//out:supportingUrl/text()', x, nss))[1]::text AS supportingUrl
FROM unnest(xpath('//out:outcomes/out:dissemination', xmlRecords, nss)) x;

FOR outcome IN
    SELECT * FROM xmlImport
LOOP
    INSERT INTO gtrDisseminations VALUES(
        outcome.uuid::uuid,
        outcome.title,
        outcome.description,
        outcome.form,
        outcome.primaryAudience,
        outcome.yearsOfDissemination,
        outcome.results,
        outcome.impact,
        outcome.typeOfPresentation,
        outcome.geographicReach,
        bool(outcome.partOfOfficialScheme),
        outcome.supportingUrl
    ) ON CONFLICT DO NOTHING;
END LOOP;

-- Collaborations

DROP TABLE IF EXISTS xmlImport;
CREATE TEMP TABLE xmlImport AS
SELECT
    (xpath('//@api:id', x, nss))[1]::text AS uuid,
    (xpath('//out:description/text()', x, nss))[1]::text AS description,
    (xpath('//out:parentOrganisation/text()', x, nss))[1]::text AS parentOrganisation,
    (xpath('//out:childOrganisation/text()', x, nss))[1]::text AS childOrganisation,
    (xpath('//out:principalInvestigatorContribution/text()', x, nss))[1]::text AS
    ↪  principalInvestigatorContribution,
    (xpath('//out:partnerContribution/text()', x, nss))[1]::text AS partnerContribution,
    (xpath('//out:start/text()', x, nss))[1]::text AS startDate,
    (xpath('//out:end/text()', x, nss))[1]::text AS endDate,
    (xpath('//out:sector/text()', x, nss))[1]::text AS sector,
    (xpath('//out:country/text()', x, nss))[1]::text AS country,
    (xpath('//out:impact/text()', x, nss))[1]::text AS impact,
    (xpath('//out:supportingUrl/text()', x, nss))[1]::text AS supportingUrl
FROM unnest(xpath('//out:outcomes/out:collaboration', xmlRecords, nss)) x;

FOR outcome IN
    SELECT * FROM xmlImport
LOOP
    INSERT INTO gtrCollaborations VALUES(
        outcome.uuid::uuid,
        outcome.description,
        outcome.parentOrganisation,
        outcome.childOrganisation,
        outcome.principalInvestigatorContribution,
        outcome.partnerContribution,
```

```
        date(outcome.startDate),
        date(outcome.endDate),
        outcome.sector,
        outcome.country,
        outcome.impact,
        outcome.supportingUrl
    ) ON CONFLICT DO NOTHING;
END LOOP;

-- Key findings

DROP TABLE IF EXISTS xmlImport;
CREATE TEMP TABLE xmlImport AS
SELECT
    (xpath('//@api:id', x, nss))[1]::text AS uuid,
    (xpath('//out:description/text()', x, nss))[1]::text AS description,
    (xpath('//out:nonAcademicUses/text()', x, nss))[1]::text AS nonAcademicUses,
    (xpath('//out:exploitationPathways/text()', x, nss))[1]::text AS exploitationPathways,
    (xpath('//out:sectors/text()', x, nss))[1]::text AS sectors,
    (xpath('//out:supportingUrl/text()', x, nss))[1]::text AS supportingUrl
FROM unnest(xpath('//out:outcomes/out:keyFinding', xmlRecords, nss)) x;

FOR outcome IN
    SELECT * FROM xmlImport
LOOP
    INSERT INTO gtrKeyFindings VALUES(
        uuid(outcome.uuid),
        outcome.description,
        outcome.nonAcademicUses,
        outcome.exploitationPathways,
        outcome.sectors,
        outcome.supportingUrl
    ) ON CONFLICT DO NOTHING;
END LOOP;

-- Further fundings

DROP TABLE IF EXISTS xmlImport;
CREATE TEMP TABLE xmlImport AS
SELECT
    (xpath('//@api:id', x, nss))[1]::text AS uuid,
    (xpath('//out:title/text()', x, nss))[1]::text AS title,
    (xpath('//out:description/text()', x, nss))[1]::text AS description,
    (xpath('//out:narrative/text()', x, nss))[1]::text AS narrative,
    (xpath('//out:amount/@api:amount', x, nss))[1]::text AS amount,
    (xpath('//out:organisation/text()', x, nss))[1]::text AS organisation,
    (xpath('//out:department/text()', x, nss))[1]::text AS department,
    (xpath('//out:fundingId/text()', x, nss))[1]::text AS fundingId,
    (xpath('//out:start/text()', x, nss))[1]::text AS startDate,
    (xpath('//out:end/text()', x, nss))[1]::text AS endDate,
    (xpath('//out:sector/text()', x, nss))[1]::text AS sector,
    (xpath('//out:country/text()', x, nss))[1]::text AS country
FROM unnest(xpath('//out:outcomes/out:furtherFunding', xmlRecords, nss)) x;

FOR outcome IN
    SELECT * FROM xmlImport
LOOP
    INSERT INTO gtrFurtherFundings VALUES(
        uuid(outcome.uuid),
        outcome.title,
```

```
        outcome.description,
        outcome.narrative,
        money(outcome.amount),
        outcome.organisation,
        outcome.department,
        outcome.fundingId,
        date(outcome.startDate),
        date(outcome.endDate),
        outcome.sector,
        outcome.country
    ) ON CONFLICT DO NOTHING;
END LOOP;

-- Impact summaries

DROP TABLE IF EXISTS xmlImport;
CREATE TEMP TABLE xmlImport AS
SELECT
    (xpath('//@api:id', x, nss))[1]::text AS uuid,
    (xpath('//out:title/text()', x, nss))[1]::text AS title,
    (xpath('//out:description/text()', x, nss))[1]::text AS description,
    (xpath('//out:impactTypes/text()', x, nss))[1]::text AS impactTypes,
    (xpath('//out:summary/text()', x, nss))[1]::text AS summary,
    (xpath('//out:beneficiaries/text()', x, nss))[1]::text AS beneficiaries,
    (xpath('//out:contributionMethod/text()', x, nss))[1]::text AS contributionMethod,
    (xpath('//out:sector/text()', x, nss))[1]::text AS sector,
    (xpath('//out:firstYearOfImpact/text()', x, nss))[1]::text AS firstYearOfImpact
FROM unnest(xpath('//out:outcomes/out:impactSummary', xmlRecords, nss)) x;

FOR outcome IN
    SELECT * FROM xmlImport
LOOP
    INSERT INTO gtrImpactSummaries VALUES(
        outcome.uuid::uuid,
        outcome.title,
        outcome.description,
        outcome.impactTypes,
        outcome.summary,
        outcome.beneficiaries,
        outcome.contributionMethod,
        outcome.sector,
        outcome.firstYearOfImpact::int
    ) ON CONFLICT DO NOTHING;
END LOOP;

-- Policy influences

DROP TABLE IF EXISTS xmlImport;
CREATE TEMP TABLE xmlImport AS
SELECT
    (xpath('//@api:id', x, nss))[1]::text AS uuid,
    (xpath('//out:influence/text()', x, nss))[1]::text AS influence,
    (xpath('//out:type/text()', x, nss))[1]::text AS type,
    (xpath('//out:guidelineTitle/text()', x, nss))[1]::text AS guidelineTitle,
    (xpath('//out:impact/text()', x, nss))[1]::text AS impact,
    (xpath('//out:methods/text()', x, nss))[1]::text AS methods,
    (xpath('//out:areas/text()', x, nss))[1]::text AS areas,
    (xpath('//out:geographicReach/text()', x, nss))[1]::text AS geographicReach,
    (xpath('//out:supportingUrl/text()', x, nss))[1]::text AS supportingUrl
FROM unnest(xpath('//out:outcomes/out:policyInfluence', xmlRecords, nss)) x;
```

```

FOR outcome IN
    SELECT * FROM xmlImport
LOOP
    INSERT INTO gtrPolicyInfluences VALUES(
        outcome.uuid::uuid,
        outcome.influence,
        outcome.type,
        outcome.guidelineTitle,
        outcome.impact,
        outcome.methods,
        outcome.areas,
        outcome.geographicReach,
        outcome.supportingUrl
    ) ON CONFLICT DO NOTHING;
END LOOP;

-- Research materials

DROP TABLE IF EXISTS xmlImport;
CREATE TEMP TABLE xmlImport AS
SELECT
    (xpath('//@api:id', x, nss))[1]::text AS uuid,
    (xpath('//out:title/text()', x, nss))[1]::text AS title,
    (xpath('//out:description/text()', x, nss))[1]::text AS description,
    (xpath('//out:type/text()', x, nss))[1]::text AS type,
    (xpath('//out:impact/text()', x, nss))[1]::text AS impact,
    (xpath('//out:softwareDeveloped/text()', x, nss))[1]::text AS softwareDeveloped,
    (xpath('//out:softwareOpenSourced/text()', x, nss))[1]::text AS softwareOpenSourced,
    (xpath('//out:providedToOthers/text()', x, nss))[1]::text AS providedToOthers,
    (xpath('//out:yearFirstProvided/text()', x, nss))[1]::text AS yearFirstProvided,
    (xpath('//out:supportingUrl/text()', x, nss))[1]::text AS supportingUrl
FROM unnest(xpath('//out:outcomes/out:researchMaterials', xmlRecords, nss)) x;

FOR outcome IN
    SELECT * FROM xmlImport
LOOP
    INSERT INTO gtrResearchMaterials VALUES(
        outcome.uuid::uuid,
        outcome.title,
        outcome.description,
        outcome.type,
        outcome.impact,
        bool(outcome.softwareDeveloped),
        bool(outcome.softwareOpenSourced),
        bool(outcome.providedToOthers),
        outcome.yearFirstProvided::int,
        outcome.supportingUrl
    ) ON CONFLICT DO NOTHING;
END LOOP;
END$$; -- BEGIN

DROP TABLE IF EXISTS xmlImport;
```

10.9 similarGtrOrgs.sql

```
CREATE OR REPLACE FUNCTION REPLACE_MANY(txt text, replaces text[][2])
RETURNS text AS $$
DECLARE i int;
```

```
BEGIN
    FOR i IN 1 .. array_upper(replaces, 1) LOOP
        txt := REGEXP_REPLACE(txt, replaces[i][1], replaces[i][2], 'gi');
    END LOOP;
    RETURN txt;
END;
$$ LANGUAGE plpgsql IMMUTABLE;

DO $$
DECLARE
    name_replaces text[][] := array[
        array['\yltd(\.|\y)',  'Limited'],
        array['\ycorp(\.|\y)', 'Corporation'],
        array['\yco(\.|\y)',   'Company'],
        array['\yuni(\.|\y)',  'University'],
        array['&',             'and'],
        array['The',           '']
    ];
BEGIN
    SET pg_trgm.similarity_threshold = 0.9;

    INSERT INTO similarGtrOrgs(
        orgUuid1,
        orgUuid2,
        simTrigramName,
        simTrigramAddress
    )
    SELECT
        uuid1,
        uuid2,
        similarity(name1, name2) AS simName,
        similarity(addr1, addr2) AS simAddress
    FROM
        (SELECT
            o1.orgUuid AS uuid1,
            o2.orgUuid AS uuid2,
            REPLACE_MANY(o1.name, name_replaces) AS name1,
            REPLACE_MANY(o2.name, name_replaces) AS name2,
            CONCAT(o1.address1, '\n',
                   o1.address2, '\n',
                   o1.address3, '\n',
                   o1.address3, '\n',
                   o1.address5, '\n',
                   o1.postCode, '\n',
                   o1.city) AS addr1,
            CONCAT(o2.address1, '\n',
                   o2.address2, '\n',
                   o2.address3, '\n',
                   o2.address3, '\n',
                   o2.address5, '\n',
                   o2.postCode, '\n',
                   o2.city) AS addr2
            FROM gtrOrgs o1
            JOIN gtrOrgs o2
                ON  o1.orgUuid < o2.orgUuid
                AND REPLACE_MANY(o1.name, name_replaces)
                  % REPLACE_MANY(o2.name, name_replaces)
            WHERE NOT EXISTS(
                SELECT
                    *
```

```sql
                FROM
                    junkGtrOrgs
                WHERE
                    orgUuid IN (o1.orgUuid, o2.orgUuid)
            )
        ) q
    ON CONFLICT (orgUuid1, orgUuid2) DO UPDATE SET (
        simTrigramName,
        simTrigramAddress
    ) = (
        excluded.simTrigramName,
        excluded.simTrigramAddress
    );
END $$; -- BEGIN
```

10.10 similarGtrPeople.sql

```sql
DO $$
DECLARE
    rec record;
BEGIN
    SET pg_trgm.similarity_threshold = 0.5;

    INSERT INTO similarGtrPeople(
        personUuid1,
        personUuid2,
        simTrigram)
    SELECT
        uuid1,
        uuid2,
        GREATEST((simFirst + simSur) / 2, (simFirst + simSur + simOther) / 3)
    FROM (SELECT
            p1.personUuid                              AS uuid1,
            p2.personUuid                              AS uuid2,
            similarity(p1.firstName,  p2.firstName)    AS simFirst,
            similarity(p1.surname,    p2.surname)      AS simSur,
            similarity(p1.otherNames, p2.otherNames)   AS simOther
        FROM gtrPeople p1
        INNER JOIN gtrPeople p2
            ON  p1.personUuid <  p2.personUuid
            AND p1.firstName  %  p2.firstName
            AND p1.surname    %  p2.surname
    ) q
    ON CONFLICT (personUuid1, personUuid2) DO UPDATE SET
        simTrigram = excluded.simTrigram;
END $$; -- BEGIN
```

10.11 stats.sql

```sql
-- Perform and export statistical analyses on the dataset
-- Execute using:
-- $ psql -U <username> -d <database> -f setup.sql

-- Utility types
DO $$ BEGIN
    CREATE TYPE PercentileInt AS (
        percentile int,
```

```
        academic bigint,
        medical bigint,
        public bigint,
        private bigint,
        unknown bigint
    );
EXCEPTION
    WHEN duplicate_object THEN null;
END $$; -- BEGIN

CREATE TEMP TABLE IF NOT EXISTS orgTypeCounts AS
SELECT
    type AS type,
    COUNT(*) AS count
FROM
    orgs
GROUP BY type;

\copy (SELECT * FROM orgTypeCounts) TO data/orgTypeCounts.csv (FORMAT CSV, HEADER)

-- Similarity distributions of organisations
CREATE OR REPLACE
FUNCTION OrgSimDist(
    lowerBound int default 50,
    step int default 2
)
RETURNS TABLE("lowerBound" int, "count" bigint) AS $$
DECLARE
BEGIN
    RETURN QUERY
    SELECT
        *,
        (SELECT
            COUNT(*)
        FROM similarGtrOrgs
        WHERE simTrigramName >= (q.bound::float/100))
    FROM (
        SELECT * FROM GENERATE_SERIES(lowerBound, 100, step) AS bound
    ) q;
END; $$ -- FUNCTION
LANGUAGE plpgsql;

\copy (SELECT * FROM OrgSimDist()) TO data/orgSimDist.csv (FORMAT CSV, HEADER)

-- Similarity distributions of organisations with the same postcode
CREATE OR REPLACE
FUNCTION OrgSimDistPostcode(
    lowerBound int default 50,
    step int default 2
)
RETURNS TABLE("lowerBound" int, "count" bigint) AS $$
DECLARE
BEGIN
    RETURN QUERY
    SELECT
        *,
        (SELECT
            COUNT(*)
        FROM similarGtrOrgs s
            INNER JOIN gtrOrgs o1
```

```
                ON s.orgUuid1 = o1.orgUuid
            INNER JOIN gtrOrgs o2
                ON s.orgUuid2 = o2.orgUuid
        WHERE
                simTrigramName >= (q.bound::float/100)
            AND o1.postcode IS NOT NULL
            AND STRIP(o1.postcode) = STRIP(o2.postcode))
    FROM (
        SELECT * FROM generate_series(lowerBound, 100, step) AS bound
    ) q;
END; $$ -- FUNCTION
LANGUAGE plpgsql;

\copy (SELECT * FROM OrgSimDistPostcode()) TO data/orgSimDistPostcode.csv (FORMAT CSV, HEADER)

CREATE OR REPLACE
FUNCTION ProjectFundingPercentiles(
    lowerBound int DEFAULT 1,
    upperBound int DEFAULT 99
)
RETURNS TABLE("percentile" int, "value" numeric) AS $$
DECLARE
    fractions numeric[] := PERCENTILE_FRACTIONS(lowerBound, upperBound);
BEGIN
    RETURN QUERY
    SELECT
        ROUND(fraction * 100)::int,
        ROUND(result::numeric, 2)
    FROM (
        SELECT
            unnest(fractions) AS fraction,
            unnest(
                (SELECT PERCENTILE_CONT(fractions) WITHIN GROUP (ORDER BY sum)
                FROM (
                    SELECT
                        SUM(COALESCE(offer::numeric, 0))
                    FROM gtrProjectOrgs po
                    WHERE offer > 0::money
                    GROUP BY po.projectUuid
                ) q)
            ) AS result
    ) q;
END; $$ -- FUNCTION
LANGUAGE plpgsql;

\copy (SELECT * FROM ProjectFundingPercentiles()) TO data/projectFundingPercentiles.csv (FORMAT CSV, HEADER)

-- Percentiles for total number of projects organisations are involved in
CREATE OR REPLACE
FUNCTION OrgProjectPercentiles(
    lowerBound int DEFAULT 1,
    upperBound int DEFAULT 99
)
RETURNS TABLE("percentile" int, "value" double precision) AS $$
DECLARE
    fractions numeric[] := PERCENTILE_FRACTIONS(lowerBound, upperBound);
BEGIN
    RETURN QUERY
```

```
    SELECT
        ROUND(fraction * 100)::int,
        result
    FROM (
        SELECT
            unnest(fractions) AS fraction,
            unnest(
                (SELECT PERCENTILE_CONT(fractions) WITHIN GROUP (ORDER BY count)
                FROM (
                    SELECT
                        COUNT(*)
                    FROM gtrProjectOrgs po
                    LEFT OUTER JOIN duplicateGtrOrgs d
                        ON d.duplicateUuid = po.orgUuid
                    GROUP BY COALESCE(d.orgUuid, po.orgUuid)
                ) q)
            ) AS result
    ) q;
END; $$ -- FUNCTION
LANGUAGE plpgsql;

\copy (SELECT * FROM OrgProjectPercentiles()) TO data/orgProjectPercentiles.csv (FORMAT
↪  CSV, HEADER)

-- Organisations' percentiles in total projects involved in
CREATE TEMP TABLE IF NOT EXISTS orgProjectPercentiles AS
SELECT
    q.orgUuid,
    q.duplicateUuid,
    PERCENT_RANK() OVER (ORDER BY count) AS rank
FROM (
    SELECT
        COALESCE(d.orgUuid, po.orgUuid) AS orgUuid,
        po.orgUuid AS duplicateUuid,
        COUNT(*)
    FROM
        gtrProjectOrgs po
        LEFT OUTER JOIN duplicateGtrOrgs d
            ON d.duplicateUuid = po.orgUuid
    GROUP BY COALESCE(d.orgUuid, po.orgUuid), po.orgUuid
) q;

-- Percentile distributions for organisations, by number of projects involved
-- with
CREATE OR REPLACE
FUNCTION OrgProjectPercentileDist(
    lowerBound int DEFAULT 0,
    upperBound int DEFAULT 100
)
RETURNS SETOF PercentileInt AS $$
DECLARE
    fractions numeric[] := PERCENTILE_FRACTIONS(lowerBound, upperBound);
    fraction numeric;
BEGIN
    DROP TABLE IF EXISTS res;
    CREATE TEMP TABLE IF NOT EXISTS res OF PercentileInt;

    FOREACH fraction in ARRAY fractions LOOP
        INSERT INTO res
        SELECT
```

```
            COALESCE(ct.fraction, 0) AS fraction,
            COALESCE(ct.academic, 0) AS academic,
            COALESCE(ct.medical,  0) AS medical,
            COALESCE(ct.public,   0) AS public,
            COALESCE(ct.private,  0) AS private,
            COALESCE(ct.unknown,  0) AS unknown
        FROM crosstab('
            SELECT
                ROUND('||fraction||' * 100)::int,
                type,
                COUNT(*)
            FROM (
                SELECT
                    COALESCE(type, ''Unknown'') AS type,
                    COUNT(*)
                FROM
                    orgProjectPercentiles opp
                    INNER JOIN orgs o
                        ON o.gtrOrgUuid = opp.orgUuid
                WHERE
                    rank <= '||fraction||'
                GROUP BY orgUuid, type
                ORDER BY 1, 2
            ) q
            WHERE count > 0
            GROUP BY type
            ',
            'VALUES
                (''Academic''::orgType),
                (''Medical''),
                (''Public''),
                (''Private''),
                (''Unknown'')'
        ) AS ct (
            fraction numeric,
            academic bigint,
            medical bigint,
            public bigint,
            private bigint,
            unknown bigint
        );
    END LOOP;

    RETURN QUERY SELECT * FROM res;
END; $$ -- FUNCTION
LANGUAGE plpgsql;

\copy (SELECT * FROM OrgProjectPercentileDist()) TO data/orgProjectPercentileDist.csv
↪  (FORMAT CSV, HEADER)

-- Percentage of organisations who collaborated with at least one other
-- organisation within project count percentile ranges
CREATE OR REPLACE
FUNCTION OrgProjectPercentileCollab(
    lowerBound int DEFAULT 0,
    upperBound int DEFAULT 100
)
RETURNS SETOF PercentileInt AS $$
DECLARE
    fractions numeric[] := PERCENTILE_FRACTIONS(lowerBound, upperBound);
```

```
    fraction numeric;
BEGIN
    CREATE TEMP TABLE IF NOT EXISTS pairedOrgProjects AS
    SELECT
        COALESCE(d1.orgUuid, o1.orgUuid) AS orgUuid1,
        type,
        o2.orgUuid AS orgUuid2,
        rank
    FROM
        -- First organisation
        gtrOrgs o1
        LEFT OUTER JOIN duplicateGtrOrgs d1
            ON d1.duplicateUuid = o1.orgUuid
        INNER JOIN orgs oo1
            ON oo1.gtrOrgUuid = COALESCE(d1.orgUuid, o1.orgUuid)
        -- Second organisation
        INNER JOIN orgProjectPercentiles o2
            ON o2.orgUuid <> o1.orgUuid
        -- Projects both organisations are involved in
        INNER JOIN gtrProjectOrgs po1
            ON po1.orgUuid IN (d1.orgUuid, o1.orgUuid)
        INNER JOIN gtrProjectOrgs po2
            ON  po2.projectUuid = po1.projectUuid
            AND po2.orgUuid = o2.orgUuid;

    DROP TABLE IF EXISTS res;
    CREATE TEMP TABLE IF NOT EXISTS res OF PercentileInt;

    FOREACH fraction in ARRAY fractions LOOP
        INSERT INTO res
        SELECT
            COALESCE(ct.fraction, 0) AS fraction,
            COALESCE(ct.academic, 0) AS academic,
            COALESCE(ct.medical,  0) AS medical,
            COALESCE(ct.public,   0) AS public,
            COALESCE(ct.private,  0) AS private,
            COALESCE(ct.unknown,  0) AS unknown
        FROM crosstab('
            SELECT
                ROUND('||fraction||' * 100)::int,
                type,
                COUNT(*)
            FROM (
                SELECT
                    COALESCE(type, ''Unknown'') AS type,
                    COUNT(*)
                FROM
                    pairedOrgProjects
                WHERE
                    rank <= '||fraction||'
                GROUP BY orgUuid1, type
                ORDER BY 1, 2
            ) q
            WHERE count > 0
            GROUP BY type
            ',
            'VALUES
                (''Academic''::orgType),
                (''Medical''),
                (''Public''),
```

```
                ('''Private''),
                ('''Unknown''')'
        ) AS ct (
            fraction numeric,
            academic bigint,
            medical bigint,
            public bigint,
            private bigint,
            unknown bigint
        );
    END LOOP;

    RETURN QUERY SELECT * FROM res;
END; $$ -- FUNCTION
LANGUAGE plpgsql;

\copy (SELECT * FROM OrgProjectPercentileCollab()) TO data/orgProjectPercentileCollab.csv (FORMAT CSV, HEADER)

-- Percentiles for funding received by organisations
CREATE OR REPLACE
FUNCTION OrgFundingPercentiles(
    lowerBound int DEFAULT 1,
    upperBound int DEFAULT 99
)
RETURNS TABLE("percentile" int, "value" numeric) AS $$
DECLARE
    fractions numeric[] := PERCENTILE_FRACTIONS(lowerBound, upperBound);
BEGIN
    RETURN QUERY
    SELECT
        ROUND(fraction * 100)::int,
        ROUND(result::numeric, 2)
    FROM (
        SELECT
            unnest(fractions) AS fraction,
            unnest(
                (SELECT PERCENTILE_CONT(fractions) WITHIN GROUP (ORDER BY sum)
                FROM (
                    SELECT
                        SUM(COALESCE(offer::numeric, 0))
                    FROM gtrProjectOrgs po
                    LEFT OUTER JOIN duplicateGtrOrgs d
                        ON d.duplicateUuid = po.orgUuid
                    WHERE offer > 0::money
                    GROUP BY COALESCE(d.orgUuid, po.orgUuid)
                ) q)
            ) AS result
    ) q;
END; $$ -- FUNCTION
LANGUAGE plpgsql;

\copy (SELECT * FROM OrgFundingPercentiles()) TO data/orgFundingPercentiles.csv (FORMAT CSV, HEADER)

-- Organisations' percentiles in total funding received
CREATE TEMP TABLE IF NOT EXISTS orgFundingPercentiles AS
SELECT
    q.orgUuid,
    q.duplicateUuid,
```

```
    PERCENT_RANK() OVER (ORDER BY sum) AS rank
FROM (
    SELECT
        COALESCE(d.orgUuid, po.orgUuid) AS orgUuid,
        po.orgUuid AS duplicateUuid,
        SUM(COALESCE(offer::numeric, 0))
    FROM
        gtrProjectOrgs po
        LEFT OUTER JOIN duplicateGtrOrgs d
            ON d.duplicateUuid = po.orgUuid
    WHERE offer > 0::money
    GROUP BY COALESCE(d.orgUuid, po.orgUuid), po.orgUuid
) q;

-- Number of organisations who collaborated with at least one other
-- organisation within funding percentile ranges
CREATE OR REPLACE
FUNCTION OrgFundingPercentileCollab(
    lowerBound int DEFAULT 0,
    upperBound int DEFAULT 100
)
RETURNS SETOF PercentileInt AS $$
DECLARE
    fractions numeric[] := PERCENTILE_FRACTIONS(lowerBound, upperBound);
    fraction numeric;
BEGIN
    CREATE TEMP TABLE IF NOT EXISTS pairedOrgFunding AS
    SELECT
        COALESCE(d1.orgUuid, o1.orgUuid) AS orgUuid1,
        oo1.type AS type,
        o2.orgUuid AS orgUuid2,
        rank
    FROM
        -- First organisation
        gtrOrgs o1
        LEFT OUTER JOIN duplicateGtrOrgs d1
            ON d1.duplicateUuid = o1.orgUuid
        LEFT OUTER JOIN orgs oo1
            ON oo1.gtrOrgUuid IN (d1.orgUuid, o1.orgUuid)
        -- Second organisation
        INNER JOIN orgFundingPercentiles o2
            ON o2.orgUuid <> o1.orgUuid
        -- Projects both organisations are involved in
        INNER JOIN gtrProjectOrgs po1
            ON po1.orgUuid IN (d1.orgUuid, o1.orgUuid)
        INNER JOIN gtrProjectOrgs po2
            ON  po2.projectUuid = po1.projectUuid
            AND po2.orgUuid = o2.orgUuid;

    DROP TABLE IF EXISTS res;
    CREATE TEMP TABLE IF NOT EXISTS res OF PercentileInt;

    FOREACH fraction in ARRAY fractions LOOP
        INSERT INTO res
        SELECT
            COALESCE(ct.fraction, 0) AS fraction,
            COALESCE(ct.academic, 0) AS academic,
            COALESCE(ct.medical,  0) AS medical,
            COALESCE(ct.public,   0) AS public,
            COALESCE(ct.private,  0) AS private,
```

```
            COALESCE(ct.unknown,  0) AS unknown
        FROM crosstab('
            SELECT
                ROUND('||fraction||' * 100)::int,
                type,
                COUNT(*)
            FROM (
                SELECT
                    COALESCE(type, ''Unknown'') AS type,
                    COUNT(*)
                FROM
                    pairedOrgFunding
                WHERE
                    rank <= '||fraction||'
                GROUP BY orgUuid1, type
                ORDER BY 1, 2
            ) q
            WHERE count > 0
            GROUP BY type
            ',
            'VALUES
                (''Academic''::orgType),
                (''Medical''),
                (''Public''),
                (''Private''),
                (''Unknown'')'
        ) AS ct (
            fraction numeric,
            academic bigint,
            medical bigint,
            public bigint,
            private bigint,
            unknown bigint
        );
    END LOOP;

    RETURN QUERY SELECT * FROM res;
END; $$ -- FUNCTION
LANGUAGE plpgsql;

\copy (SELECT * FROM OrgFundingPercentileCollab()) TO data/orgFundingPercentileCollab.csv (FORMAT CSV, HEADER)

-- Percentile distributions for organisations, by total amount of funding
-- received
CREATE OR REPLACE
FUNCTION OrgFundingPercentileProjects(
    lowerBound int DEFAULT 0,
    upperBound int DEFAULT 100
)
RETURNS TABLE("percentile" int, "value" bigint) AS $$
DECLARE
    fractions numeric[] := PERCENTILE_FRACTIONS(lowerBound, upperBound);
BEGIN
    CREATE TEMP TABLE IF NOT EXISTS projectPairedRanks AS
    SELECT
        projectUuid,
        rank
    FROM
        orgFundingPercentiles o
```

```
        INNER JOIN gtrProjectOrgs po
            ON po.orgUuid IN (o.orgUuid, o.duplicateUuid);

    RETURN QUERY
    SELECT
        ROUND(fractions.upperBound * 100)::int,
        (SELECT
            COUNT(*)
        FROM (
            SELECT
                COUNT(*)
            FROM
                projectPairedRanks
            WHERE
                rank <= fractions.upperBound
            GROUP BY projectUuid
        ) q
        WHERE count > 0
        ) AS result
    FROM unnest(fractions) AS fractions(upperBound);
END; $$ -- FUNCTION
LANGUAGE plpgsql;

\copy (SELECT * FROM OrgFundingPercentileProjects()) TO
↪ data/orgFundingPercentileProjects.csv (FORMAT CSV, HEADER)
```

10.12 mergeGtrOrgs.sql

```
BEGIN;

INSERT INTO junkGtrOrgs(
    orgUuid
)
SELECT
    orgUuid
FROM
    gtrOrgs
WHERE name ~* '^unknown*|^unlisted*'
ON CONFLICT DO NOTHING;

INSERT INTO duplicateGtrOrgs(
    orgUuid,
    duplicateUuid
)
SELECT
    LEAST(o1.orgUuid, o2.orgUuid) AS orgUuid,
    GREATEST(o1.orgUuid, o2.orgUuid) AS duplicateUuid
FROM
    similarGtrOrgs s
    INNER JOIN gtrOrgs o1
        ON s.orgUuid1 = o1.orgUuid
    INNER JOIN gtrOrgs o2
        ON s.orgUuid2 = o2.orgUuid
WHERE
        simTrigramName >= 0.9
    AND COALESCE(manualResult, TRUE)
    OR  (simTrigramName >= 0.5
        AND o1.postCode IS NOT NULL
```

```
        AND strip(o1.postcode) = strip(o2.postcode))
    -- Check that the first rec isn't already marked as a duplicate
    AND NOT EXISTS(SELECT *
        FROM duplicateGtrOrgs
        WHERE duplicateUuid IN (o1.orgUuid, o2.orgUuid))
    AND NOT EXISTS(SELECT *
        FROM junkGtrOrgs
        WHERE orgUuid IN (o1.orgUuid, o2.orgUuid))
GROUP BY
    LEAST(   o1.orgUuid, o2.orgUuid),
    GREATEST(o1.orgUuid, o2.orgUuid)
ON CONFLICT (duplicateUuid) DO NOTHING;

INSERT INTO orgs(
    gtrOrgUuid,
    name,
    address1,
    address2,
    address3,
    address4,
    address5,
    postCode,
    city,
    region,
    country
)
SELECT
DISTINCT ON (orgUuid)
    LEAST(o.orgUuid, d.orgUuid, d.duplicateUuid) AS orgUuid,
    MERGE(name)                                  AS name,
    MERGE(address1)                              AS address1,
    MERGE(address2)                              AS address2,
    MERGE(address3)                              AS address3,
    MERGE(address4)                              AS address4,
    MERGE(address5)                              AS address5,
    MERGE(postCode)                              AS postCode,
    MERGE(city)                                  AS city,
    COALESCE(MERGE(NULLIF(region, 'Unknown')),
        'Unknown')                               AS region,
    MERGE(country)                               AS country
FROM
    gtrOrgs o
    LEFT OUTER JOIN duplicateGtrOrgs d
        ON o.orgUuid IN (d.orgUuid, d.duplicateUuid)
WHERE
    NOT EXISTS(SELECT *
        FROM junkGtrOrgs
        WHERE orgUuid IN (o.orgUuid, d.orgUuid, d.duplicateUuid))
GROUP BY
    LEAST(o.orgUuid, d.orgUuid, d.duplicateUuid)
ON CONFLICT (gtrOrgUuid) DO UPDATE SET
    name     = excluded.name,
    address1 = excluded.address1,
    address2 = excluded.address2,
    address3 = excluded.address3,
    address4 = excluded.address4,
    address5 = excluded.address5,
    postCode = excluded.postCode,
    city     = excluded.city,
    region   = excluded.region,
```

```
    country   = excluded.country;

COMMIT;
```

10.13 mergeGtrPeople.sql

```
BEGIN;

INSERT INTO duplicateGtrPeople(
    personUuid,
    duplicateUuid
)
SELECT
    LEAST(s.personUuid1, s.personUuid2) AS personUuid,
    GREATEST(s.personUuid1, s.personUuid2) AS duplicateUuid
FROM
    similarGtrPeople s
    INNER JOIN gtrOrgPeople op1
        ON s.personUuid1 = op1.personUuid
    INNER JOIN gtrOrgPeople op2
        ON  s.personUuid2 = op2.personUuid
    LEFT OUTER JOIN duplicateGtrOrgs d1
        ON d1.duplicateUuid = op1.orgUuid
    LEFT OUTER JOIN duplicateGtrOrgs d2
        ON d2.duplicateUuid = op2.orgUuid
WHERE
        COALESCE(d1.orgUuid, op1.orgUuid) = COALESCE(d2.orgUuid, op2.orgUuid)
    AND simTrigram >= 0.9
    AND COALESCE(s.manualResult, TRUE)
    -- Check that the first rec isn't already marked as a duplicate
    AND NOT EXISTS(SELECT *
        FROM duplicateGtrPeople
        WHERE duplicateUuid IN (s.personUuid1, s.personUuid2))
GROUP BY
    LEAST(   s.personUuid1, s.personUuid2),
    GREATEST(s.personUuid1, s.personUuid2)
ON CONFLICT (duplicateUuid) DO NOTHING;

INSERT INTO people(
    gtrPersonUuid,
    firstName,
    surname,
    otherNames
)
SELECT
    LEAST(p1.personUuid, d.personUuid, d.duplicateUuid) AS personUuid,
    MERGE(firstName)                                    AS firstName,
    MERGE(surname)                                      AS surname,
    MERGE(otherNames)                                   AS otherNames
FROM
    gtrPeople p1
    LEFT OUTER JOIN duplicateGtrPeople d
        ON p1.personUuid IN (d.personUuid, d.duplicateUuid)
GROUP BY
    LEAST(p1.personUuid, d.personUuid, d.duplicateUuid)
ON CONFLICT (gtrPersonUuid) DO UPDATE SET
    firstName  = excluded.firstName,
    surname    = excluded.surname,
    otherNames = excluded.otherNames;
```

```

COMMIT;
```

10.14 eastMidlandsGraphGephi.sql

```
DROP TABLE IF EXISTS nodes;
CREATE TEMP TABLE IF NOT EXISTS nodes AS
SELECT
    gtrOrgUuid AS id,
    name AS label,
    *
FROM
    orgs o
WHERE
    region = 'East Midlands';

\copy (SELECT * FROM nodes) TO data/eastMidlandsGraphGephiNodes.csv (FORMAT CSV, HEADER)

DROP TABLE IF EXISTS edges;
CREATE TEMP TABLE IF NOT EXISTS edges AS
SELECT
    po1.projectUuid AS projectUuid,
    o1.gtrOrgUuid AS source,
    GREATEST(po1.offer::numeric::int, po1.cost::numeric::int, 1) AS weight,

    o2.gtrOrgUuid AS target,
    p.title AS label,
    p.startDate AS startDate,
    p.endDate AS endDate
FROM
    -- First organisation
    nodes o1
    LEFT OUTER JOIN duplicateGtrOrgs d1
        ON d1.orgUuid = o1.gtrOrgUuid
    INNER JOIN gtrProjectOrgs po1
        ON po1.orgUuid IN (o1.gtrOrgUuid, d1.duplicateUuid)

    -- Second organisation
    INNER JOIN gtrProjectOrgs po2
        ON po2.orgUuid NOT IN (o1.gtrOrgUuid, d1.duplicateUuid)
        AND po2.projectUuid = po1.projectUuid
    LEFT OUTER JOIN duplicateGtrOrgs d2
        ON po2.orgUuid IN (d2.orgUuid, d2.duplicateUuid)
    INNER JOIN nodes o2
        ON o2.gtrOrgUuid IN (d2.orgUuid, po2.orgUuid)

    -- Shared project
    INNER JOIN gtrProjects p
        ON p.projectUuid = po1.projectUuid;

\copy (SELECT * FROM edges) TO data/eastMidlandsGraphGephiEdges.csv (FORMAT CSV, HEADER)
```

10.15 eastMidlandsGraphNodeXL.sql

```
DROP TABLE IF EXISTS edges;
CREATE TEMP TABLE edges AS
SELECT
```

```
    o1.gtrOrgUuid AS source,
    o1.name AS sourceName,
    o1.region AS sourceRegion,
    o1.city AS sourceCity,
    o1.country AS sourceCountry,
    o1.type AS sourceType,
    po1.role AS sourceRole,
    po1.offer::numeric::int AS offer,
    po1.cost::numeric::int AS cost,

    o2.gtrOrgUuid AS target,
    o2.name AS targetName,
    o2.region AS targetRegion,
    o2.city AS targetCity,
    o2.country AS targetCountry,
    o2.type AS targetType,

    p.projectUuid AS projectUuid,
    p.startDate AS startDate,
    p.endDate AS endDate
FROM
    -- First organisation
    orgs o1
    LEFT OUTER JOIN duplicateGtrOrgs d1
        ON d1.orgUuid = o1.gtrOrgUuid
    INNER JOIN gtrProjectOrgs po1
        ON po1.orgUuid IN (o1.gtrOrgUuid, d1.duplicateUuid)

    -- Second organisation
    INNER JOIN gtrProjectOrgs po2
        ON po2.orgUuid NOT IN (o1.gtrOrgUuid, d1.duplicateUuid)
        AND po2.projectUuid = po1.projectUuid
    LEFT OUTER JOIN duplicateGtrOrgs d2
        ON po2.orgUuid IN (d2.orgUuid, d2.duplicateUuid)
    INNER JOIN orgs o2
        ON o2.gtrOrgUuid IN (d2.orgUuid, po2.orgUuid)

    -- Shared project
    INNER JOIN gtrProjects p
        ON p.projectUuid = po1.projectUuid
WHERE
    p.startDate >= '1/1/2010'
    AND (p.endDate IS NULL
    OR p.endDate > '1/1/2015')
    AND o1.region = 'East Midlands'
    AND o2.region = 'East Midlands';

\copy (SELECT * FROM edges) TO data/eastMidlandsGraphNodeXL.csv (FORMAT CSV, HEADER)
```

10.16 Database schema manual

An technical description of the contents of the database schema, visualised in figure 16.

10.16.1 Organisations

10.16.1.1 gtrOrgs Organisations stored in the Gateway to Research (GtR) database. 47822 records

Field	Type	Description
orgUuid	UUID	Unique identifier
name	text	Name of organisation
address1	text	Address line 1
address2	text	Address line 2
address3	text	Address line 3
address4	text	Address line 4
address5	text	Address line 5
postCode	text	Post code
city	text	City organisation is based in
region	gtrRegion	Region organisation is based in
country	text	Country organisation is based in
recorded	date	Date this record was created (in the GtR database)

10.16.1.2 gtrRegion: The regions an organisation can be based in:

- `Channel Islands/Isle of Man`
- `East Midlands`
- `East of England`
- `London`
- `North East`
- `Northern Ireland`
- `North West`
- `Scotland`
- `South East`
- `South West`
- `Wales`
- `West Midlands`
- `Yorkshire and The Humber`
- `Outside UK`
- `Unknown`

10.16.2 Projects

10.16.2.1 gtrProjects Projects stored in the Gateway to Research (GtR) database. 97277 records

Field	Type	Description
projectUuid	uuid	Unique identifier
title	text	Title of project
status	gtrProjectStatus	Current status of project: 'Active' or 'Closed'
category	gtrGrantCategory	Category of project

Field	Type	Description
leadFunder	gtrFunder	The main UKRI (sub)organisation providing the majority of the funding for this project
abstract	text	Abstract text summarising the project's background and goals
techAbstract	text	Technical summary of research being undertaken
potentialImpact	text	Planned impact that will result from the project
startDate	date	Date of the beginning of project funding
endDate	date	Date of the end of project funding
recorded	date	Date this record was created (in the GtR database)

10.16.2.2 gtrGrantCategory Categories that projects can be contained within

- `BIS-Funded Programmes`
- `Centres`
- `Collaborative R&D`
- `CR&D Bilateral`
- `EU-Funded`
- `European Enterprise Network`
- `Fast Track`
- `Feasibility Studies`
- `Fellowship`
- `GRD Development of Prototype`
- `GRD Proof of Concept`
- `GRD Proof of Market`
- `Intramural`
- `Knowledge Transfer Network`
- `Knowledge Transfer Partnership`
- `Large Project`
- `Launchpad`
- `Legacy Department of Trade & Industry`
- `Legacy RDA Collaborative R&D`
- `Legacy RDA Grant for R&D`
- `Missions`
- `Other Grant`
- `Procurement`
- `Research Grant`
- `Small Business Research Initiative`
- `SME Support`
- `Special Interest Group`
- `Studentship`
- `Study`
- `Third Party Grant`
- `Training Grant`
- `Unknown`
- `Vouchers`

10.16.2.3 gtrFunder UKRI, or sub-organisations of UKRI, that provide funding

- `UKRI`
- `AHRC`

- BBSRC
- EPSRC
- ESRC
- Innovate UK
- MRC
- NC3Rs
- NERC
- STFC

10.16.2.4 gtrSubjects Research subjects associated with projects.
83 records

Field	Type	Description
subjectUuid	uuid	Unique identifier
name	text	Name of subject, as specified by users

10.16.2.5 gtrProjectSubjects Subjects that particular projects are associated with. Percentages are specified by users, each project can be associated with multiple research subjects.
78270 records

Field	Type	Description
subjectUuid	uuid	Unique identifier of subject in gtrSubjects
projectUuid	uuid	Unique identifier of project in gtrProjects
percent	numeric	Percentage of project that is related to this subject

10.16.2.6 gtrTopics Research topics associated with projects.
610 records

Field	Type	Description
topicUuid	uuid	Unique identifier
name	text	Name of topic, as specified by users

10.16.2.7 gtrProjectTopics Topics that particular projects are associated with. Percentages are specified by users, each project can be associated with multiple research Topics.
150016 records

Field	Type	Description
topicUuid	uuid	Unique identifier of topic in gtrTopics
projectUuid	uuid	Unique identifier of project in gtrProjects
percent	numeric	Percentage of project that is related to this topic

10.16.3 Project outcomes

The outcomess of a project as logged by users

10.16.3.1 gtrDisseminations

Disseminations published as a result of a project. 58 records.

Field	Type	Description
disUuid	uuid	Unique identifier
title	text	Title
description	text	Description of the contents
form	text	The form of publishing, e.g. presentations, conferences, newsletter
primaryAudience	text	Audience this dissemination is targeted at, e.g. policymakers, students
yearsOfDissemination	text	Comma-separated list of years it was active
results	text	Description of the results of publishing (unused)
impact	text	Descriptions of the outcomes of publishing
typeOfPresentation	text	Type of presentation (mostly unused)
geographicReach	text	Local, Regional, National, or International
partOfOfficialScheme	boolean	Whether engagement activity is part of an official scheme
supportingUrl	text	URLs for accessing supporting material

10.16.3.2 gtrCollaborations

Further collaborations that formed as a result of a project. 8 records

Field	Type	Description
collabUuid	uuid	Unique identifier
description	text	Description of collaboration
parentOrganisation	text	Parent organisation of collaboration partner
childOrganisation	text	Child organisation of collaboration partner
principalInvestigatorContribution	text	Benefits provided to new collaborator by principal investigator
partnerContribution	text	Benefits provided to principal investigator by new collaborator
startDate	date	Date collaboration began
endDate	date	Date collaboration ended
sector	gtrSector	Sector of collaborating organisation
country	text	Country collaborator is based in
impact	text	Resulting impact of the collaboration
supportingUrl	text	URLs for accessing supporting material

10.16.3.3 gtrKeyFindings

Key findings from a project. 1 record

Field	Type	Description
keyFindingUuid	uuid	Unique identifier
description	text	Description of findings
nonAcademicUses	text	Non-academic applications of findings
exploitationPathways	text	Possible further research and development
sectors	text	Sectors that findings could be applied to
supportingUrl	text	URLs for accessing supporting material

10.16.3.4 gtrFurtherFundings

Further project funding after the initial funding period expired.
9 records

Field	Type	Description
furtherFundingUuid	uuid	Unique identifier
title	text	Tile of grant/funding received (unused)
description	text	Description of further funding received
narrative	text	Reasoning for further funding (unused)
amount	money	Amount of funding received
organisation	text	Organisation providing funding
department	text	Department of organisation providing funding
fundingId	text	External reference identifier for the funding received
startDate	date	Start of funding period
endDate	date	End of funding period
sector	gtrSector	Sector of organisation providing funding
country	text	Country of organisation providing funding

10.16.3.5 gtrImpactSummaries

Further project funding after the initial funding period expired.
5 records

Field	Type	Description
impactSummaryUuid	uuid	Unique identifier
title	text	Unused
description	text	Description of impact
impactTypes	text	Unused
summary	text	Unused
beneficiaries	text	Unused
contributionMethod	text	Unused
sector	text	Comma-delimited list of areas affected, e.g. ‘Healthcare’, ‘Manufacturing’
firstYearOfImpact	int	Year that the impact took effect

10.16.3.6 gtrPolicyInfluences

Any influence on an organisation/government’s policy as a result of a project.
5 records

Field	Type	Description
policyInfluenceUuid	uuid	Unique identifier
influence	text	Description of influence, e.g. committee influenced
type	text	Form of incluence, e.g. membership of a guideline committee
guidelineTitle	text	Unused
impact	text	Unused
methods	text	Unused
areas	text	Unused
geographicReach	text	Region influenced, e.g. ‘National’, ‘Europe’
supportingUrl	text	URLs for accessing supporting material

10.16.3.7 gtrResearchMaterials

Research material published as a result of a project. 0 records

Field	Type	Description
researchMatUuid	uuid	Unique identifier
title	text	Title
description	text	Description
type	text	Type
impact	text	Impact from releasing
softwareDeveloped	boolean	Whether software was developed
softwareOpenSourced	boolean	Whether software was open-sourced
providedToOthers	boolean	Whether material was provided to other entities
yearFirstProvided	int	First year research material was provided
supportingUrl	text	URLs for accessing supporting material

10.16.4 Links

10.16.4.1 gtrOrgPeople

Organisations that people in `gtrPeople` are linked to. 82015 records

Field	Type	Description
personUuid	uuid	UUID of person in `gtrPeople`
orgUuid	uuid	UUID of organisation in `gtrOrgs`

10.16.4.2 gtrProjectPeople

Individuals working on a particular project. 201596 records

Field	Type	Description
projectUuid	uuid	UUID of project in `gtrProjects`
personUuid	uuid	UUID of person in `gtrPersonRole`
role	gtrPersonRole	Role of individual in the project

10.16.4.3 gtrPersonRole

Possible roles an individual can have in a project:

- `Principal Investigator`
- `Co-Investigator`
- `Project Manager`
- `Fellow`
- `Training Grant Holder`
- `Primary Supervisor`
- `Researcher Co-Investigator`
- `Researcher`

10.16.4.4 gtrProjectOrgs

Organisations working on a particular project. 283527 records

Field	Type	Description
projectUuid	uuid	UUID of project in `gtrProjects`

Field	Type	Description
orgUuid	uuid	UUID of project in `gtrOrgs`
role	gtrOrgRole	Role of organisation in the project
startDate	date	Start of collaboration period
endDate	date	End of collaboration period
cost	money	Total funds spent by this organisation on the project
offer	money	Total funding offered by the funder to this organisations

10.16.4.5 gtrRelatedProjects Projects that are related to one another in some way. 25392 records

Field	Type	Description
projectUuid1	uuid	UUID of project in `gtrProjects`
projectUuid2	uuid	UUID of project in `gtrProjects`
relation	gtrProjectRelation	Relationship between the projects
startDate	date	Start of related (2nd) project
endDate	date	End of related (2nd) project

10.17 Database schema

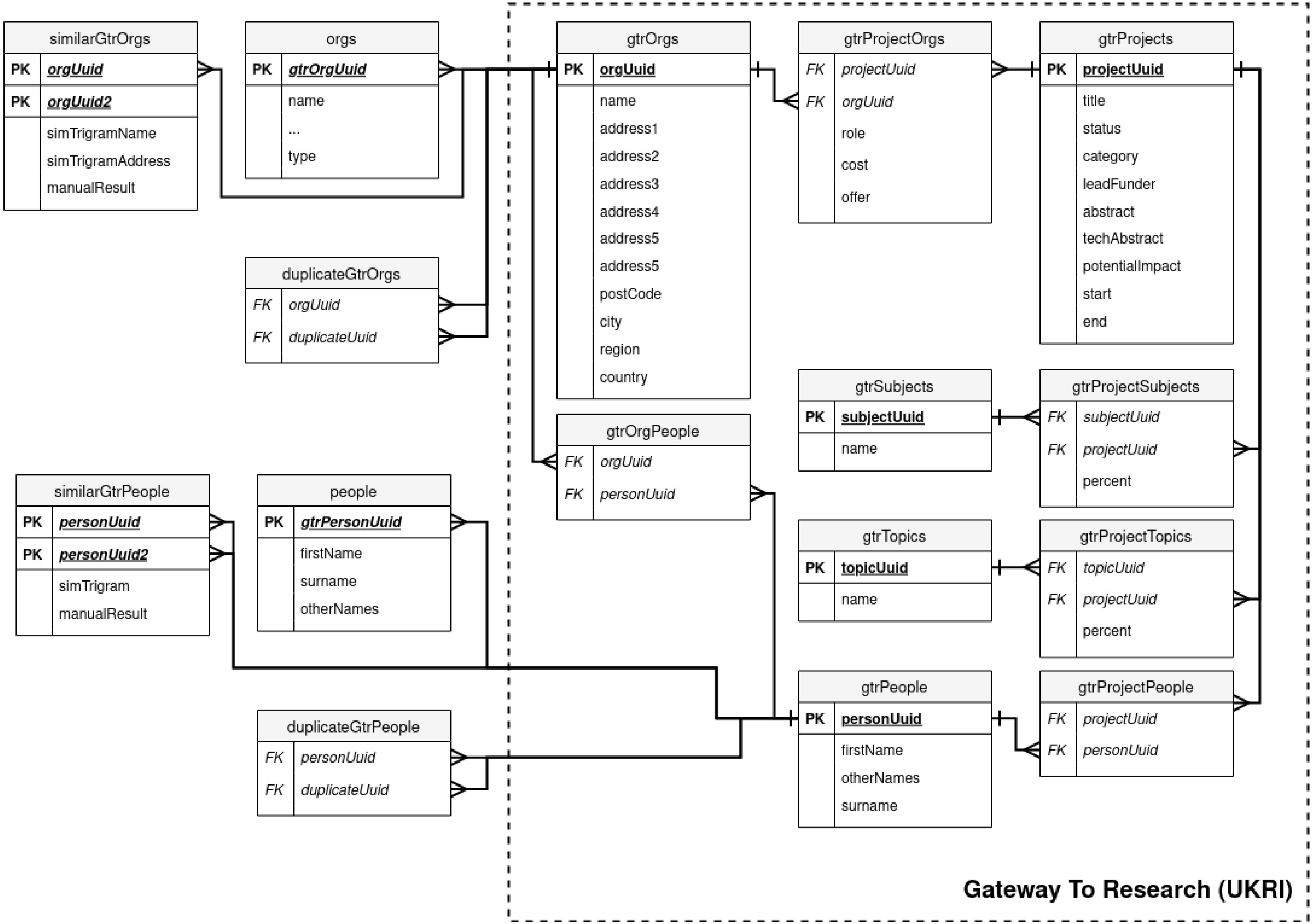

Figure 16: Entity Relationship Diagram describing the database schema

10.18 Database schema (Initial design)

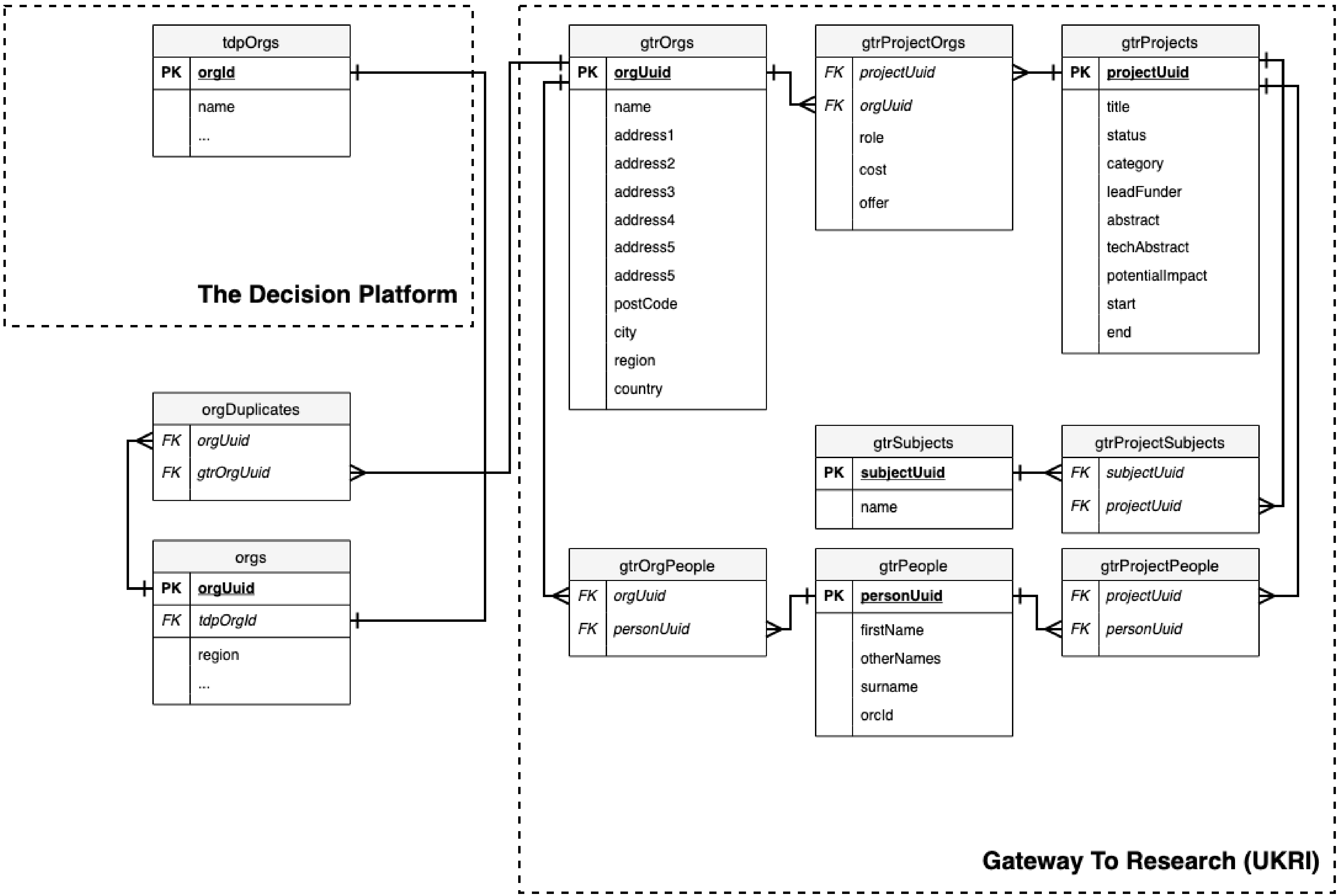

Figure 17: Entity Relationship Diagram describing the planned database schema, before changes were made throughout the implementation of the project

10.19 East Midlands network (2002-2008)

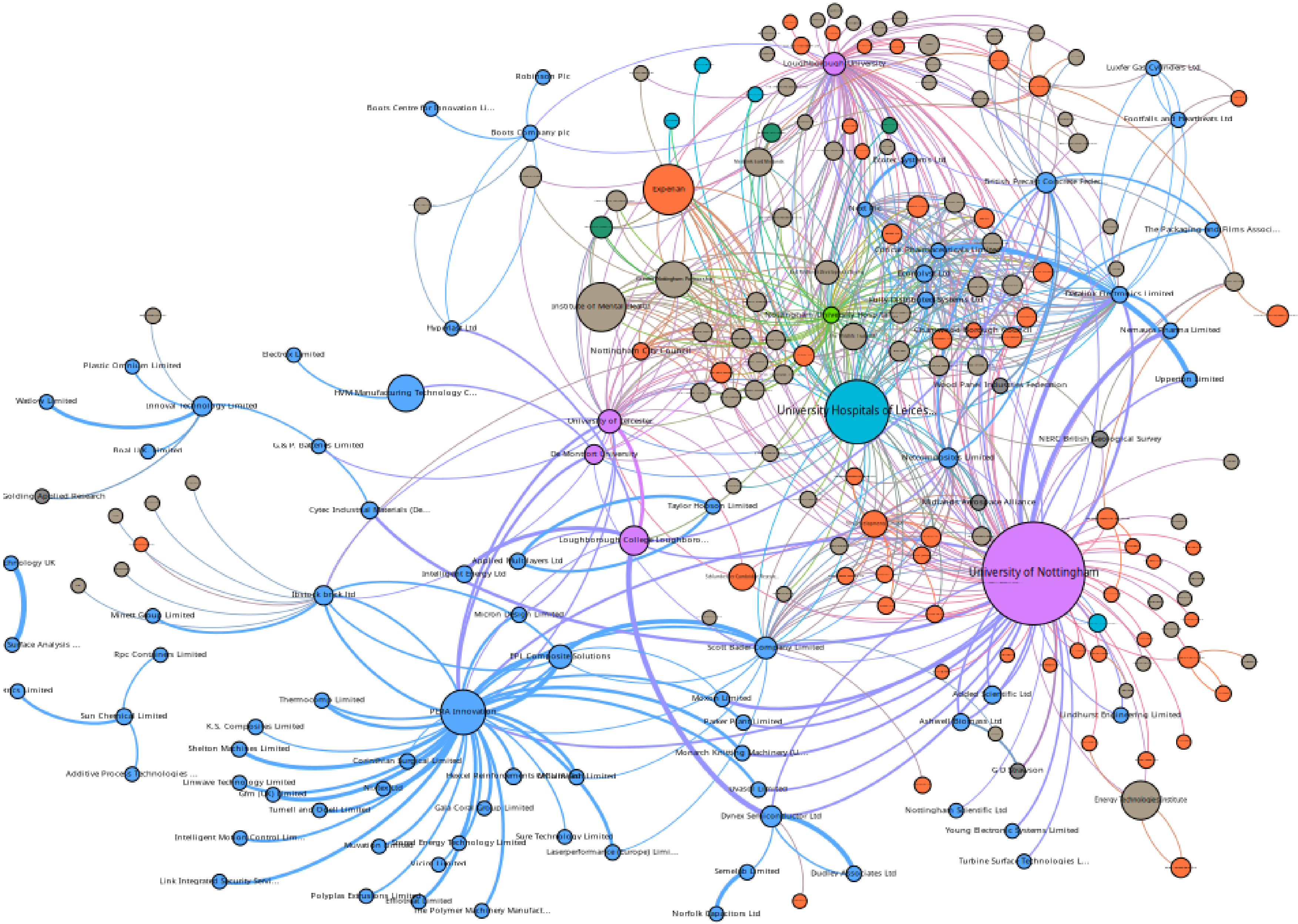

Figure 18: Funding network for organisations in the East Midlands within the years 2002 to 2008

10.20 East Midlands network (2005-2010)

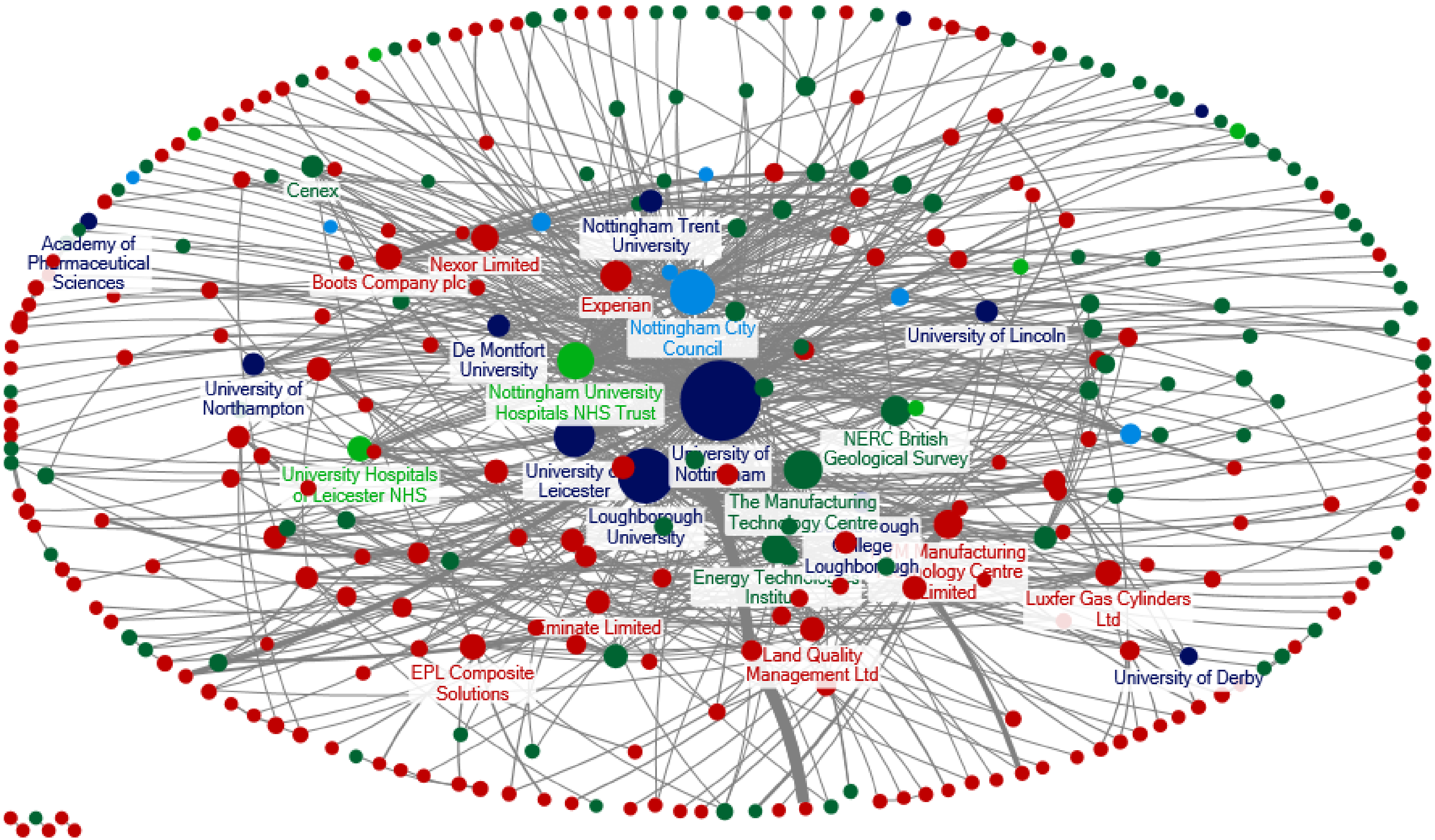

Figure 19: Funding network for organisations in the East Midlands within the years 2005 to 2010