APDUDS

Proposal for possible continuation and expansion

Summary

With a basic version of APDUDS completed, it has shown real-world use potential if expanded and refined if more real-world data about an area is included. If the results are better checked and designed for realism, almost no manual adjusting of the basic aspect of the network can be achieved.

Two significant expansions came out of a meeting regarding this subject: The incorporation of area elevation and land-use data. Almost any area has some slight elevation changes, and so almost all systems that APDUDS makes are not realistic, as they ignore the needed design changes to work around/with this terrain feature. The package used for downloading the road network (OSMnx) also has its own elevation data downloader, which incorporates the height information directly into the network using google earth. With this, making a gravity-based system that also adheres to the real-world elevation only requires the addition of extra restrictions to the flow-direction calculations. To make the subcatchment area calculations work, the current closest point method (2D Voronoi) can be swapped out for a unit-by-unit rasterized approach, which finds its closest drainage point by looking at the 3D space instead of 2D. It also allows for easier implementation of the second main feature, the land-use incorporation.

APDUDS currently ignores the actual distribution of greenery, water, or build-up space in an area, letting the user only enter a single value to determine the drainage rate for the entire area. This is not realistic, as of course spaces like ponds or forest drain result in very little to no water draining into the sewer. OpenStreetMap already has a feature in which patches of land can be designated with certain tags, such as ‘forest’, ‘water’, or ‘residential’, which can also be downloaded using OSMnx. With this and the rasterized approach needed for the elevation inclusion, it is possible to make the drainage values for each subcatchment a custom value that is determined by the distribution of terrain. Taking all the different pixels inside a subcatchment and finding an average drainage rate between them will give more realistic and representative inflow values.

The two features described above will make the software significantly more complex. To make this advanced version easier to use, a user interface should be implemented to make all the new options and results manageable. It also comes with many additional advantages, like allowing for easy backtracking and variable adjustment, having information and graphs shown in an easier-to-digest format, making save files easier to implement, and giving the user more context and explanations of the workings of the software. A list of secondary features is also given. While these are not necessary for the completion of this new version of APDUDS, they will improve the ease of use, accuracy of the results, and stability of the codebase.

Lastly, a development time is estimated to complete the main features. The complexity of this new version will likely be over double the complexity of the previous one. Considering the previous version was made in 7 weeks with around 45 hours per week, and work on this version can only be performed in up to 20 hours per week, half a year is seen as the most reasonable estimate for obtaining a sufficient working version. If the inclusion of the secondary features and further optimization and testing are desired, the development time will likely be more than three-quarters of a college year.

Table of Contents

1. Main Features

1.1. Elevation Incorporation

As almost any area has some slight height changes, most systems that APDUDS makes are unrealistic, as they ignore the needed design changes to work around/with these elevation changes. To make the software viable for real-world use this data must be included and incorporated into the designing process.

OSMnx (package used for downloading the road network) can access Google Earth elevation data using build-in functions. This greatly simplifies the coding process as no new packages are needed. All that is needed to use these functions is a Google Maps API Key, and it returns the elevation data for all nodes in the area. With this, multiple parts of the current calculations done by APDUDS need to be reworked to include elevation data.

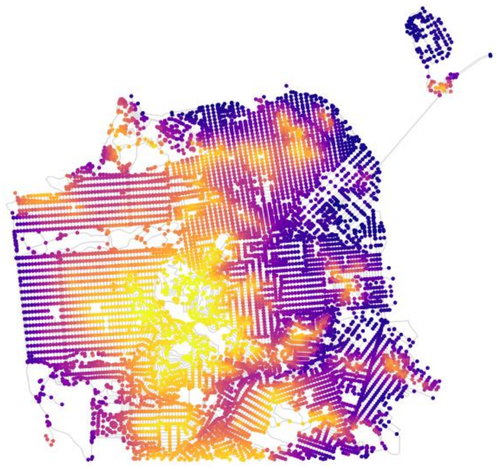

Figure 1: Example of Nodal Elevation Data Using OSMNX

1.1.1. Catchment Area for each Node

Now that elevation is included, a subarea close to a particular node may not always flow towards that node, as, for example, the area may be downhill from it. The current 2D Voronoi (closest neighbor) method could be upgraded to a 3D Voronoi, but as this gives a volume for each node, more complicated math is needed to obtain the subcatchment planes from these volumes.

An arguably easier-to-adjust solution is rasterization. Splitting the area into "pixels" or "units" will allow each unit to calculate its own best route to a drainage point. It also allows for greater control over detail, as the user can manually set the size of each pixel. (It also simplifies the Land-use Incorporation). The middle of each pixel in the raster is the point from which we calculate the distance to surrounding nodes. If this distance requires going uphill, we can remove that option from the possible drainage points. The shortest 3D distance from all the downhill nodes, calculated using Pythagoras, determines the final drainage node. Extrapolation of the node elevation data to fill the entire area means we can also get the elevation of each pixel. All pixels that flow towards a particular node are consolidated into a single polygon.

This solution will require safeguarding against edge cases. When, for example, there is a hill between a raster pixel and a possible drainage node, but the latter is still lower than the former, it will still consider the latter a viable endpoint. Sampling points along the path might provide the solution, as they should all be above or at the terrain elevation. If this is not the case, the route must be going uphill somewhere, invalidating it. This can be repeated for the 2nd and 3rd place shortest path (etc) until a truly viable route is found.

1.1.2. Direction of Flow through the Conduits

With the area elevation in mind, the possible flow direction of the conduits becomes significantly more limited. Assuming we still want to adhere to a gravity-based design philosophy, conduits at an angle only allow flow in one direction, meaning that only for very flat areas system could have two-directional flow throughout.

Incorporating this into the current flow direction scheme is done by switching the type of graph used for the route determination. Currently, a standard NetworkX Graph is used, which is undirected. A DiGraph can be used instead, which is directed, meaning that an “edge” or conduit only allows travel in specified directions. The rest of the routing operations remain largely the same as in the current version with this substitution.

Adding this directionality does allow the possibility for “unsolvable” graph creation. Choosing the pumping point as the highest part of the network will make water unable to flow out of the system without having the pumping point at unreasonable depths. With the location of the pumping point selected by the user, a relatively simple option to solve this will be to throw an error when the graph is unsolvable and ask the user to pick a different point.

This unreachable pumping point is not the only issue that can occur. Entire sections being situated lower than the rest of the network or small but extreme hills or valleys in the terrain can cause all sorts of issues for the system. The overflow location selection can cause similar issues to the pumping point problem. These will all have to be dealt with as they come along, as it is not yet clear which problems (and in what form) can occur.

1.2. Land-use Incorporation

Currently, APDUDS assumes that the entire area has the same impervious ground percentage, meaning that the amount of water that flows from a subcatchment into the system is determined only by the size of the catchment for the entire area. This is not very realistic as, for example, a creek or forest lets nearly zero percent of the water that falls on it drain into the system. To account for this, we need to use the actual terrain disruption or "land-use" distribution.

OpenStreetMap has a feature that lets users designate areas by "land-use". OSMnx allows downloading of this data in the form of polygons.

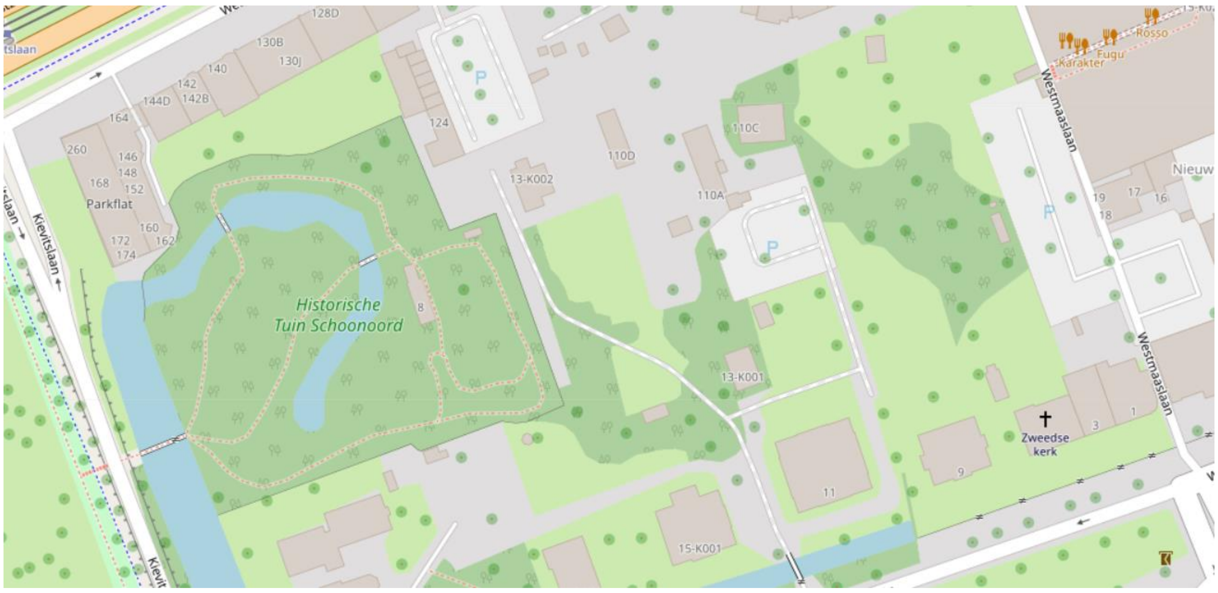

Figure 2: Screenshot of OpenStreetMap showing residential area (gray), meadow (light green), forest (dark green), and water (blue)

With the polygons obtained and the implementation of rasterization explained in <u>2.1.1 Catchment Area for each Node,</u> constructing subcatchment areas that use this land-use information is possible. Once the shortest path drainage calculations of <u>2.1.1</u> are done, the software will check what kind of land-use polygon the pixel resides in and assign that pixel a corresponding impervious percentage. Once all the pixels for a node can be consolidated, the resulting value for the total subcatchment will be the average of all the pixels it contains.

The user manually sets the impervious percentage values for different kinds of land use (with default values given if the user does not enter anything), which allows for more worldwide use as the values for the different land uses might differ per situation or country.

1.3. User Interface

With the incorporation of the features described in sections 2.1 and 2.2, the software will become significantly more complex than its previous iteration. With more features, options, and adjustable values, operating APDUDS only through the terminal has the possibility of becoming so cumbersome and unintuitive to the point that using it is more of a hassle dan doing it by hand. That is why it will be important to have a user interface (UI) going forward. Not only will this create a better user experience (UX), but it will also allow for more advanced options and functionalities, such as backtracking, saving, and simultaneous adjusting of values. The features are explained in more detail further down in this section.

Just as in the current version of APDUDS, the UI will be split into three main windows: The area selection/downloader stage, the attribute calculation/designing stage, and lastly the SWMM conversion stage. Splitting the UI will make each separate window contain less information, making it easier to navigate. It will also make it clear to the user which steps need to be completed first to access the later ones.

(Note: The design below is an initial design of what a possible UI may look like and should not be seen as the required lay-out/look to make a UI for APDUDS work).

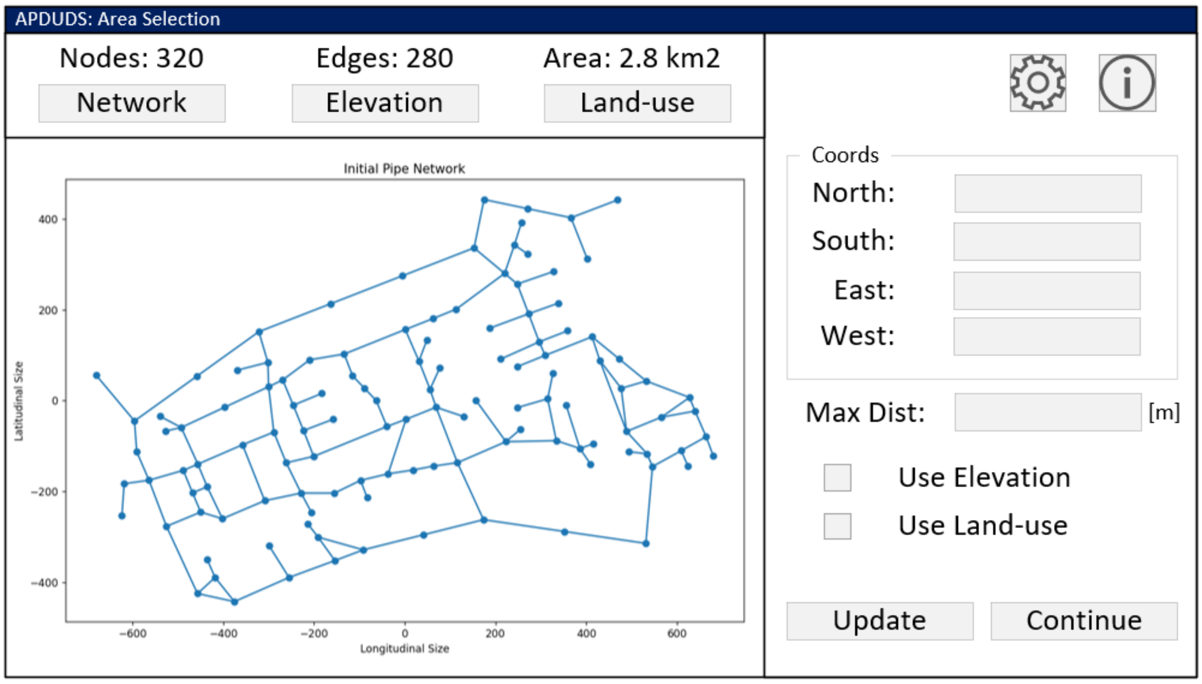

Figure 3: Initial design for Area Selection window (window elements not to scale)

Free variable adjustment

The example of the first window seen above highlights most of the possible advantages a UI will have. The first, most basic one that improves the UX the most is allowing variables to be changed and updated out of sequence. The user can enter the coordinates and max distance or check the check boxes in whatever order they like and can go back and change them immediately without having to enter any additional commands.

Better Graph Viewing

With the incorporation of the elevation and land-use data, lots of other ways to visualize the data and results are possible. Having all these new graphs shown to the user at once in a single window will require each graph to become smaller to make room for the others, losing readability. A UI will allow the user to switch between individual graphs, which can then take up a significantly larger part of the screen, making the viewing and analysing of these graphs much easier.

Easier Backtracking to Previous Steps

In the UI, if a user wants to return to a previous step (for example from the attribute calculation back to the downloading stage), they will have to re-enter all variables of that step even if they want to change just one. With a UI, the values that they entered can be stored and, once the user want to go back to a previous step, can be automatically re-entered, saving time for the user.

Constant Information Display

If information is printed to the terminal, the user will have to constantly re-request it if they want it to remain visible, as going through the designing process will constantly move the prints higher up due to new prints being added. With a UI this information can be constantly displayed, again allowing for quicker and easier result observation and analyzation.

Adding Save Files

The in-use variables saving that is done to facilitate the backtracking and variable updating can be extended by allowing the user to save the variables that they entered onto a separate file. This allows users to be able to close the program and to quickly come back to the project they were working one. Most importantly, it allows users to share specific variable and setting setups with each other.

Easier Addition of Smaller Settings

A lot of smaller variables regarding the downloaded data need to be set in order to make the designing process work. Will most of these variables will not have any large impact on the final results, giving the user the options to tweak these will allow for even more personalization without needing to dive into the code.

Help and Pop-up Windows

In a UI setting it becomes a lot easier to add an explanation for each variable, button and graph, lowering the possibility of confusion or wrong analyzation of results for new users, thus improving UX. Pop-up windows can also be implement to, for example, tell the user on which download or calculation step the code is, if there is an error with the data, or that one of their variables could or did cause problems.

The list of possible advantages can be extended even more, but by highlighting the biggest ones listed above it can be strongly argued that to take this software go from a gimmick to a full program, a UI is almost required.

2. Secondary Features/Changes

Next to the main features, other smaller improvements can be made, listed below.

2.1. Convert to Standalone Executable

Currently, the software needs to be run in an integrated development environment (IDE) like, for example, VS Code. Not only this, but to run the code other packages need to be installed via both conda and git, which requires some knowledge of the command line. This could deter possible users who are unexperienced with either the command line or IDE's. To make APDUDS usable for as many people as possible, it should be converted to a stand-alone executable (in similar fashion to the SWMM program).

2.2. Codebase File Restructuring

If all the main features are to be implemented, many extra scripts, folders and files will be created. The current version of APDUDS does not need a strict file naming and codebase structure, with these additions it would be advantageous to implement both, mainly to make making adjustment to the code easier for outsider who want to make their own changes or adjustments.

2.3. Create Tests

As the new features (mainly the UI) will make the interconnectivity of the codebase go up significantly, implementing basic testing to ensure code stability will be essential, also aiding in bug-fixing and assessing the validity of results.

2.4. Improve Quality of Road Network

APDUDS currently ignores the curvature of the roads. While not a big issue for relatively straight road network, it can give large inaccuracies for subcatchment areas and flow directions for more curvier ones. With the expansion of the software a large reality-ignoring issue like this should be fixed if possible.

2.5. Choose a New Name

APDUDS is just a mouthful.

3. Development Time

Making a quantified estimate for the required development time is difficult, as it is not known specifically how long the implementation of each feature will take, and which kinds of issues will be encountered. What is known however is that incorporating the features into the existing code will dramatically increase the complexity of the code and will likely require much more tweaking and finetuning than seen during the development of the current version.

During the Bachelor End Project, the average time spent on the software was around 40 hours per week. As I intend to partly continue with my masters the available development time is limited to a maximum of around 20 hours per week. With this and what was mentioned in the previous paragraph, the rough estimate for incorporating the new feature and getting APDUDS to an acceptable working state hovers around a half a year. If further optimization, edge-case handling, and incorporation of some if not all of the secondary features are desired this figure hovers at around three-quarters of a year.