RENESENG 4th Year Deliverables

Report for Resource Mapping for Supply Chains Optimisations

March 2018

Dr. Nathanial Cooper
Marie Curie ITN Experienced Researcher
ER 3.2

Imperial College London, Chemical Engineering, London

Report for the Roadmap Visualisation of Supply Chains

Specification & Functionalities

The purpose of this algorithm is to extract data sets that are necessary for biomass supply chain optimisation algorithms from map data. It provides a tool which can create a cellular grid, analyse the data sets that are provided, aggregate that data to the cellular grid and export the data to a form which is read by the GAMS optimisation. This is beneficial as it offers a reliable way to obtain all of the information that the optimisation needs so the procedure is efficient and consistent. This will reduce the time it takes to process data and reduce the chance of errors in processing.

The resource mapping tool provides several key matrices which are necessary for the supply chain optimisation:

First, it creates a cellular grid and provides relevant state properties of the grid. The cellular grid is necessary for the optimisation, so the tool creates one for the user based on some parameters that they provide. The tool gives the user the opportunity to save the shapefile of the grid that is produced. The tool also calculates several important parameters about the grid and exports these, such as the area of each grid cell, the linear distance between each grid cell and tortuosity between grid cells based on adjacency.

Second, it analyses and calculates aggregate data for each grid cell based on the provided map data sets. It uses the constraint map and the land use map to find the total arable land area per grid cell and outputs the matrix of that information. It also uses the constraint map, the land use map and the yield maps to output the average yield for the entire cell over arable land and provides this as a matrix as well.

All of these data sets can be saved by the user for later use in the optimisation. They are also added to the GIS interface by default (this can be turned off by the user if they wish). From here the data sets and the cellular grid shapefile can be examined in the GIS interface. It is worth noting that the data sets cannot be used outside of the GIS program while they are open within the program. They can either be removed from GIS session or the GIS session can be closed, however, mitigating this concern.

The resource mapping tool is designed to be used in conjunction with an open source geographic information software known as Quantum GIS or QGIS. This is a software that displays, analyses and modifies geographic data, and can handle a variety of raster and shapefile data (including most data types produced by ArcGIS). The software is available for free under a GNU General Public License from the website https://www.qgis.org. The algorithm is written in Python, which is the required language for algorithms in QGIS. The algorithm was developed using QGIS 2.18.7, and written in a variant of Python 2.7, called PyQGIS – the version of python used by this version of QGIS. More recent versions of QGIS have not been tested for stability, but hopefully will be tested in the future.

Currently the algorithm and user guide are held by Imperial College London and are available upon request. The source code is also available on GitHub, through user *nathanialcooper*, with project name *Resource-Mapping-Tool* or similar. Currently, the link to the repository is https://github.com/nathanialcooper/Resource-Mapping-Tool. The code is available under the same license as the QGIS software, the GNU General Public License, v3.

There may be updated versions of the Specifications & Functionality report, and the user guide there as well. There may also be a plugin released for QGIS in the future.

How to include and use the algorithm in QGIS is covered in the user guide.

To produce the visualisations indicated about (refinery location and material transport), a variety of input information is required.

1) A raster map (.tif) of the land use for the region of interest. This describes what all land is used for in the region, broken into categories. This can be selected from the list of currently loaded rasters, or from a file.

2) The No Data Value that the land use raster map (see 1) uses to indicate no information. This is used when analysing the map to extract the relevant areas of the land use raster map. This is a text entry that must be a number.

3) A list of the raster values that describe the land uses of interest for the land use raster map. Since the raster map describes how all land is used, only land uses that accord with the ultimate interest of the optimisation should be included. This will be a list of raster values that correspond to these land uses. This is a text entry, with entries separated by a comma with no spaces.

4) A raster map (.tif) of constraints on the land for the region of interest. This is different than land use, as some land which may be useful for the target purpose is unavailable for other reasons such as topology, for example. This can be selected from the list of currently loaded rasters, or from a file.

5) A raster map (up to 5) (.tif) of productive yield of the land for the region of interest, in units of tons dry mass per hectare per year. This provides information about how productive the land is for a given product. This is necessary for determining the expected productivity of the land. This can be selected from the list of currently loaded rasters, or from a file.

6) The name of the product associated with each of the productive yield maps provided. This should match what the name of the product is in the GAMS analysis, as it is used when generating the data sets which are later read into the GAMS analysis. This is a text entry.

7) A shapefile (.shp) or other vector file that outlines the region of interest. This a file that indicates where the algorithm should generate a cell grid and analyse the various provided maps. This can be selected from the list of currently loaded shapefiles, or from a file.

8) The rough desired number of elements in the grid that will be used to analyse the various data sets. The actual size of the grid will vary by 5% in either direction as getting the exact value is difficult. This is a text entry which must be a number.

9) The minimum relative size of cells to be counted in the final grid, as a percentage of the largest cell. Generated grids can have arbitrarily small cells when very small slivers of the region overlap with new grid cells. To prevent this phenomenon the tool can remove all cells smaller than a certain percentage of the largest cell. This is a text entry which must be a number.

10) The x-value fraction at which the lower break point for the linear estimators occurs. The linear estimators for biomass yield intercept with the real biomass distribution at some point. This is the lower value of where the cross should occur, as a fraction. This is a numerical entry.

11) The x-value fraction at which the upper break point for the linear estimators occurs. The linear estimators for biomass yield intercept with the real biomass distribution at some point. This is the upper value of where the cross should occur, as a fraction. This is a numerical entry.

The outputs of the algorithm are generally described above, but are listed below to provide slightly more detail.

1) A matrix containing cell ID numbers and the associated area of the cell in [km^2]. The cell ID is listed vertically in order first, and the cell area is listed second. This is saved as a .csv format.

2) A matrix containing the linear distance between the centroids of all cell pairs in [km]. This is provided as an *N+1* x *N+1* matrix, where *N* is the number of cells. The first column and the first row are cell ID numbers. The intersection of a given row and cell is the distance between those centroids. This distance is only provided for adjacent cells. Non-adjacent cells are set to a placeholder value. The diagonal of the matrix is all 0, as the distance between a cell centroid and itself is 0. This is saved as a .csv format.

3) A matrix containing the cell ID and actual usable (arable) land area of the cell in [km^2]. The cell ID is listed vertically in order first, and the usable land area is listed second. This is saved as a .csv format.

4) A matrix containing the tortuosity between the cells of cell pairs. This is provided as an *N+1* x *N+1* matrix, where *N* is the number of cells. The first column and the first row are cell ID numbers. The intersection of a given row and cell is the tortuosity between those centroids. This tortuosity is 1.4 for adjacent cells. Non-adjacent cells are set to 2.6. The diagonal of the matrix is all 0, as the tortuosity between a cell centroid and itself is 0. This is saved as a .csv format.

5) A matrix containing the cell ID and average bioyield of the cell in [tons dry mass / (month * km^2)]. The cell ID is listed vertically in order first, and the average bioyield is listed second. This is saved as a .csv format.

6) The shapefile of the cellular grid that has been used in the analysis and to generate the results. This is saved a .shp

7-9) The slope of the biomass yield approximations for the Low, Middle and High approximations (where Low, Middle and High correspond to the relative position on the x-axis. Low is for low x-values, etc). The cell ID is listed vertically in order first, and the slopes are listed afterwards. This is saved as a .csv.

10-12) The intercept of the biomass yield approximations for the Low, Middle and High approximations (where Low, Middle and High correspond to the relative position on the x-

axis. Low is for low x-values, etc). The cell ID is listed vertically in order first, and the slopes are listed afterwards. This is saved as a .csv.

Algorithm Methods

The algorithm is currently a script that is run from the Processing Toolbox of QGIS. The description of how to do this is provided in the User Guide. PyQGIS scripts are interpreted rather than compiled, so once the code has been inserted, it can be run immediately, provided the necessary inputs are available.

The program imports several key packages from QGIS including the core and utils libraries, as well as the PyQt4.core packages. It also imports some smaller packages for specialized processing, including the math, csv and time packages.

The program starts by initializing and copying relevant input variables for use later in the algorithm. This is to have all variables and parameters located in one place for ease of use. Following the declaration block, several functions are defined – these will be explained once they appear in the course of the main program.

Actual calculation in the program begins by reprojecting any layers which are in different projections to the projection used by the land use raster map. Built in algorithms employed by the program require all inputs to be in the same map projection to ensure accurate results. This is done for both vector shapefiles and for raster data, using the layerReproj and rastReproj functions respectively.

Now that all layers are in the same projection, the program goes on to apply the constraint map to the land use map. The constraint map indicates which land is unusable for the purpose of the analysis, and is made up of 0 (for not useable) and 1 (for usable). By multiplying the constraint map and land use map at each individual data point making up both data sets, the resulting map (a constrained land use map) will only have areas which are available for the optimisation. After the two data sets have been multiplied, the resulting data set raster is added to the project as certain algorithms only work if the raster is present in the project.

From here, the program begins to create the cellular grid that will be used to analyse data. This begins by estimating the square cell dimension necessary to create a grid with the requested number of cells, based on the size of the region of interest. The program calculates the initial cell dimension by assuming that a square grid of square cells covering the extent of the region of interest, with cell side lengths equal to the root of total extent area divided by the requested number of cells. The program then creates this initial cellular grid to work with based on this estimated cell dimension, using the gridGen function. This initial grid is then clipped using the shapefile for the region of interest so that only cells which actually intersect the region of interest are included. The next step is to assign cell ID numbers to each grid cell, and calculate the area of each cell. This is done using the replaceFields function, which adds cell ID and area as attribute fields to layer for easy use later.

This creates the initial grid, but this is not necessarily the final grid. The program goes on to count the number of features in the current grid which are smaller than the minimum feature size, using the countSmallFeatures function. The minimum feature size is determined

by the minimum-relative-size-of-features variable which is entered by the user. This variable indicates the minimum allowable fraction of cell area relative to the largest cell in the grid. Cells smaller than this will be removed from the final grid. The program subtracts the number of cells which are too small from the total number of grid cells. If this value is too far from the requested number of grid cells (default 5%), then a new grid must be generated.

To create the new grid with a greater number of cells, the process outlined in the previous two paragraphs is repeated with an ever increasing number of "requested" grid cells until the final grid has an adequate number of cells of sufficient size. This is done within a while loop, so that it can break once the grid is of sufficient size, or the iteration limit has been reached. An iteration counter is included to ensure a situation of vacillation between two values outside of the bounds does not occur. Once a grid that will have a sufficient number of features has been identified, the grid cells that are too small are actually removed from the cellular grid, leaving the final grid. The grid is then loaded to the project for use later in the project, and for the user to have access to it. The shapefile of the cellular grid is also saved to the location specified by the user.

Now that the final cellular grid has been completed, relevant data from it may be extracted. The program proceeds to write a .csv containing the cell ID and cell area of all cells in the final grid, saving it in the location indicated by the user. It does this using the writeAttributeTable function. This iterates through each cell in the grid, copies the cell ID and the previously calculated cell area from the attribute field, and writes this copied information to the .csv, line by line.

The program continues to produce spatial data about the generated grid, this time writing a .csv with the linear distance between the centroids of each grid cell to each other grid cell. This is done with the writeLinDist function. In this function, the grid cells are iterated through in a nested for loop, so that all grid cells are compared to all other grid cells. The cell IDs are then compared to see if it is the same cell being examined in both loops. If it is, the distance is set to 0. Next the grid cells are checked to determine if they are adjacent to one another. If they are, then the distance between centroids is calculated. If they are not adjacent, the distance is set to an arbitrarily high number. This distance is then written to the csv. The reason adjacency is checked is that in the optimisation, only adjacent cells can have material transport between them, so by setting the non-adjacent cells to have a large distance between them, the optimisation can more easily ignore those connections.

The next step in the program is to write a .csv with a tortuosity between each grid cell to each other grid cell. This is done with the writeTortuosity function. In this function the grid cells are iterated through in a nested for loop, so that all grid cells are compared to all other grid cells. The cell IDs are then compared to see if it is the same cell being examined in both loops. If it is, the tortuosity is set to 0. Next the grid cells are checked to determine if they are adjacent to one another. If they are, then the tortuosity between the cells is set to 1.4. If they are not adjacent, the tortuosity is set to 2.6. This tortuosity is then written to the csv. The reason adjacency is checked is that in the optimisation, only adjacent cells can have material transport between them, so by setting the non-adjacent cells to have a large tortuosity between them, the optimisation can more easily ignore those connections. The tortuosity between adjacent cells set to 1.4 is from spending time working with google maps, comparing the linear distance relative to the driving distance between the centroids of adjacent cells. Regardless of road density, the tortuosity (defined as driven distance divided by linear distance) was found to be

approximately 1.4, with some variation, but no noticeable trends. The decision was made to set all adjacent cells to have a tortuosity of 1.4, for consistency.

Now all major calculations regarding the spatial information of the grid have been performed, the next step is to start analysing the map data provided by the user. This first major map to analyse is the constrained land use raster map. This is done with using the gridToRast and gridStats functions. The algorithm used to perform statistics on the land use raster and count the amount of land available for use requires that the masking map (in this case the grid whose cells we want to aggregate information into) also be a raster. To this end, the program employs the gridToRast function to create a raster version of the grid, with the cell ID as the raster values. The rasterised version of the grid is then added to the project for use in the next section. After the grid has been rasterised, grid statistics can be performed to calculate the amount of available land for each grid cell. This is done in the gridStats function. The statistics algorithm employed returns a list of the total area for each land use within each grid cell. This list is then parsed into a vector in order of cell ID, where the total amount of land of the land uses of interest are summed from the list and added to this respective position in the vector. The final step is to write this vector of available land areas to a .csv. This is done with the writeVector function. The writeVector function saves a .csv to the location indicated by the user. The function writes the cell ID and the available land for use for that cell on each line.

The last steps are to analyse the biomass yield data sets provided by the user. This is done using the bioyield and writeBioyield functions. The only biomass yield values that matter are the yield of biomass in areas which are usable and available for use. Since the land use raster has already been multiplied by the constraint raster data set, only the land use areas which are available for use remain. Therefore, all the land of the specific type of interest which remains should be used to calculate the average biomass yield. To do this, a data set raster of booleans of equal size to the land use raster is created, where each data point is of the land use type of interest is 1, and all other data points are 0. This data set raster is then multiplied at each point by the biomass yield raster data set, yielding a data set raster of only the biomass yield values for areas where the land is usable and available.

Now this must be aggregated into the cellular grid cells to determine an average value for the biomass yield. This is done using the same algorithm as in the gridStats function, using the same rasterised version of the cellular grid. It aggregates all various biomass yield values and the area they make up per grid cell into a text file, which then must be parsed. This is parsed line by line, writing each biomass yield value and its respective coverage area to a vectors, which is in turn made up of vectors. The yield values and areas are appended to the vector whose position is equal to the cell ID that the information came from. Once all the information has been written to this vector, the average value of biomass yield is calculated per cell by iterating through each cell and summing the total biomass growing potential per cell, then dividing by the total usable land. This provides the average biomass yield across the whole cell, which is then written to a new vector, just containing the average biomass yield value.

Once all the information has been written to this vector, the average value of biomass yield is calculated per cell by iterating through each cell and summing the total biomass growing potential per cell, then dividing by the total usable land. This provides the average biomass yield across the whole cell, which is then written to a new vector, just containing the average biomass yield value.

To calculate the linear estimates for biomass yield, a vector of the stepwise cumulative sum of biomass produced from each land quality (starting with the best land and moving to worst land), and a vector of the stepwise cumulative sum of land used (in the same order) are calculated, for each cell. Each of these vectors are divided by the total amount of biomass produced, and total amount of land used, respectively, for each cell, to normalize the final values to 1. Next, the y-value associated with each breakpoint provided by the user is interpolated from the real data. These interpolated y-values, and the x-value breakpoints, are used to calculate slope and intercept of linear approximations which pass through them. The first linear estimate goes through (0,0) and (breakpoint_1,interp_y1), the second goes through (breakpoint_1,interp_y1) and (breakpoint_2,interp_y2), and the third goes through (breakpoint_2,interp_y2) and (1,1). This is done for all cells.

This process is done for all the biomass yield data sets that are provided by the user, up to a maximum of 5. The final step is to write this information to a .csv which is done with the writeBioyield function. This function takes the average biomass yield for all biomass types and writes it to a .csv sequentially in order of all average biomass yields for the first biomass type, then for the second biomass type and so on. The average biomass value is in units of tons dry mass per square kilometre per productive month (there are only 4 for the purposes of the optimisation). The yield is then written in the form of the name of the biomass, followed by the cell ID number, then the average biomass yield value repeated 4 times, ending with a 0 repeated 8 times. This provides the average biomass yield for each month of the year for the land, starting with the first month of productivity. This is saved where the user has indicated.

The program ends by removing the rasters and maps that it has added to the project with the exception of the cellular grid, as it is useful for the user to have visual feedback on the distribution of grid cells.