Performance

Table of contents

1 Home **4**

I Using it **5**

2 Getting started **6**
- 2.1 What SPC does? . . . 6
- 2.2 What is the output generated by SPC? . . . 6
- 2.3 Areas ready to use and explore . . . 6

3 Installation **8**
- 3.1 Initial Requirements . . . 8
- 3.2 One-time installation . . . 8

4 Creating new areas **9**
- 4.1 Adding a new area text file in `/config` folder . . . 9
- 4.2 Run SPC for the new area list . . . 9

5 Transform the output file **10**
- 5.1 Python . . . 10
- 5.2 Converting .pb file to JSON format . . . 10
- 5.3 Converting to numpy arrays . . . 11
- 5.4 Visualizing the output files. . . . 11

6 Outputs for England Counties **12**
- 6.1 Versioning . . . 13

7 Troubleshooting **14**
- 7.1 Docker . . . 14
- 7.2 Building proj . . . 14
- 7.3 Downloading . . . 15

II Understanding it **16**

8 The Data Schema **17**
- 8.1 Understanding the schema . . . 17

8.2 Modelling the daily activites - Flows . . . 17

9 Modelling methods **19**
9.1 Commuting flows . . . 19

10 Data sources **21**
10.1 Utility data . . . 21
10.2 County level data . . . 21
10.3 National data . . . 23

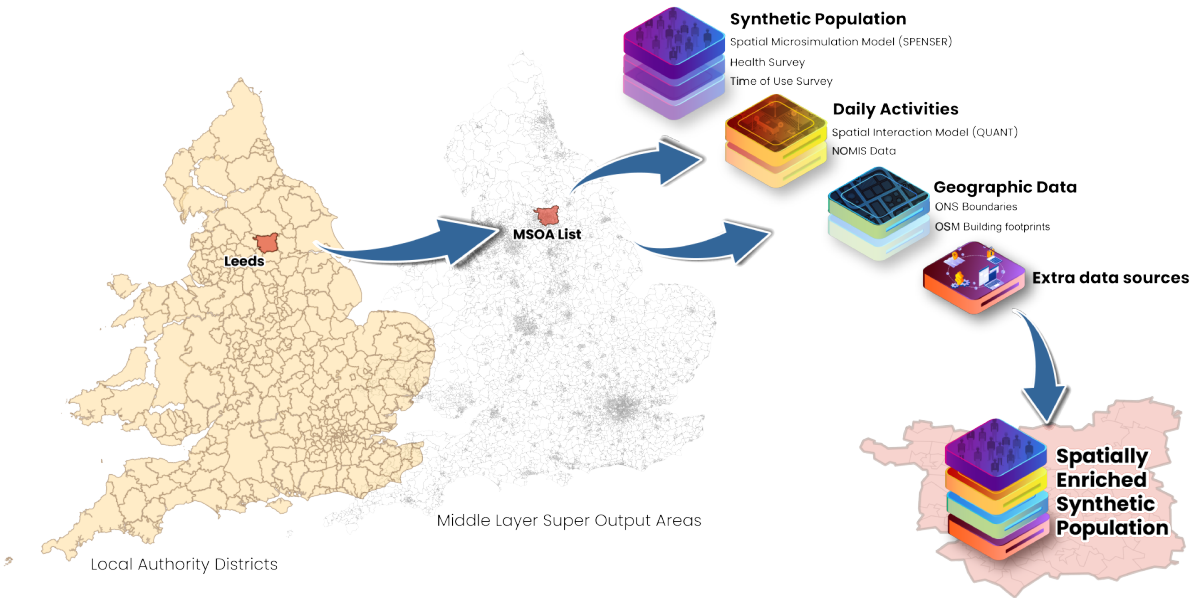
III Internals **24**

11 Developer guide **25**
11.1 Updating the docs . . . 25
11.2 Code hygiene . . . 25
11.3 Some tips for working with Rust . . . 25

12 Code walkthrough **27**
12.1 Generally useful techniques . . . 27
12.1.1 Split code into two stages . . . 27
12.1.2 Explicit data schema . . . 27
12.1.3 Type-safe IDs . . . 28
12.1.4 Idempotent data preparation . . . 28
12.1.5 Logging with structure . . . 28
12.1.6 Determinism . . . 29
12.2 Protocol buffers . . . 30

13 Performance **31**

1 Home



The Synthetic Population Catalyst (SPC) makes it easier for researchers to work with synthetic population data in England. It combines a variety of data sources and outputs a single file in protocol buffer format, describing the population in a given study area. The tool also provides methods to export the outcome in diferent formats often use for researchers like CSV or JSON.

The input of the SPC tool is a list of the Middle Layer Super Output Area (MSOAs) where you want to create a spatially enriched sythetic population to feed other dynamic models. SPC includes a script to assist you with the proper list of the MSOAs by defining a Local Authority District area in England. Get started to download SPC data or run the tool in different MSOAs.

You can also download this site as a PDF

Part I

Using it

2 Getting started

2.1 What SPC does?

The SPC generates spatially enriched synthetic population outputs for multiple administrative and census boundaries including Local Authority Districts - LADs and Counties across England. The output file generated by SPC has a granularity of Middle Layer Super Output Areas - MSOAs. This file is estructured and convided to help other researchers or urban analysts to feed other dynamic models for multiples propouses where a enriched sythetic population file is requiered. SPC includes an comprenhenvise set of variables that include social-demogragfics characterictis, dayliy activities and other extra data sources to help you model the complexity of British society.

For information about the data sources and the varibles integrated in the SPC outfile, visit data sources. For information about the administrative and census boundaries included in SPC visit the Open Geography portal from the Office for National Statistics (ONS).

2.2 What is the output generated by SPC?

SPC creates Google's protocol buffer files a platform-neutral, extensible mechanism for serializing structured data in bi-directional way. SPC encodes the enriched synthetic population schema in .proto files (.pb). You can read the "protobuf" (shorthand for a protocol buffer file) in any supported language, and then extract and transform just the parts of the data you want.

Note that per MSOA, very few venues are represented as destinations – 10 for retail and 5 for school. Only the most likely venues from QUANT are used.

See the ASPICS conversion script for all of this in action – it has a function to collapse a person's flows down into a single weighted list.

2.3 Areas ready to use and explore

If all you need is to explore the enriched syntetic population file and mesuare if fits to you model, we have created the output file for **all counties in England**, so if the area you want

to model is listed or you just want to explore the data, no need to run SPC yourself – just download data for your area.

If you are interested in create another area or combination of MSOAs, you will need then install and run SPC in your environment.

3 Installation

3.1 Initial Requirements

- **Rust**: The latest version of Rust (1.60): https://www.rust-lang.org/tools/install
- For Mac users, we recommend to have installed **Homebrew**: https://brew.sh/
- **Python**, there are some scrips and functionalites inside SPC that requiere python. https://www.python.org/downloads/

3.2 One-time installation

```
git clone https://github.com/alan-turing-institute/uatk-spc/
cd uatk-spc
# The next command will take a few minutes the first time you do it, to build external depen
cargo build --release
```

If you get some errors during the compilation process, take a look at Troubleshooting section.

4 Creating new areas

Once SPC is installed and you need to create a synthetic population for a **new area** not listed in one of our already genereted (outputs)[outputs.qmd], you need to do the following steps:

4.1 Adding a new area text file in `/config` folder

SPC requieres a list of MSOAs for a given area. The MSOA code is not a conventional way to describe an area, so we have created a python script to assist you create the requiered input file for SPC.

Navigate to the folder `scripts` within SPC folder, and run `python select_msoas.py`, this script will ask you the name of the LAD or County you are interest in. Refer to `data/raw_data/referencedata/lookUp.csv` for all geographies that SPC support.

A new file will be created in your config folder `config/your_region.txt` with the list of MSOAs.

4.2 Run SPC for the new area list

After you write a new file, you simply run the pipeline with that as input, make sure you are located in the `spc` folder

```
cargo run --release -- config/your_region.txt
```

If you like to take a quick look of how SPC process the data, you can use one of our small examples lists:

```
cargo run --release -- config/bristol.txt
```

This will download some large files the first time. You'll wind up with `data/output/bristol.pb` as output, as well as lots of intermediate files in `data/raw_data/`. The next time you run this command (even on a different study area), it should go much faster.

5 Transform the output file

Once you run SPC to generate a new study area based on your input file text, you will get in `/data/output` folder a `.pb` file. A .pb or protocol buffer file efficiently encode the the enriched synthetic population schema. You can read the "protobuf" (shorthand for a protocol buffer file) in any supported language, and then extract and transform just the parts of the data you want.

5.1 Python

To work with SPC protobufs in Python, you need two dependencies setup:

- The protobuf library
  - You can install system-wide with `pip install protobuf`
  - Or add as a dependency to a conda, poetry, etc environment
- The generated Python library, synthpop_pb2.py
  - You can download a copy of this file into your codebase, then `import synthpop_pb2`
  - You can also generate the file yourself, following the docs: `protoc --python_out=python/ synthpop.proto`

5.2 Converting .pb file to JSON format

To interactively explore the data, viewing JSON is much easier. It shows the same structure as the protobuf, but in a human-readable text format. The example below uses a small Python script:

```
# Download a file
wget https://ramp0storage.blob.core.windows.net/spc-output/v1/rutland.pb.gz
# Uncompress
gunzip rutland.pb.gz
# Convert the .pb to JSON
python3 python/protobuf_to_json.py data/output/rutland.pb > rutland.json
```

```
# View the output
less rutland.json
```

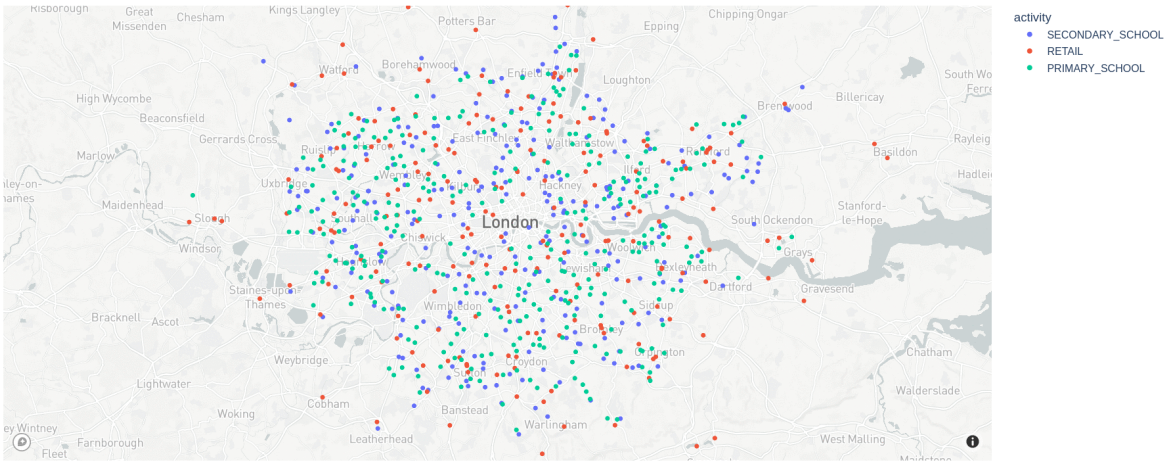
5.3 Converting to numpy arrays

The ASPICS project simulates the spread of COVID through a population. The code uses numpy, and this script converts the protobuf to a bunch of different numpy arrays.

Note the code doesn't keep using classes from the generated Python code for protobufs. The protobuf is a format optimized for reading and writing; you shouldn't use it in your model if there's a more appropriate tool you're familiar with, like data frames.

5.4 Visualizing the output files.

Use this script to read a protobuf file, then draws a dot for every venue, color-coded by activity.



6 Outputs for England Counties

This list contains 47 ceremonial counties – it's missing the City of London.

- bedfordshire
- berkshire
- bristol
- buckinghamshire
- cambridgeshire
- cheshire
- cornwall
  - Within this area, we're missing the Isles of Scilly (about 2,000 inhabitants)
- cumbria
- derbyshire
- devon
- dorset
- durham
- east_sussex
- east_yorkshire_with_hull
- essex
- gloucestershire
- greater_london
- greater_manchester
- hampshire (Southhampton)
- herefordshire
- hertfordshire
- isle_of_wight
- kent
- lancashire
- leicestershire
- lincolnshire
- merseyside (Liverpool)
- norfolk
- northamptonshire
- northumberland
- north_yorkshire

- nottinghamshire
- oxfordshire
- rutland
- shropshire
- somerset
- south_yorkshire
- staffordshire
- suffolk
- surrey
- tyne_and_wear (Newcastle)
- warwickshire
- west_midlands (Birmingham)
- west_sussex
- west_yorkshire (Leeds)
- wiltshire
- worcestershire

We also have a few special study areas defined:

- northwest_transpennine (Liverpool, Manchester, Leeds)
- oxford_cambridge_arc (Oxford, Milton Keynes, Cambridge)

See config/ for the list of MSOAs covered by each study area. If you want to run SPC for a different list of MSOAs, see here.

6.1 Versioning

Over time, we may add more data to SPC or change the schema. Protocol buffers are designed to let combinations of new/old code and data files work together, but we don't intend to use this feature. We may make breaking changes, like deleting fields. We'll release a new version of the schema and output data every time and document it here. You should depend on a specific version of the data output in your code, so new releases don't affect you until you decide to update.

- v1: released 25/04/2022, schema

7 Troubleshooting

Please open an issue if you have any trouble!

7.1 Docker

If you're having trouble building, you can run in Docker. Assuming you have Docker setup:

```
git clone https://github.com/alan-turing-institute/uatk-spc/
cd spc
# Build the Docker image initially. Once we publish to Docker Hub, this step
# won't be necessary.
docker build -t spc .
# Run SPC in Docker
docker run --mount type=bind,source="$(pwd)"/data,target=/spc/data -t spc /spc/target/releas
```

This will make the `data` directory in your directory available to the Docker image, where it'll download the large input files and produce the final output.

7.2 Building proj

Symptom: When you run `cargo build --release`, you get an error like:

```
 error: failed to run custom build command for 'proj-sys v0.18.4'
```

Cause: The Rust code depends on proj to transform coordinates. You may need to install additional dependencies to build it.

Suggested Solution

On Ubuntu, run:

```
apt-get install cmake sqlite3 libclang-dev
```

On Mac, run:

```
brew install pkg-config cmake proj
```

7.3 Downloading

If you have trouble downloading any of the large files, you can download them manually. The logs will contain a line such as `Downloading https://ramp0storage.blob.core.windows.net/nationaldata/ to data/raw_data/nationaldata/QUANT_RAMP_spc.tar.gz`. This tells you the URL to retrieve, and where to put the output file. Note that SPC won't attempt to download files if they already exist, so if you wind up with a partially downloaded file, you have to manually remove it.

Part II

Understanding it

8 The Data Schema

8.1 Understanding the schema

Here are some helpful tips for understanding the schema.

Each .pb file contains exactly one `Population` message. In contrast to datasets consisting of multiple `.csv` files, just a single file contains everything. Some of the fields in `Population` are lists (of people and households) or maps (of venues keyed by activity, or of MSOAs). Unlike a flat `.csv` table, there may be more lists embedded later. Each `Household` has a list of `members`, for example.

The different objects refer to each other, forming a graph structure. The protobuf uses `uint64` IDs to index into other lists. For example, if some household has `members = [3, 10]`, then those two people can be found at `population.people[3]` and `population.people[10]`. Each of them will have the same `household` ID, pointing back to something in the `population.households` list.

8.2 Modelling the daily activites - Flows

SPC models daily travel behavior of people as "flows." Flows are broken down by by an activity – shopping/retail, attending primary or secondary school, working, or staying at home. For each activity type, a person has a list of venues where they may do that activity, weighted by a probability of going to that particular venue.

Note that `flows_per_activity` is stored in `InfoPerMSOA`, not `Person`. The flows for retail and school are only known at the MSOA level, not individually. So given a particular `Person` object, you first look up their household's MSOA – `msoa = population.households[ person.household ].msoa` and then look up flows for that MSOA – `population.info_per_msoa[msoa].flows_per_activity`.

Each person has exactly 1 flow for home – it's just `person.household` with probability 1. A person has 0 or 1 flows to work, based on the value of `person.workplace`.

This doesn't mean that all people in the same MSOA share the same travel behavior. Each person has their own `activity_durations` field, based on time-use survey data. Even if two

people share the same set of places where they may go shopping, one person may spend much more time on that activity than another.

See the ASPICS conversion script for all of this in action – it has a function to collapse a person's flows down into a single weighted list.

How do you interpret the probabilities/weights for flows? If your model needs people to visit specific places each day, you could randomly sample a venue from the flows, weighting them appropriately. For retail, you may want to repeat this sampling every day of the simulation, so they visit different venues. For primary and secondary school, it may be more appropriate to sample once and store that for the simulation – a student probably doesn't switch schools daily.

Alternatively, you can follow what ASPICS does. Every day, each person logically visits all possible venues, but their interaction there (possibly receiving or transmitting COVID) is weighted by the probability of each venue.

9 Modelling methods

The principles behind the generation of the enriched SPENSER population data and behind the modelling of trips to schools and retail from QUANT are detailed in

> Spooner F et al. A dynamic microsimulation model for epidemics. Soc Sci Med. 291:114461 (2021). (DOI)

9.1 Commuting flows

In order to distribute each individual of the population to a unique physical workplace, we first created a population of all individual workplaces in England, based on a combination of the Nomis UK Business Counts 2020 dataset and the Nomis Business register and Employment Survey 2015 (see Data sources). The first dataset gives the number of individual workplace counts per industry, using the SIC 2007 industry classification, with imprecise size (i.e. number of employees) bands at MSOA level. The second dataset gives the total number of jobs available at LSOA level per SIC 2007 industry category. We found that the distribution of workplace sizes follows closely a simple 1/x distribution, allowing us to draw for each workplace a size within their band, with sum constraints given by the total number of jobs available, according to the second dataset.

The workplace 'population' and individual population are then levelled for each SIC 2007 category by removing the exceeding part of whichever dataset lists more items. This takes into account that people and business companies are likely to over-report their working availability (e.g. part time and seasonal contracts are not counted differently than full time contracts, jobseekers or people on maternity leave might report the SIC of their last job). This process can be controlled by a threshold in the parameter file that defines the maximal total proportion of workers or jobs that can be removed. If the two datasets cannot be levelled accordingly, the categories are dropped and the datasets are levelled globally. Tests in the West Yorkshire area have shown than when the level 1 SIC, containing 21 unique categories, is used, 90% of the volume of commuting flows were recovered compared to the Nomis commuting OD matrices at MSOA level.

The employees for each workplace are drawn according to the 'universal law of visitation', see

Schläpfer M et al. The universal visitation law of human mobility. Nature 593, 522–527 (2021). (DOI)

This framework predicts that visitors to any destination follow a simple

(r,f)= K / (rf)2

distribution, where (r,f) is the density of visitors coming from a distance r with frequency f and K is a balancing constant depending on the specific area. In the context of commuting, it can be assumed that f = 1. Additionally, we only need to weigh potential employees against each other, which removes the necessity to compute explicitly K. In the West Yorkshire test, we found a Pearson coefficient of 0.7 between the predicted flows when aggregated at MSOA level and the OD matrix at MSOA level available from Nomis.

10 Data sources

The data is sorted around the 2011 Middle-layer Super Output Area (MSOA) geographical unit. These units where created for census collection and are designed to be relatively homogeneous, with an average population size of 8000. Any list of MSOAs in England can be run, with the exception of the MSOAs forming the City of London (i.e the London borough called the City, not London as a whole).

The data from Open Street Map (OSM) is downloaded directly from https://www.openstreetmap.org. Everything else is hosted as local copies on one Azure repository that interacts automatically with the model, and divided into utilities, county level data and national data.

10.1 Utility data

`lookUp.csv`

The look-up table links different geographies together. It is used internally by the model, but can also help the user define their own study area. `MSOA11CD`, `MSOA11NM`, `LAD20CD`, `LAD20NM`, `ITL321CD`, `ITL321NM`, `ITL221CD`, `ITL221NM`, `ITL121CD`, `ITL121NM` are all standard denominations fully compatible with ONS fields of the same name. They are based on ONS lookups. See ONS documentation for more details. `CTY20NM` and `CCTY20NM` are custom denominations for the counties of England (used to sort the county level population data) and the ceremonial counties of England respectively. Their spelling may vary in different data sources and the field `CTY20NM` is not compatible with the ONS field of the same name (which excludes all counties that are also unitary authorities). `GoogleMob` and `OSM` are different spellings for the counties of England used by Google and OSM for their data releases.

10.2 County level data

Contains 47 files, each representing the population in 2020 of one of the counties of England mentioned above, and named

`tus_hse_<county_name>.gz`

This data is based on the 2011 UK census, the Time Use Survey 2014-15 and the Health Survey for England 2017. The SPENSER (Synthetic Population Estimation and Scenario Projection) microsimulation model (reference) distributes a synthetic population based on the census at MSOA scale and projects it to 2020 according to estimates from the Office for National Statistics (ONS). This information was enriched with some of the content of the other two datasets through propensity score matching (PSM) by Prof. Karyn Morrissey (Technical University of Denmark). The rest of the datasets can be added *a posteriori* from the identifiers provided.

The fields currently contained are:

- `idp`: a unique individual identifier within the present data
- `MSOA11CD`: MSOA code where the individual lives
- `hid`: household identifier, includes communal establishments
- `pid`: identifier linking to the 2011 Census
- `pid_tus`: identifier linking to the Time Use Survey 2015
- `pid_hse`: identifier linking to the Health Survey for England 2017
- `sex`: 0 female; 1 male
- `age`: in years
- `origin`: 1 White; 2 Black; 3 Asian; 4 Mixed; 5 Other
- `nssec5`: National Statistics Socio-economic classification:
  - 1: Higher managerial, administrative and professional occupations
  - 2: Intermediate occupations
  - 3: Small employers and own account workers
  - 4: Lower supervisory and technical occupations
  - 5: Semi-routine and routine occupations
  - 0: Never worked and long-term unemployed
- `soc2010`: Previous version of the Standard Occupational Classification
- `sic1d07`: Standard Industrial Classification of Economic Activities 2007, 1st layer (number corresponding to the letter in alphabetical order)
- `sic2d07`: Standard Industrial Classification of Economic Activities 2007, 2nd layer
- Proportion of 24h spent doing different daily activities:
  - `punknown` + `pwork` + `pschool` + `pshop` + `pservices` + `pleisure` + `pescort` + `ptransport` = `pnothome`
  - `phome` + `pworkhome` = `phometot`
  - `pnothome` + `phometot` = 1
- `cvd`: has a cardio-vascular disease (0 or 1)
- `diabetes`: has diabetes (0 or 1)
- `bloodpressure`: has high blood pressure (0 or 1)
- `BMIvg6`: Body Mass Index:
  - Not applicable

  - Underweight: less than 18.5
  - Normal: 18.5 to less than 25
  - Overweight: 25 to less than 30
  - Obese I: 30 to less than 35
  - Obese II: 35 to less than 40
  - Obese III: 40 or more
- `lng`: longitude of the MSOA11CD centroid
- `lat`: latitude of the MSOA11CD centroid

Some other fields were kept from specific projects but are not from official sources and should be used.

10.3 National data

`businessRegistry.csv`

Contains a breakdown of all business units (i.e. a single workplace) in England at LSOA scale (smaller than MSOA), estimated by the project contributors from two nomis datasets: UK Business Counts - local units by industry and employment size band 2020 and Business Register and Employment Survey 2015. Each item contains the `size` of the unit and its main `sic1d07` code in reference to standard Industrial Classification of Economic Activities 2007 (number corresponding to the letter in alphabetical order). It is used to compute commuting flows.

`MSOAS_shp.tar.gz`

Is a simple shapefile taken from ONS boundaries.

`QUANT_RAMP.tar.gz`

See: Milton R, Batty M, Dennett A, dedicated RAMP Spatial Interaction Model GitHub repository. It is used to compute the flows towards schools and retail.

`timeAtHomeIncreaseCTY.csv`

This file is a subset from Google COVID-19 Community Mobility Reports, cropped to England. It describes the daily reduction in mobility, averaged at county level, due to lockdown and other COVID-19 restrictions between the 15th of February 2020 and 15th of April 2022. Missing values have been replaced by the national average. These values can be used directly to reduce `pnothome` and increase `phometot` (and their sub-categories) to simulate more accurately the period.

Part III

Internals

11 Developer guide

11.1 Updating the docs

The site is built with Quarto. You can iterate on it locally: `cd docs; quarto preview`

11.2 Code hygiene

We use automated tools to format the code.

```
cargo fmt

# Format Markdown docs
prettier --write *.md docs/*.md
```

Install prettier for Markdown.

11.3 Some tips for working with Rust

There are two equivalent ways to rebuild and then run the code. First:

```
cargo run --release -- devon
```

The `--` separates arguments to `cargo`, the Rust build tool, and arguments to the program itself. The second way:

```
cargo build --release
./target/release/aspics devon
```

You can build the code in two ways – **debug** and **release**. There's a simple tradeoff – debug mode is fast to build, but slow to run. Release mode is slow to build, but fast to run. For the

ASPICS codebase, since the input data is so large and the codebase so small, I'd recommend always using `--release`. If you want to use debug mode, just omit the flag.

If you're working on the Rust code outside of an IDE like VSCode, then you can check if the code compiles much faster by doing `cargo check`.

12 Code walkthrough

12.1 Generally useful techniques

The code-base makes use of some techniques that may be generally applicable to other projects.

12.1.1 Split code into two stages

Agent-based models and spatial interaction models require some kind of input. Often the effort to transform external data into this input can exceed that of the simulation component. Cleanly separating the two problems has some advantages:

- iterate on the simulation faster, without processing raw data every run
- reuse the prepared input for future projects
- force thinking about the data model needed by the simulation, and transform the external data into that form

12.1.2 Explicit data schema

Dynamically typed languages like Python don't force you to explicitly list the shape of input data. It's common to read CSV files with `pandas`, filter and transform the data, and use that throughout the program. This can be quick to start prototyping, but is hard to maintain longer-term. Investing in the process of writing down types:

- makes it easier for somebody new to understand your system – they can first focus on **what** you're modeling, instead of how that's built up from raw data sources
- clarifies what data actually matters to your system; you don't carry forward unnecessary input
- makes it impossible to express invalid states
    - One example is here – per person and activity, there's a list of venues the person may visit, along with a probability of going there. If the list of venues and list of probabilities are stored as separate lists or columns, then their length may not match.

- reuse the prepared input for future projects

There's a variety of techniques for expressing strongly typed data:

- protocol buffers or flatbuffers
- JSON schemas
- Python data classes and optional type hints
- statically typed languages like Rust

12.1.3 Type-safe IDs

Say your data model has many different objects, each with their own ID – people, households, venues, etc. You might store these in a list and use the index as an ID. This is fine, but nothing stops you from confusing IDs and accidentally passing in venue 5 to a function instead of household 5. In Rust, it's easy to create "wrapper types" like this and let the compiler prevent these mistakes.

This technique is also useful when preparing external data. GTFS data describing public transit routes and timetables contains many string IDs – shapes, trips, stops, routes. As soon as you read the raw input, you can store the strings in more precise types that prevent mixing up a stop ID and route ID.

12.1.4 Idempotent data preparation

If you're iterating on your initialisation pipeline's code, you probably don't want to download a 2GB external file every single run. A common approach is to first test if a file exists and don't download it again if so. In practice, you may also need to handle unzipping files, showing a progress bar while downloading, and printing clear error messages. This codebase has some common code for doing this in Rust. We intend to publish a separate library to more easily call in your own code.

12.1.5 Logging with structure

It's typical to print information as a complex pipeline runs, for the user to track progress and debug problems. But without any sort of organization, it's hard to follow what steps take a long time or break. What if your logs could show the logical structure of your pipeline and help you understand where time is spent?

```
[192.30s] [get_info_per_msoa] Loading buildings from data/raw_data/countydata/OSM/west-yorkshire-latest-free/
[192.64s] [get_info_per_msoa] Found 474,207 buildings from data/raw_data/countydata/OSM/west-yorkshire-latest-free/gis
osm_buildings_a_free_1.shp
[192.70s] [get_info_per_msoa] Matching 474,207 points to 299 polygons. Building R-Tree...
[194.22s] [calculate_lockdown_per_day] Calculating per-day lockdown values
[194.24s] [load_events] Loading events data
[194.25s] [initialisation] By the end, Memory usage: 1.53GiB
[200.89s] [Writing snapshot] Merging flows for all activities

  212.24s          initialisation WestYorkshireLarge
    31.18ms          grab_raw_data
    192.04s          creating population
      8.20s            read_individual_time_use_and_health_data
        4.35s            Reading "data/raw_data/countydata/tus_hse_west-yorkshire.csv"
        3.83s            Creating households
      152.38s          create_commuting_flows
      8.30s            setup_venue_flows Retail
        6.59s            Copying flows to people Retail
      7.47s            setup_venue_flows Nightclub
        6.63s            Copying flows to people Nightclub
      8.68s            setup_venue_flows PrimarySchool
        6.58s            Copying flows to people PrimarySchool
      7.00s            setup_venue_flows SecondarySchool
        6.50s            Copying flows to people SecondarySchool
    2.03s            get_info_per_msoa
    24.48ms          calculate_lockdown_per_day
    251.20µs         load_events "model_parameters/eventDataConcerts.csv"
    1.07s            Writing population to "data/processed_data/WestYorkshireLarge/rust_cache.bin"
    16.93s           Writing snapshot
```

The screenshot above shows a summary printed at the end of a long pipeline run. It's immediately obvious that the slowest step is creating commuting flows.

This codebase uses the tracing framework for logging, with a custom piece to draw the tree. (We'll publish this as a separate library once it's more polished.) The tracing framework is hard to understand, but the main conceptual leap over regular logging framworks is the concept of a **span**. When your code starts one logical step, you call a method to create a new span, and when it finishes, you close that span. Spans can be nested in any way – `create_commuting_flows` happens within the larger step of `creating population`.

12.1.6 Determinism

Given the same inputs, your code should always produce identical output, no matter where it's run or how many times. Otherwise, debugging problems becomes very tedious, and it's more difficult to make conclusions from results. Of course, many projects have a stochastic element – but this should be controlled by a random number generator (RNG) seed, which is part of the input. You vary the seed and repeat the program, then reason about the distribution of results.

Aside from organizing your code to let a single RNG seed influence everything, another possible source of non-determinism is iteration order. In Rust, a `HashMap` could have different order every time it's used, so we use a `BTreeMap` instead when this matters. In Python, dictionaries are ordered. Be sure to check for your language.

12.2 Protocol buffers

SPC uses protocol buffers for output. This has some advantages explained the "explicit data schema" section above, but the particular choice of protocol buffer has some limitations.

First, proto3 doesn't support required fields. This is done to allow schemas to evolve better over time, but this isn't a feature SPC makes use of. There's no need to have new code work with old data, or vice versa – if the schema is updated, downstream code should adapt accordingly and use the updated input files. The lack of required fields leads to imprecise code – a person's health structure is always filled out, but in Rust, we wind up with `Option<Health>`. Differentiating 0 from missing data also becomes impossible – `urn` is optional, but in protobuf, we're forced to map the missing case to 0 and document this.

Second, protocol buffers don't easily support type-safe wrappers around numeric IDs, so downstream code has to be careful not to mix up household, venue, and person IDs.

Third, protocol buffers support limited key types for maps. Enumerations can't be used, so we use the numeric value for the activity enum.

We'll evaluate flatbuffers and other alternative encodings.

Note that in any case, SPC internally doesn't use the auto-generated code until the very end of the pipeline. It's always possible to be more precise with native Rust types, and convert to the less strict types last.

13 Performance

The following tables summarizes the resources SPC needs to run in different areas.

study_area	num_msoas	num_households	num_people	pbf_file_size	runtime	commuting_runtime	memory_usage
bedfordshire	74	271,487	650,950	94.33MiB	13 seconds	6 seconds	293.76MiB
berkshire	107	363,653	878,045	127.21MiB	14 seconds	7 seconds	300.25MiB
bristol	55	196,230	456,532	68.11MiB	5 seconds	2 seconds	151.48MiB
buckinghamshire	99	324,843	759,879	109.30MiB	10 seconds	5 seconds	297.42MiB
cambridgeshire	98	346,532	834,141	120.30MiB	13 seconds	7 seconds	300.23MiB
cheshire	139	463,106	1,040,634	150.80MiB	15 seconds	6 seconds	304.91MiB
cornwall	74	246,873	564,604	83.30MiB	7 seconds	2 seconds	277.31MiB
cumbria	64	224,779	485,035	70.69MiB	8 seconds	3 seconds	152.02MiB
derbyshire	131	457,791	1,024,952	148.85MiB	17 seconds	9 seconds	304.58MiB
devon	156	521,790	1,178,315	171.93MiB	20 seconds	10 seconds	556.07MiB
dorset	95	344,246	751,334	109.25MiB	10 seconds	5 seconds	298.97MiB
durham	117	406,164	904,785	130.75MiB	10 seconds	4 seconds	299.72MiB
east_sussex	102	380,180	830,761	120.14MiB	11 seconds	5 seconds	299.81MiB
east_yorkshire_with_hull	75	261,267	579,746	85.31MiB	7 seconds	2 seconds	278.80MiB
essex	211	771,734	1,798,893	261.01MiB	34 seconds	23 seconds	600.82MiB

study_area	num_msoas	num_households	num_people	file_size	runtime	commuting_runtime	memory_usage
gloucestershire	107	392,120	901,395	129.84MiB	13 seconds	5 seconds	302.14MiB
greater_london	983	3,135,814	8,672,103	1.23GiB	11 minutes	10 minutes	4.32GiB
greater_manchester	346	871,651	2,746,858	389.21MiB	2 minutes	84 seconds	1.08GiB
hampshire	225	775,203	1,803,991	262.78MiB	41 seconds	29 seconds	601.86MiB
herefordshire	23	83,115	191,282	27.72MiB	4 seconds	1 second	74.83MiB
hertfordshire	153	492,783	1,144,974	165.59MiB	19 seconds	11 seconds	555.75MiB
isle_of_wight	18	64,602	135,125	20.48MiB	3 seconds	1 second	70.05MiB
kent	220	692,896	1,808,206	259.58MiB	29 seconds	18 seconds	599.43MiB
lancashire	191	603,524	1,472,550	210.80MiB	22 seconds	12 seconds	592.20MiB
leicestershire	120	417,621	1,043,283	147.79MiB	17 seconds	11 seconds	302.29MiB
lincolnshire	134	473,854	1,064,174	153.14MiB	13 seconds	6 seconds	551.97MiB
merseyside	184	546,791	1,401,012	200.02MiB	25 seconds	17 seconds	590.71MiB
norfolk	110	379,188	891,006	130.04MiB	11 seconds	4 seconds	302.39MiB
north_yorkshire	138	434,489	1,069,514	154.54MiB	14 seconds	6 seconds	552.40MiB
northamptonshire	91	293,580	733,190	106.21MiB	10 seconds	5 seconds	297.24MiB
northumberland	40	113,436	316,618	45.27MiB	5 seconds	1 second	139.76MiB
northwest_transpennine	829	2,378,868	6,419,933	928.03MiB	6 minutes	6 minutes	2.32GiB
nottinghamshire	138	413,097	1,139,096	163.50MiB	19 seconds	11 seconds	553.29MiB
oxford_cambridge_arc	353	1,152,245	2,823,838	409.60MiB	2 minutes	78 seconds	1.16GiB
oxfordshire	86	254,974	669,237	96.27MiB	9 seconds	4 seconds	279.53MiB

study_area	num_msoas	num_households	num_people	pb_file_size	runtime	commuting_runtime	memory_usage
rutland	5	16,688	39,475	5.52MiB	2 seconds	1 second	18.60MiB
shropshire	62	153,284	497,064	70.66MiB	6 seconds	2 seconds	150.19MiB
somerset	124	384,165	944,394	138.07MiB	14 seconds	6 seconds	302.66MiB
south_yorkshire	172	358,717	1,373,401	191.33MiB	25 seconds	18 seconds	554.04MiB
staffordshire	143	439,176	1,104,925	157.81MiB	14 seconds	7 seconds	552.70MiB
suffolk	90	326,760	739,296	106.71MiB	9 seconds	4 seconds	296.93MiB
surrey	151	461,466	1,136,090	164.06MiB	20 seconds	13 seconds	555.61MiB
tyne_and_wear	145	414,128	1,111,239	157.15MiB	12 seconds	6 seconds	551.14MiB
warwickshire	108	319,511	933,391	132.32MiB	16 seconds	10 seconds	300.49MiB
west_midlands	314	523,264	2,475,918	348.50MiB	38 seconds	25 seconds	1.05GiB
west_sussex	100	373,326	838,440	122.18MiB	11 seconds	4 seconds	300.54MiB
west_yorkshire	299	960,426	2,272,063	331.50MiB	42 seconds	28 seconds	1.08GiB
wiltshire	89	305,679	686,963	100.69MiB	8 seconds	3 seconds	295.39MiB
worcestershire	85	254,383	572,751	83.91MiB	8 seconds	3 seconds	277.79MiB

Notes:

- `pb_file_size` refers to the size of the uncompressed protobuf file in `data/output/`
- The total `runtime` is usually dominated by matching workers to businesses, so `commuting_runtime` gives a breakdown
- Measuring memory usage of Linux processes isn't straightforward, so `memory_usage` should just be a guide
- These measurements were all taken on one developer's laptop, and they don't represent multiple runs. This table just aims to give a general sense of how long running takes.
  - That machine has 16 cores, which matters for the parallelized commuting calculation.

- `scripts/collect_stats.py` produces the table above