Distributed and Memory Efficient Machine Learning for Spatial Analysis Applications

Carlos Ribeiro

Instituto Superior Técnico

Abstract. In the context of spatial analysis, spatial disaggregation or spatial downscaling are processes by which information at a coarse spatial scale is translated to finer scales, while maintaining consistency with the original dataset. Fine grained descriptions of geographical information is a key resource in fields such as social-economic studies, urban and regional planning, transport planning, or environment impact analysis. However, spatial disaggregation involves heavily iterative, and computational demanding algorithms, seriously compromising performance when the data sets at play involve large geospatial regions and contains large volumes of information. This paper presents a M.Sc. thesis project that concerns with the development of a scalable and efficient approach to spatial disaggregation, taking advantage of state-of-art approaches for parallel and distributed computation, as well as methods for leveraging secondary memory.

1 Introduction

Spatial analysis includes a broad range of techniques for analyzing spatial data. Such techniques are applied in multiple fields, including biology, epidemiology, ethnology, geography and sociology. However, spatial information (e.g., social indicators or economic activities) is often presented at a relatively aggregated level. When in need of providing localized estimates, aggregated spatial data sets can be useless or even misleading, as potential critical spatial variations are diluted throughout the region been considered.

In the context of spatial analysis, spatial disaggregation techniques can be used as a solution to this problem. Such techniques may include classic methods of dasymetric mapping and pycnophylactic interpolation as well as state-of-the art regression analysis procedures leveraging ancillary data. Through them, aggregated spatial data can be transformed from a set of source zones into a set of target zones with different geometry and higher granularity.

The preferred methods to estimate the values of the missing target zones often involve linear or non-linear regression, depending on the relation of the source data and the ancillary information used to compute such regressions (i.e., depending where the target variable has a linear dependence on the covariates, or a more complex and non-linear relationship), however, impracticalities regarding regression analysis arise when data associated with large geographical regions is used, namely, high computational overhead and associated time penalties.

One of the most promising solutions for this problem is parallelization, aided by new hardware architectures such as multi-core processors and the corresponding software support. Memory-efficient storage of large data on disk can also be used to achieve improvements in terms of scalability.

In the context of my M.Sc. thesis, I will explore the development of efficient and scalable approaches for spatial downscaling. With basis on a disseveration-based procedure[1] previously developed by Monteiro (2016) in the R programing language, I aim to develop a new implementation, exploring state-of-the-art parallel and distributed programming paradigms, together with methods for leveraging secondary storage, to reach the desired level of scalability.

The rest of this paper is organized as follows: Section 2 presents fundamental concepts, while Section 3 describes related work. Section 4 details my proposal to implement computationally efficient spatial disaggregation mechanisms. Finally, Section 5 presents an overview on this report, together with the planing and scheduling for the next stages.

2 Fundamental Concepts

Spatial disaggregation methods rely on a multitude of areal interpolation techniques, i.e., processes were known data from a set of source zones are redistributed to another set of overlapping target zones, were the underlying values are unknown. Spatial attributes of the target zones, such as area and perimeter, are easily calculated. However, the estimation of nonspatial attributes is a challenge, due to the difficulty of estimating the unknown underlying spatial distribution of those attributes. The process of computing the target variables and their areal distributions can be performed according to various criteria, using underlying principles inherent to the data being reallocated (i.e., simple areal interpolation) or on the basis of available data from other sources, covering totally or partially the same area (i.e., intelligent areal interpolation). Understanding seminal intelligent areal interpolation is a key step towards the development of a scalable spatial disaggregation method.

The following sections describe commonly used statistical analysis methods for geospatial data, spatial disaggregation algorithms and relevant frameworks for big data analytics.

2.1 Statistical Analysis of Geospatial Data

Statistical analysis of geospatial data is a fundamental approach in understanding and extrapolating relevant information hidden in the raw data. Frequent statistical analysis include identifying statistically significant clusters or hot spots, uncovering geographic distributions, or finding patterns and relationships between different types of variables. This section discusses some of the commonly used methods in statistical analysis of geospatial data. Section 2.1.1 starts by

[1] `https://github.com/bgmartins/dissever`

enumerating some data structures used to manage these types of data sets, followed in Section 2.1.2 by a discussion about commonly used regression analysis algorithms. Section 2.1.3 explains an algorithm, named kernel density estimation, to estimate the probability density functions of random variables. Finally, in Sections 2.1.4 and 2.1.5, spatial interpolation strategies are discussed, namely inverse distance weighting interpolation and kriging respectively.

2.1.1 Data Models for Geospatial Data

In order to store and manage geospatial, data representations such as *rasters* and *shapefiles* are key factors to take into account. A *raster* is a data structure that divides a geographical region into rectangles, called *cells* or *pixels*, that can store one or more values associated with the region they represent.

The *raster* package for the R language, available in CRAN[2], has functions for creating, reading, manipulating, and writing raster data. The package also provides, among other things, general raster data manipulation functions, in order to support the development of more specific user-functions. Some of the core classes used by the *raster* package are the *RasterLayer*, *RasterBrick*, and *RasterStack* classes. The *RasterLayer* object represents a single-layer raster data whereas the *RasterBrick* and *RasterStack* objects represent multi-variable raster data sets. The principal difference between the last two is that a *RasterBrick* can only be linked to a single (i.e., multi-layer) file and the *RasterStack* can be formed from separate files and/or from a few layers bands from a single file.

Another widely used file format for storing geopatial data is the *ESRI Shapefile* format, commonly comprised of three files: the first file (*.shp) contains the geography of each shape, used to georeference the attribute values, the second file (*.shx) is an index file which contains record offsets. The third file (*.dbf) contains feature attributes with one record per feature.

Due to its popularity, several different R packages[3] provide functions for reading, writing, and manipulate shapefiles. Some examples are the *rgdal*[4], the *maptools*[5] and the *PBSmapping*[6] packages.

2.1.2 Regression Analysis

Regression analysis is a statistical process in which functional relationships are established between the ancillary data and the dependent values in what is called a *learning phase*. After this relationship is estimated, new values of the dependent variable can be predicted within some degree of confidence.

Various regression approaches can be used in spatial analysis problems such as interpolation. one simple example is the Ordinary Least Squares (OLS) method.

[2] `https://cran.r-project.org/web/packages/raster/index.html`
[3] `https://www.nceas.ucsb.edu/scicomp/usecases/ReadWriteESRIShapeFiles`
[4] `https://CRAN.R-project.org/package=rgdal`
[5] `https://CRAN.R-project.org/package=maptools`
[6] `https://CRAN.R-project.org/package=PBSmapping`

In the OLS method, linear relationships between the independent ancillary variables (i.e., preexisting land cover types like urban, suburban or forest, in the case of spatial disaggregation applications) and the dependent variable (i.e., population counts) are estimated. The general OLS approach can be expressed as follows:

$$y = \alpha + \left(\sum_{c=1}^{C} \beta_c x_c \right) \tag{1}$$

In the equation, y is the dependent variable, C comprises the total number of independent variables (i.e., ancillary variables), α is the intercept in the OLS model, x_c is the independent variable and β_c is the regression coefficient for that same independent variable c, computed in the learning phase.

In order to estimate the the regression coefficients β_c, and the constant of intercept α, in relation to the dependent variables, the minimization of the sum of the squared residuals must be computed, from where the following equations can be derived.

$$\hat{\beta} = \left(X^T X \right)^{-1} X^T y \tag{2}$$

In Equation 2, $\hat{\beta}$ denotes the maximum likelihood estimate of β, a vector containing all the β_c regression coefficients. The matrix X contains the set of independent observed predictor variables, and y is the vector containing the observed dependent or response variables.

$$\hat{\alpha} = \bar{y} - \left(\sum_{c=1}^{C} \hat{\beta}_c \bar{X}_c \right) \tag{3}$$

In Equation 3, $\hat{\alpha}$ is the estimate of the intercept constant α, $\bar{y}$ is the mean value of y and $\bar{X}_c$ is the mean value of X_c.

As in many regression based models, in order to maintain the mass-preserving property (i.e., the pycnophylactic property), the total estimated population associated to a source zone must be adjusted by multiplying the ratio of the original observed population to the new estimated value. Also, the computed correlation coefficients can be negative, resulting in negative population estimates, which can be seen as an inconsistency in this model.

Another example of a regression method is the decision tree approach, which is a type of non-linear regression where a continuous dependent variable is modeled by a step curve, based on observations relating the dependent and independent variables.

Regarding the mechanism for building a regression tree, the classification and regression tree (CART) algorithm is nowadays the most commonly used. In the CART algorithm it is easier to imagine the plane containing all observations as a multidimensional one, where the number of dimensions is defined by the number of independent variables x_j plus a dependent one y, where all data points (i.e.,

observations) can be expressed as pairs (x_{ij}, y_i). The first node of the regression tree requires a division in this plane. The algorithm tries every possible binary division of the original plane, selecting the split resulting in a smallest impurity measure. In this scenario the measure of impurity translates into how well the division was performed.

The CART algorithm usually relies on one of two splitting rules or impurity functions for a regression tree; (1) the Least Squares (LS) function and (2) the Least Absolute Deviation (LAD) function. An example the LS function can be expressed as the minimization of the sum of squares, in a multidimensional plane, expressed as:

$$\arg\min_{j,\,S} \sum_{i:x_{ij}>S} (\bar{y} - y_i)^2 + \sum_{i:x_{ij}\leq S} (\bar{y} - y_i)^2 \tag{4}$$

In the equation, after the minimization is performed, the values j and S which are respectively the selection of the dependent variable and the region of the plane (i.e., a value for x_{ij} where the binary division will take place) define this division. This splitting rule is then applied to the resulting subsets of observations, generating a new node for each division, until some user defined threshold regarding the depth of the graph is reached. Unnecessary depth in a regression tree results in a degradation of performance in the results.

The regression trees model can be extended into a more sophisticated regression method, through the combination of multiple trees the random forest (RF) algorithm is one such procedure. The first step in creating a random forest is the sampling procedure. In this step, k different subsets are randomly bootstrapped from the original dataset S, each one containing N records, resulting in a collection of k training subsets $S_{Train} = \{S_1, S_1, ..., S_k\}$.

For each sample S_i the observations that are not included by the sampling procedure (i.e., the *out-of-bag* or OOB data) are referenced in their own subset, one for each S_i, creating the collection:

$$S_{OOB} = \{OOB_1, OOB_1, ..., OOB_k\}$$

The OOB collection is later used to obtain an estimation of the error rate. After this data partition, for each subset S_i, an unpruned decision tree h_i is created, possibly in parallel, with a slightly deviation from the standard CART algorithm: for each S_i only m randomly selected predictors are taken into account in the tree growth process. This results in k regression trees (Breiman, 2001). In order to perform the regression analysis, the final result is simply computed by averaging all the predictions from all the regression trees into a single result. However, if the RF contains noisy decision trees, this average may deviate from the optimal result. Due to this concern a weighted average is proposed, in order to mitigate those effects, requiring a weight w_i calculated right after the training process (Chen et al., 2016). This can be done by calculating the mean squared error (MSE), computed by inputting the OOB_i observations in its corresponding trained tree h_i:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{5}$$

In the Equation, n is the number of observations in the OOB_i set, $\hat{y}_i$ is the estimate of h_i and, y_i is the true value of the observation.

Now, the weighted regression of the target variable X can be expressed as:

$$H_r(X) = \frac{1}{k}\sum_{i=1}^{k}[w_i \times h_i(x)] \tag{6}$$

2.1.3 Kernel Density Estimation

Kernel density estimation (KDE) is a technique used to estimate the probability density function of a target variable, based on observations (i.e., data points).

This is done by assigning a kernel, (i.e., one of multiple of curve functions, such as Gaussian, Quadratic or Cosine) as described in Equation 9, to each data point. This creates a surface based on the aggregation of all kernel curves. The next step is a normalization process of this density surface so that the integral equals 1, resulting into a estimate of the density probability of the target variable, as expressed in Equation 7.

A d-dimensional dataset (i.e., a multivariate dataset) with sample size n can be expressed in the following way:

$$X_i = \begin{bmatrix} X_{i1} \\ \vdots \\ X_{id} \end{bmatrix},\ i = 1, ..., n$$

In the case of geospatial coordinates, X would have two dimensions, defining the locations of the observation in the geographical plane (Wand and Jones, 1993).

The kernel density estimate of X can be expressed as:

$$\hat{f}(x; H) = \frac{1}{n}\sum_{i=1}^{n}\kappa_H(x - X_i) = \frac{1}{n}\sum_{i=1}^{n}\kappa_H(x_1 - X_{i1}, ...,\ x_d - X_{id}) \tag{7}$$

In the formula, x is a new data point, $\kappa(x)$ is the kernel function, and H is the bandwidth matrix $H = diag(h_1^2, ..., h_d^2)$ which defines the smoothness level at the surface of the estimated density function. The $\kappa_H(x)$ represents the kernel function leveraged smoothness factor H, expressed as:

$$\kappa_H(x) = |H|^{1/2}\,\kappa(H^{-1/2}x) \tag{8}$$

Finally the kernel function $\kappa(x)$ itself is defined by a curve, such as the Gaussian curve defined in the following Expression:

$$\kappa(x) = (2\pi)^{-d/2}\,exp\left(-\frac{1}{2}x^T x\right) \tag{9}$$

2.1.4 Inverse Distance Weighting Interpolation

Many times, information relative to geographical regions is presented as a set of arbitrary sample points, with different density throughout the geographical plane. When in need of providing information on points outside the original dataset, interpolation may be required. One technique to target this problem is known as inverse distance weighting interpolation (IDWI). IDWI is a deterministic and nonlinear interpolation technique. The core principle of IDWI is that any unknown point can be estimated by the weighted sum of nearby known observation. This weighting value is estimated accordingly to the distance of the known and unknown points. The reason to only take into account a nearby subset of sample data points is due to correlation maximization and computational overhead minimization. A simple inverse distance weighting function, as defined by Shepard (1968), can be expressed as:

$$W(x',\, x_i) = \frac{1}{d(x',\, x_i)^p} \tag{10}$$

In the Equation, x' denotes the point at which the unknown value is being estimated, x_i is a known point nearby x' and, $d(x',\, x_i)^p$ is the distance function between points, leverage by a positive real number p which regulates the weighting decay.

Given the weighting function $W()$, the prediction $\hat{f}(x')$ for an unmeasured location x' is given by:

$$\hat{f}(x') = \frac{\sum_{i=1}^{N} W(x',\, x_i)\, f(x_i)}{\sum_{i=1}^{N} W(x',\, x_i)} \tag{11}$$

In the Equation, $f(x_i)$ is the known value in location x_i and N is the number of nearby observation surrounding x'. As inverse distance weighting is a deterministic technique it only takes into account the density of the samples, disregarding the underlying spatial structure of the sample points. Also, as this procedure is preformed by computing the average of known observations, the estimated value being calculated can never be higher or lower then the maximum and minimum values presented in the observation set. This leads to poor estimates in locations where the observations don't accurately model the underlying spatial distribution of the target variable.

2.1.5 Kriging

Another important interpolation method is the kriging algorithm, also refereed to as Gaussian process regression. In the the kriging algorithm interpolated values are modeled by a Gaussian process. As in the IDWI method the kriging method assumes that the distance between sample points reflects a spatial correlation. However in this method, the weighting function also takes into account the spatial variation among observations. In this method, the prediction of an unmeasured location is estimated by having their surrounding observations contribution weighted in the following way:

$$\hat{f}(x') = \sum_{i=1}^{N} W(x',\, x_i)\, f(x_i) \tag{12}$$

In the Equation, $W(x',\, x_i)$ is not only based in the distance between the unknown and known point but also on the overall spatial variation among the measured points. In simple kriging (i.e., one variation of the kriging algorithm) $W(x',\, x_i)$ is obtained by the following equation system:

$$\begin{bmatrix} W(x',\, x_1) \\ \vdots \\ W(x',\, x_n) \end{bmatrix} = \begin{bmatrix} c(x_1,\, x_1) & \cdots & c(x_1,\, x_n) \\ \vdots & \ddots & \vdots \\ c(x_n,\, x_1) & \cdots & c(x_n,\, x_n) \end{bmatrix}^{-1} \times \begin{bmatrix} c(x_1,\, x_0) \\ \vdots \\ c(x_n,\, x_0) \end{bmatrix} \tag{13}$$

In the Equation, $c(x,\, y)$ is the known covariance between x and y. Other variations of the kriging algorithm include ordinary kriging, and kriging with a trend. While simple kriging assumes a constant mean throughout all observations in ordinary kriging, this notion is discarded in favor of a constant mean only in the local neighborhood of each estimation point. In the kriging with a trend variation, the assumption of a constant local mean is disregarded altogether in favor of a linear or non-linear fit on the data points.

The overall performance of the kriging algorithm is best when it is known to exist spatially correlated distance or directional bias in the data. It also helps to compensate for the effects of data clustering, as isolated data points have a greater contribution than clustered ones (Burrough et al., 2015).

2.2 Spatial Disaggregation Algorithms

Spatial disaggregation procedures, often referenced as spatial downscaling, rely upon a set techniques that can be used to transform data from a set of source zones into a set of target zones, with different geometry and with a higher general level of spatial resolution.

This section describes seminal spatial disaggregation procedures.

2.2.1 Simple Area Weighting

In order to desegregate spatial data, one of the simplest approaches is the area weighting method. This approach relies exclusively on the source data *a priori* distribution to reallocate data in source zones to target zones with different geometry and resolution, assuming that a given target variable y is uniformly distributed in each source zone.

Following this assumption, the estimated value of the target variable in the target zone can be estimated as follows:

$$y_t = \sum_{s} \frac{A_{st} . y_s}{A_s} \tag{14}$$

In the formula, y_t represents the estimated value of the target variable in the target zone t, y_s is the observed value of the target variable in source zone s, A_s is the area of source zone s, and A_{st} is the area of the intersection of the source and target zones.

This method satisfies the mass-preserving property, which requires that the sum of all the values estimated for the target zones contained in a given source zone must equal the original value of that same source zone. However, the overall accuracy of this method is limited, compared with other more complex techniques overviewed in the following sections.

2.2.2 Pycnophilatic Interpolation

Pycnophilatic interpolation, also referred to as pycnophylactic reallocation, is a spatial disaggregation method that solely relies on the source data itself and on some intrinsic properties of population distributions, such as the fact that people tend to congregate, leading to neighboring and adjacent places being similar regarding population density (Tobler, 1979).

Tobler's (1979) *pycnophilatic interpolation* method starts by generating a raster map (i.e., a grid composed of target zones) based on the original data set, as shown in Figure 1, where (1) represents the original data set composed of source zones, and where (2) represents the newly generated raster map. This transformation takes into account the property of mass-preserving areal weighting, where the sum of all the cells (i.e., target zones) representing a source zone must equal to the amount of that original source zone. The values of the grid cells are then smoothed, by replacing them with the average of their four neighbours, resulting in a raster such as the one from Example (3) in Figure 1. The predicted values in each source zone on the newly generated raster map are compared with the actual values from the original source zones, and adjusted to meet the pycnophylactic condition of mass-preservation. This is done by multiplying a weight with all the cells belonging to the source zone deviating from the original values. Several iterations of the previous referenced algorithm may be required, until a smooth surface is achieved. Starting from the replacing of the cells with the average of their four neighbours, the whole process can be repeated until some convergence criterion is met (i.e., the overall value of the cells remains fairly unmodified in two consecutive iterations).

2.2.3 Dasymetric Mapping

In order to improve the estimate of the internal structure of a spatial variable distribution, within source zones, correlated ancillary data may be used. The basic premise behind the dasymetric mapping method is the usage of additional information x in order to distribute y (Wright, 1936). Provided that the information in x and y is proportional and strongly correlated the method effectively discards the previously presented assumption of homogeneity inside each source

[7] `http://www.geog.ucsb.edu/~tobler/presentations/`

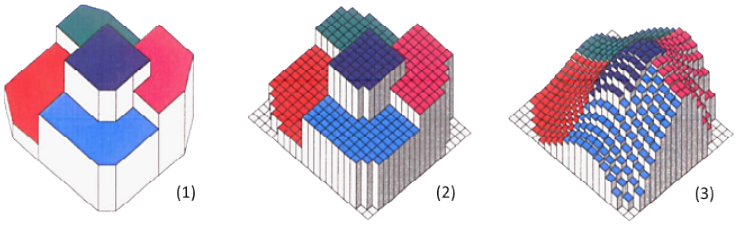

Fig. 1. Illustration of Tobler's pycnophylactic method for geographical regions.[7]

zone, in favor of the notion that data are proportional to the auxiliary information on any sub-region.

One of the simplest schemes for implementing dasymetric mapping is to use binary masks in a process, referred to as binary dasymetric mapping (Fisher and Langford, 1996). In this scenario, each source unit is divided into two binary sub-regions (i.e., land cover types: residential and nonresidential) and the source information is then allocated only to one of those sub-regions (e.g., populated residential areas). To better illustrate this case, based on Equation 14 a new expression can be reached:

$$y_t = \sum_s \frac{A_{tsr} \times y_s}{A_{sr}} \tag{15}$$

In the equation, y_t is the estimated population in the target zone t, y_s is the total population in source zone s, A_{sr} is the source zone area identified as residential, and A_{tsr} is the area of overlap between target zone t and source zone s, with a land cover type identified as residential.

One limitation of binary dasymetric mapping is the constrain of only being able to use a unique auxiliary variable. However, this method can be extended by the assignment of weights to n different categories of ancillary data, in a method known as polycategorical dasymetric mapping, in which the weights associated with each of the categories (i.e., land cover types) for the source area, represent the percentage of the target variable that is likely to be contained within that category, per source area. The difficulty in this approach is to devise an appropriate set of weights capable of modeling the real influence of each land cover type in the density distribution of the target variable. This can be done by either consulting a specialist on the field or by using regression analysis, as discussed in Section 3.1. In the context of his M.Sc. thesis, Monteiro (2016) presented an hybrid disaggregation procedure, combining some of the methods referenced in this section with a dissevertion procedure based on regression analysis. This method will be presented in detail in Section 3.3.

2.3 Frameworks for Big Data Analytics

This section explores some available frameworks suitable for extending the already referenced disseveration procedure in R, both inside an outside the R programing environment.

2.3.1 R Packages for Scalable Programing

R provides rich functionality and ease of use for data analysis and modeling, including functionalities for spatial data analysis. However is primarily used as a single threaded, single machine installation. R is not particularly scalable, nor does it support incremental processing, thus incurring in significant time penalties while computing large workloads. For instance, by cataloging where time is spent when running R programs, it is possible to identify some bottlenecks within the R kernel. More than 85% of total execution time for any given workloads in R is spent in stalls. Memory stalls alone account for nearly 75% of total execution time due to excessive use of main memory, resulting in R swapping to disk even when the input datasets are far smaller than the main memory size (Sridharan and Patel, 2014).

R's native support for efficient and scalable computation is limited, although there is a list of CRAN packages[8] that target some of the existing referenced limitations. This section describes some of the packages that might be useful to develop a scalable and efficient approach to aforementioned problem of spatial disaggregation.

Packages like ff[9], biglm[10] and bigmemory[11] (Kane et al., 2013) provide efficient memory manipulation by chunking the data and swapping it in and out of the main memory, when required. The bigmemory package, besides providing access to arbitrarily large data structures, potential larger than random access memory (RAM), also provides shared access across different processing cores, in order to support efficient parallel computing techniques. Nonetheless, attention must be made in the use of such packages, as using a disk storage layout different from R's internal structure representation can render most CRAN packages incompatible.

In regard to explicit parallel computing, starting from release 2.14.0, R includes a package called *parallel*, incorporating revised copies of CRAN packages *multicore* and *snow* (Simple Network of Workstations). The *parallel* package is designed to exploit a multi-core environment, offering a parallelized version of the *lapply* method, named *mclapply*, were it is possible to apply a given function f to an array x in a parallel fashion, obtaining *length(x)* results. This is done by creating multiple copies of the current R session, based on the Linux fork mechanism, being the master R session responsible for collecting the results from all worker sessions. For non-Linux users the alternative is the parLapply function,

[8] `https://cran.r-project.org/web/views/HighPerformanceComputing.html`
[9] `https://CRAN.R-project.org/package=ff`
[10] `https://CRAN.R-project.org/package=biglm`
[11] `https://CRAN.R-project.org/package=bigmemory`

which supports different platforms. However, a computing group (cluster) must be created first and used as argument in this function. The clustering mechanism in the *parallel* package is leverage by the integration of *snow*, exposing methods like *makePSOCKcluster* to create and manage a cluster of processes residing on different machines, or *makeForkCluster* which is equivalent to calling *mclapply* when given as argument to the *parLapply* function, creating the worker process by using the Unix system call *fork*. It should nonetheless be noted that, *parallel* does not include fault tolerance, although *snowFT*[12] provides an extension to the already mentioned *snow* package. This is done by implementing an alternative to the snow function *clusterApply*, called *clusterApplyFT*.

Regarding the *snowFT* package and some of the fault tolerance improvements: the master node's primary objective is to distribute computational tasks and collect the resulting outputs. This means that the master node spends most of the time waiting for the slave nodes to finish. The waiting time spent by the master node can be allocated to searching for failed nodes. Failure detection uses a function named *.PVM.pstats* which returns process status and allows to easily identify failed nodes. Upon failure node detection, the recovery procedure replaces it by a new created node. All initialization procedures must be performed prior to any computation.

Another option available in CRAN is associated with the project *Programming with Big Data in R* (pbdR), which is comprised of a set of R packages[13] for large scale, distributed computing and profiling, including high performance and high-level interfaces to MPI (Message Passing Interface, a communication protocol for programming parallel computers) and others. One interesting package from the *pbdR* repository, in regard to the problem at hand, is called *pbdMPI*. The package offers a a high-level interface to MPI, capable of handling linking issues, exposing a simple R interface for MPI programming intended for batch mode programming. In pbdMPI, programs are written in the *Single Program/-Multiple Data* or SPMD style, eliminating the need for master or manager nodes. Each process (MPI rank) gets to run the same copy of the program as every other process, but operates on its own data, making it easy to adapt existing R scripts to run in a parallel fashion.

Some other more complete solutions rely on frameworks that besides maintaining a high degree of R compatibility, focus on seamlessly integrating many other techniques of parallel and distributed computing, thus minimizing the development effort by the user. RABID is one such system. Implemented on top of the Spark framework (Zaharia et al., 2010), it uses distributed data structures that act like regular R data structures. RABID includes as well a serialization strategy that is transparent to users, building on top of the Renjin[14] R execution engine written in Java, thus effectively bridging Spark and the traditional user R environment (Lin et al., 2014).

[12] `https://CRAN.R-project.org/package=snowFT`
[13] `https://github.com/RBigData`
[14] `http://www.renjin.org/`

	multicore	cluster	GPU	fault-tolerance	external memory
parallel	Yes	Yes	No	No	-
snowFT	Yes	Yes	No	Yes	-
pbdMPI	Yes	Yes	No	No	-
RABID/spark	Yes	No	No	Yes	No
theano	Yes	Yes	Yes	No	No
tensorflow	Yes	Yes	Yes	Yes	Yes
spartan	Yes	Yes	No	Yes	Yes

Table 1. Comparison between possible tools for efficient and scalable programing

Presto is another R extension, presenting new language and runtime extensions to manage distributed and parallel executions. This is achieved with the introduction of distributed arrays, providing a shared in-memory view of multi-dimensional data stored across multiple machines (Venkataraman et al., 2012).

2.3.2 Other Popular Frameworks for Big Data Analytics

Regarding general support for efficient and scalable programing, some interesting options are also available, namely: Theano, TensorFlow and Spartan.

Theano[15] is a Python library designed to optimize and evaluate mathematical expressions involving multi-dimensional arrays efficiently (Theano Development Team, 2016). Theano combines aspects of a computer algebra system (CAS) with aspects of an optimizing compiler. It generates customized C code for many mathematical operations. This is particularly useful for tasks in which complicated mathematical expressions are evaluated repeatedly, and where evaluation speed is critical, also allowing the user, by modifying a configuration flag, to have them compiled in a highly optimized fashion to work either on CPUs or GPUs (the latter using the CUDA library)[16].

TensorFlow is an open source software library for numerical computation, whose core is developed in C++ (Abadi, 2016). In this environment, all the computations, shared states and operations on data are encapsulated in a data-flow graph where each node is mapped across multiple computation devices in a cluster, and within a machine across multiple computational devices, including multicore CPUs and general-purpose GPUs. The placement algorithm computes a feasible set of devices for each operation, calculates the sets of operations that

[15] `http://deeplearning.net/software/theano/index.html`
[16] `http://www.nvidia.com/object/cuda_home_new.html`

must be colocated, and selects a satisfying device for each colocation group. Each *edge* in this architecture carrys *tensors* (multi-dimensional arrays) between nodes representing the output from, or input to, a vertex. The TensorFlow API is composed of a set of Python modules that enable constructing and executing TensorFlow graphs. The tensorflow package also provides access to the complete TensorFlow API from within R[17].

Spartan is a distributed array framework, exposing a friendly development interface by implementing a small number of high-level parallel operators that encapsulate common patterns, efficiently expressing most types of computations (Huang et al., 2015). The Spartan framework supports array based logic to scale across multiple machines. This creates the issue of how to maximize the locality of accesses to array data spread out across the memory of many machines, also referred as *tiling*. One of the major contributions of the Spartan framework is in solving the *tiling* problematic automatically, via evaluation of the previously referenced high-level operators in an expression graph computed during the user program's execution. However, automatic *tiling* only aims to minimize network communications, ignoring other aspects such as how tiling impacts each machine's cache locality.

3 Related Work on Spatial Disaggregation

This section describes some relevant previous studies regarding spatial disaggregation techniques, on which this work will be based upon.

3.1 Spatial Disaggregation Leveraging Regression Analysis

Spatial disaggregation procedures based on dasymetric mapping techniques can be greatly improved by the introduction of a statistical perspective in order to quantify the functional relationship between the ancillary data (i.e., land-cover types) and the target variable. Regression is the preferred approach to quantify this relationship (Mennis, 2009).

Unlike traditional dasymetric mapping, referenced in Section 2.2.3, the target variable density is not only estimated by allocating the source information to the overlapping areas between land-cover types an the source area, but instead by a contribution of each land-cover type, weighted by a regression model.

The previously explained Equation 15 can now be extended as:

$$y_t = \sum_s \sum_c \frac{A_{stc}.y_s}{A_{sc}} = \sum_s \sum_c A_{stc}\, d_{sc} \tag{16}$$

In the Equation, A_{stc} is the area of interception between target zone t and source zone s, identified as land cover c, A_{sc} is the area of land cover type c in

[17] `https://rstudio.github.io/tensorflow/index.html`

the source zone s and d_{sc} is a regression based estimate for land cover type c in the source zone s.

In this context, basic regression models may have some disadvantages, such as the model not meeting the mass preserving pycnophylactic property or in some scenarios the regression equation predicting a negative population density. The first problem may be addressed by simply shifting the intercept regression parameter to the origin and the second one by scaling the estimated density values to fit the original values (Reibel and Agrawal, 2007).

3.2 Geographically Weighted Regression

In general, dasymetric models provide better results in comparison to simple statistical models. This is due to the fact that the discussed regression methods are fitted globally, whereas dasymetric estimates are locally fitted (Fisher and Langford, 1995). This is the case regarding population attributes as the relationship between land cover types and population density are nonstationary in most cases (i.e., varies from one geographical region to another). In this context geographically weighted regression (GWR) is presented as a solution to nonstationary relationship between ancillary and the target variables.

In order to take into account the different spatial relationships between the dependent and independent variables, multiple regression procedures can be performed in different geographical regions. This method is called Geographically Weighted Regression (GWR) (Brunsdon et al., 1998).

Very much like Equation 1 the estimate for the dependent variable given by the GWR method can be expressed by the following Equation:

$$y_t = \alpha_t + \left(\sum_{c=1}^{C} \beta_{ct} x_{tc} \right) \tag{17}$$

The difference from to standard OLS regression is that the regression coefficient for an independent variable c is not the same for the complete spatial geographical region. Instead, many β_{ct} are computed independently for different spatial subregions t, and the intercept constant α is now α_t referring only to its geographical subregion.

This approach requires computing estimates β for each predictive variable c (i.e., β_c) in each geographical region t, (i.e., the set β_{ct}). This is done only by considering data points near the location t. For sake of simplicity let us imagine that the region t is defined by the radius r and the β_c outputted by the OLS model, only takes into account observations within this circle, thus in fact representing the value β_{ct}.

If weights were to be assigned to all observations, this binary inclusion system, would compute them in the following way:

$$\alpha_{tk} = \begin{cases} 1 \text{ , if } d_{tk} < r \\ 0 \text{ , otherwise} \end{cases} \tag{18}$$

In the Equation 18, α_{tk} is the weight given to observation k in the region t and d_{tk} is the distance between observation k and the location t.

This is the underling notion of GWR. However, due to the unsatisfactory notion that observations in the farthest inner periphery region of the circle defined by the radius r have the same weight as observations in the center of the circle and, observations immediately outside of this region are not considered at all in the regression procedure an optimization referenced as kernel-weighting my be applied to the previously presented GWR model (Brunsdon et al., 1998).

Given the kernel-weighting optimization its possible to rewrite Equation 18 to accommodate a Gaussian distance decay function:

$$\alpha_{tk} = \exp\left(\frac{-d_{tk}^2}{2h^2}\right) \tag{19}$$

In this example the value of the weights associated with the observations decays gradually with distance. The parameter h provides control over the range of the circle of influence. Other distance-decay functions (also referenced as kernel functions) my be used in this model.

Having defined a weighting function, Equation 2 can now be presented in a new form:

$$\hat{\beta}_t = \left(X^T W_t X\right)^{-1} X^T W_t \, y \tag{20}$$

In the previous equation, W_t is the diagonal matrix expressed in the following manner:

$$W_t = \begin{bmatrix} \alpha_{t1} & 0 & \cdots & 0 \\ 0 & \alpha_{t2} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_{tN} \end{bmatrix} \tag{21}$$

In the definition for W_t, N is the number of observations and the diagonal elements correspond to the weights associated with the observations of the geographical region t. The output of Expression 20 is an array of equations, one for each region t. Each equation from this array outputs a vector of coefficients, one for each land cover type, effectively creating a matrix of coefficients β_{ct} required for the GWR model.

3.3 Hybrid Techniques for Spatial Disaggregation

Monteiro (2016) described the use of state-of-the-art regression analysis on top of classic methods of dasymetric mapping and pycnophylactic interpolation, effectively extending a previously proposed spatial downscaling algorithm called dissever (Malone et al., 2012) to perform spatial disaggregation. His approach has also been made available[18] as an open-source package for the R system.

The main logical steps used in this disaggregation algorithm are as follows:

[18] `https://github.com/bgmartins/dissever`

1. We start with a thematic map (i.e., a type of map designed to show a particular theme connected with a specific geographic area) of the aggregated information (e.g population counts) associated with the respective polygonal regions.
2. The data is redistributed using an iterative pycnophylactic-interpolation process in order to generate a smoothed surface to better represent the real distribution of the values, as previously explain in Section 2.1 (Tobler, 1979). The result of this process is used as an initial estimate, in a given target resolution.
3. All the ancillary data is represented in several other rasters, one for each type, normalized to the same target resolution.
4. A new raster is created based on an iterative regression analysis, using as input (1) ancillary correlated variables, sampled to diminish computational stress, and (2) the raster produced at Step 2 as an initial estimate (Malone et al., 2012).
5. The values returned by the downscaling procedure are proportionality readjusted, in order to maintain the mass-preserving property.
6. The estimates are again feeded to the regression analysis algorithm until some convergence criterion is reached, thus reaching a final solution.

For implementing the regression methods, Monteiro (2016) relied on the caret[19] package for R, which contains numerous tools for developing different types of predictive models facilitating the realization of experiments with different types of regression approaches.

One limitation referenced by the author is the impracticality associated with the time it takes to desegregate data belonging to large geographical regions. Sampling the data in the learning phase can mitigate this constrain, at the expense of creating a trade-off between statistical confidence in the results and the time it takes to produce those results. The proposal of the work here presented follows directly from this problem.

4 Thesis Proposal

My M.Sc. thesis proposal focuses extending the hybrid disaggregation procedure from Monteiro (2016), making use of secondary as well as parallel and distributed programming, in order to solve scalability constrains associated to the current approach.

4.1 An Efficient and Scalable Approach for Spatial Downscaling Based on Disseveration

The first step in this project requires a carefully revised profile of the disseveration algorithm referenced in Section 3.3, including performance tests regarding execution times, in order to create a benchmark for future comparison. This is

[19] `https://topepo.github.io/caret/`

aimed at identifying code sections that promise parallelism possibilities as well relevant data structures used in the main algorithm. After this analysis is performed, a multicore parallel implementation of the disseveration algorithm will be constructed, leveraging packages discussed in Section 2.3, such as the *parallel* package. All concurrent memory accesses constrains must be carefully accessed in order to prevent inconsistencies in the algorithm. This implementation must be probed by a second battery of performance tests, accessing the improvements of the referenced modification. The second main step in my proposal, and the more sensitive one, is the migration of the new parallel disseveration algorithm to a distributed environment, making use of the *snow* package, or using one of the general purpose scalable programing frameworks discussed in Section 2.3 such as *theano*, *tensorflow* or *spartan*. At this stage new performance tests will be conducted in order to access in which circumstances a distributed environment can be beneficial.

4.2 Datasets and Validation Plan

The online platform *A Vision of Britain through Time*[20] provides a historical database from the mid-nineteenth century to 1974, regarding the Great Britain territories, including census on several themes such as population densities, rates of growth, social class and socio-economic indexes, housing or land use, among many others. The content is provided in form of shapefiles, describing the digital boundaries, as well as *.csv* files containing the relevant statistical data, meaning that any census data can be joined to the exact spatial units that were used to publish them. Depending on the themes, the information is presented at the parish level, at the district level or at the town level.

The multitude of different information presented in this portal, the possible correlation between different themes, and the organization in which those data sets are presented, motivates their use as a test bed for future experiments related to this work. In the past, these data sets have already been used in studies related to spatial downscaling (Gregory, 2002).

Spatial disaggregation is never a error-free process, and it is vulnerable to error propagation. However, due to the different resolutions in which the datasets are presented, accuracy assessments may be performed by aggregating the target zone estimates to the source zones or some intermediary zones, and then compare the aggregated estimates against the original counts. Some statistical strategies of error assessment may include Root Mean Square Error (RMSE) between estimated and observed values, or the Mean Absolute Error (MAE). The RMSE and MAE are expressions are expressed as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} \tag{22}$$

$$MAE = \frac{\sum_{i=1}^{n}|\hat{y}_i - y_i|}{n} \tag{23}$$

[20] http://www.visionofbritain.org.uk/

In both Equations, $\hat{y}_i$ corresponds to a predicted value, y_i corresponds to the true value, and n is the number of predictions. It is important to use multiple error metrics in order to emphasizes more than one aspect of the model performance. While the MAE weights all errors in the same way, RMSE penalizes variance as it gives errors with larger absolute values more weight than errors with smaller absolute values. It should also be noted that, RMSE and MAE are not equivalent (i.e., one is not deductible from the other) (Willmott and Matsuura, 2005) Previous studies argued that the MAE is preferable to describe uniformly distributed errors. However, model errors are likely to have a normal distribution rather than uniform distribution, making the RMSE a better metric to present than the MAE (Chai and Draxler, 2014).

Assessments regarding execution time must be performed in order to evaluate code efficiency. Two R commands can be used to this end, namely: *proc.time* and *system.time*. The *proc.time* command works as a stop-watch. It requires a initialization at the starting time and then, by subtracting the starting time from the ending time in the bottom of a code sequence to be evaluated, the elapsed time is obtained.

The *system.time* command works as the previous one, however it takes the single R expression to be evaluated as its argument, avoiding the need to initialize and define variables.

Both commands output an object of class *proc_time* which is a numeric vector of length 5 containing counts for user, system, total elapsed times for the currently running R process and the cumulative sum of user and system times of any child processes spawned by it on which it has waited. Depending on the OS the definition of user and system times may vary. As a rule of thumb the user time is considered to be the CPU time charged for the execution of user instructions of the calling process. The system time is the CPU time charged for execution by the system on behalf of the calling process.

5 Conclusions and Planning for Next Stage

In this paper, I described my M.Sc. thesis proposal. The main objective in this work is to enhance the scalability of the disseveration algorithm used by Monteiro (2016), by making use of secondary memory as well as state-of-art parallel and distributed programming methods.

In the course of developing my M.Sc. thesis, spanning weeks 1 to 38, I plan to address the following tasks:

1. Data collection (Weeks 1 to 2)
2. Profiling of the disseveration algorithm. (Weeks 1 to 4)
3. Modification of the disseveration algorithm to support multi-core computation and secondary memory storage. (Weeks 5 to 14)
4. Performance results assessment. (Weeks 15 to 16)
5. Migration of the algorithm to a distributed environment. (Weeks 17 to 29)
6. Performance results assessment. (Weeks 30 to 35)
7. Writing the M.Sc. dissertation and a summarized report. (Weeks 1 to 38)

References

Abadi, M. (2016). Tensorflow: Learning functions at scale. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1).

Brunsdon, C., Fotheringham, S., and Charlton, M. (1998). Geographically weighted regression. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 47(3).

Burrough, P. A., McDonnell, R. A., McDonnell, R., and Lloyd, C. D. (2015). *Principles of geographical information systems*. Oxford University Press.

Chai, T. and Draxler, R. (2014). Root mean square error (rmse) or mean absolute error (mae)? *Geoscientific Model Development Discussions*, 7.

Chen, J., Li, K., Tang, Z., Bilal, K., Yu, S., Weng, C., and Li, K. (2016). A parallel random forest algorithm for big data in a Spark cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems*, PP(99).

Fisher, P. F. and Langford, M. (1995). Modelling the errors in areal interpolation between zonal systems by monte carlo simulation. *Environment and Planning A*, 27(2).

Fisher, P. F. and Langford, M. (1996). Modeling sensitivity to accuracy in classified imagery: A study of areal interpolation by dasymetric mapping. *The Professional Geographer*, 48(3).

Gregory, I. N. (2002). The accuracy of areal interpolation techniques: standardising 19th and 20th century census data to allow long-term comparisons. *Computers, environment and urban systems*, 26(4).

Huang, C.-C., Chen, Q., Wang, Z., Power, R., Ortiz, J., Li, J., and Xiao, Z. (2015). Spartan: A distributed array framework with smart tiling. In *Proceedings of the USENIX Annual Technical Conference*.

Kane, M. J., Emerson, J., and Weston, S. (2013). Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14).

Lin, H., Yang, S., and Midkiff, S. P. (2014). Rabid: A distributed parallel R for large datasets. In *proceedings of the IEEE International Congress on Big Data*.

Malone, B. P., McBratney, A. B., Minasny, B., and Wheeler, I. (2012). A general method for downscaling earth resource information. *Computers & Geosciences*, 41(2).

Mennis, J. (2009). Dasymetric mapping for estimating population in small areas. *Geography Compass*, 3(2).

Monteiro, J. (2016). Spatial disaggregation using geo-referenced social media data as ancillary information. Master's thesis, Instituto Superior Técnico.

Reibel, M. and Agrawal, A. (2007). Areal interpolation of population counts using pre-classified land cover data. *Population Research and Policy Review*, 26(5-6).

Shepard, D. (1968). A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*. ACM.

Sridharan, S. and Patel, J. M. (2014). Profiling R on a contemporary processor. *Proceedings of the VLDB Endowment*, 8(2).

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.

Tobler, W. R. (1979). Smooth pycnophylactic interpolation for geographical regions. *Journal of the American Statistical Association*, 74(367).

Venkataraman, S., Roy, I., AuYoung, A., and Schreiber, R. S. (2012). Using R for iterative and incremental processing. In *Proceedings of the USENIX Conference on Hot Topics in Cloud Ccomputing*.

Wand, M. and Jones, M. (1993). Comparison of smoothing parameterizations in bivariate kernel density estimation. *Journal of the American Statistical Association*, 88(422).

Willmott, C. J. and Matsuura, K. (2005). Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1).

Wright, J. K. (1936). A method of mapping densities of population: With cape cod as an example. *Geographical Review*, 26(1).

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: cluster computing with working sets. *Proceedings of the USENIX Conference on Hot Topics in Cloud Computing*.