parspatstat: An R Package for Large-Scale Spatial Analysis with Parallel Computing

Jonathan S. W. Lee[1], Reg J. Kulperger[1], and Hao Yu[1,2]
[1]The University of Western Ontario, London, ON, CANADA
[2]Corresponding author: Hao Yu, e-mail: hyu@stats.uwo.ca

Abstract

The **parspatstat** package is an extension of the **spatstat** package for spatial analysis. It implements some of the more common functions of **spatstat** in a parallel environment using the **Rmpi** implementation of the message passing interface (MPI) framework. Spatial descriptive statistics such as Ripley's K-function have wide applicability in spatial analysis but current implementations do not scale well for large data sets. Parallel computing (high performance computing) is one solution that can provide almost linear scalability to these applications. The usages of these functions are kept as similar as possible to the current **spatstat** function to aid in updating existing algorithms. Implementation, optimizations, and complications that arise from parallelizing existing algorithms are discussed.

Keywords: Spatial statistics, K-function, Parallel computing, Rmpi.

1 Introduction

A spatial point process is a model that aims to describe an observed point pattern. A point pattern is a collection of points, each of which has a spatial location in d dimensions attached to it.
A point pattern is observed within a spatial window. Often, the spatial window is rectangular or circular in shape for simplicity but can technically be any polygon. Complex polygon spatial windows arise in practice both naturally (geographic boundaries such as tree lines) and artificially (man made boundaries such as roads or political divisions).
Superficial characteristics of a point pattern such as clustering (points with a tendency to appear near each other) or regularity (points appearing far from each other, also known as inhibition) can be observed from visual inspection. These characteristics, as well as other less obvious characteristics, can be characterized through summary statistics and summary functions.

1.1 Spatial summary statistics

The most basic summary statistic is point intensity, defined as the expected number of points in a unit area of the point pattern. Going beyond a single numeric summary statistics are summary functions of a search radius. In a spatial point pattern, several descriptive statistics are often used to describe the point process characteristics. These include first-order statistics such as the empty space function F, the nearest-neighbour distance distribution function, G, and second-order statistics such as Ripley's K-function [2].

The F-function, known as the empty space function, denotes the probability that an area of radius r around a typical point contains at least one point. The G-function, known as the nearest neighbour function is the distribution of distances of a typical point to its nearest neighbours, not including the point itself. The J-function is a combination of the F- and G- functions. For a given point pattern, these functions can be compared to the corresponding theoretical functions for the homogenous Poisson process as an indicator of inhibition, clustering, or complete spatial randomness. The homogenous Poisson process is assumed to be the completely spatial random (CSR) case.

However, the aforementioned first order summary functions do not consider points beyond the nearest neighbour, and hence may be ignoring a lot of important information, so we can look at higher order summary functions. Ripley's K-function [7] is a second-order descriptive statistic that is commonly used to measure homogeneity of spatial point patterns. Under the assumptions of stationarity and isotropy, the function, $K(r)$, compares the expected number of points within a distance r of a typical point with how many points are actually observed within distance r, not including the point itself. Related to the K-function is the L-function which is a variance stabilized version of the K-function and has the added benefit of being easier to interpret visually.

1.2 Edge correction methods for spatial point processes

When computing the summary functions mentioned above, a search radius around a certain point is often examined. When a point is near the edge of the observation window, any part of the search circle outside the observation window will be censored, leading to a biased estimate of the statistic. As such, edge correction methods are applied and in fact, the K-function is independent of the shape of the study area when edge effects are corrected for properly [3]. The uncorrected estimate is known as the naive estimator.

Common edge correction methods include toroidal correction, where the edge on one side can be thought of as being wrapped around to the opposite edge. This only works for rectangular observation windows and only if the assumption of point behaviour on opposite sides of the window to be the same is met. Computationally for large data sets, the border method (minus sampling) is often used. Here, points near the edge are ignored for the purposes of the estimation but are still considered as neighbours for interior points. The disadvantage here is that much of your data may be ignored, especially for large search radiuses, r.

Perhaps the most popular edge correction is Ripley's original proposed edge correction, known as isotropic correction or simply Ripley's correction. Here, the naive K estimate is adjusted by scaling it by the ratio of the circumference of the search circle that lies outside the window to the circumference of the search circle that lies inside the window. Explicit formulas can be given for simple rectangular or circular windows [4] but can be difficult to compute for non-standard window shapes. Also, this correction assumes isotropy of points (our search area is a circle as opposed to an ellipse), hence the name isotropic correction. If the degree of anisotropy is known, one could first perform an appropriate projection of the points and use the same method.

Another common edge correction method is translation correction [5]. Similar to

the isotropic correction, the naive K estimate is adjusted by a scaling factor. The scaling factor is the proportion of the number of ways a pair of points x and y can be translated and still remain in the window. One advantage of the translation correction method is that it can be applied to windows of any shape, though it can be more computationally intensive for windows of non-standard shape.
The last two methods are computationally prohibitive for large data sets but provide us with an excellent opportunity to take advantage of parallel computing.

2 Parallel computation

For the past several years, computing power has plateaued due to physical constraints on the design of microchips which limits frequency scaling as a means of increasing computing power. Instead, parallel computing, which uses multiple processors to work concurrently to speed computation, has been the focus of research and development in recent years. From a statistical computing perspective, high-performance computing (HPC) that makes use of computer clusters consisting of thousands of cores is one of the primary tools to compute intensive simulations and calculations.
Two aspects of parallel computing are multithreading and multiple processing. The main difference between multithreading and multiple processing is the memory architecture. Multithreading involves dividing a single process into the work of multiple computing threads. Each of these threads share the same memory architecture. Multiple processing on the other hand divides a problem into smaller problems that are then worked on independently by separate processors. Each processor sees only it's own memory structure and if required, can exchange information with other processors. Multithreading provides a simpler method of parallelizing problems due to all threads working on the same data but is limited in the amount of speed-up possible. Multiple processing can scale almost linearly (ignoring communication overhead) with the number of processors used but is more complicated to implement. For the purposes of this paper, multiple processing is the primary focus. Currently in R, there are several packages such as **snow**, **doMPI**, **multicore**, and **foreach** that can ease parallel computations.

2.1 The *parspatstat* package

A package for the R statistical computing language [6] called **parspatstat**, has been implemented to extend some of the functions in **spatstat** [1] in a parallel setting by dividing the estimation into smaller portions while still maintaining proper edge correction where appropriate. It is built on top of **Rmpi** [8], the R implementation of the message passing interface (MPI) framework. In addition to computing traditional statistical summary functions in parallel, it can also speed up simulation of envelopes for model checking and comparison. In additional, there is support for the reading of large datasets in parallel to divide up a spatial point pattern into smaller patterns that can fit locally onto each worker.
The **Rmpi** and **spatstat** packages are required to be set up and installed on all computers that are to be included in the parallel environment (see respective documentation for instructions). First, one must either spawn the desired number of workers (slaves) or launch through a batch job scheduler with the desired number

of workers before invoking any of the parallel functions in **parspatstat**. Any data passed into the parallel functions of **parspatstat** are automatically propagated to all workers but any other packages or dependencies that the workers require will need to be loaded explicitly.

```
library(Rmpi)
mpi.spawn.Rslaves(nslaves=8) #spawn 8 workers
mpi.bcast.cmd(require(spatstat)) #have workers load spatstat library
```

2.2 Function usage

Names for parallel functions in **parspatstat** are the same as functions in **spatstat** but with a **par** preceding it. For example, the K and L functions, **Kest** and **Lest**, are **parKest** and **parLest**. For all functions, arguments are defined in the same order and passed directly to their **spatstat** counterparts. A few additional parameters specific to parallelization are required as well. One can refer to the package documentation for details but most notably is the **job.num** parameter which defines how many jobs to split the computation into whilst taking into account edge correction methods. This parameter should typically be at least twice the size of the number of workers available so that loads can be balanced to allow for a more even distribution of computation time between processors. Load balancing is further improved by sorting jobs by complexity and also by increasing the number of jobs. However, more jobs also require more communication overhead one needs to choose this parameter carefully.

2.3 Datasets that do not fit in memory

Occasionally we may encounter a dataset that is too large to fit into memory itself and hence needs to be read in sequentially in chunks. As an illustrating example, the lightning data set described in Section 3 is 400 megabytes and if we were working on a system with less than 400MB of RAM allocated to the R process, we can get workers to maintain portions of the dataset in chunks by passing a **parppp** object created using **parread.ppp** to **parKest** argument. For example, if the 400MB lightning datafile was called **lightning.csv** with a header indicating the x and y coordinates as the 5th and 6th column, the following would compute the K function estimate of the entire dataset across all years without ever having the entire dataset exist on a single machine.

```
ltg <- parread.ppp(file="lightning.csv", xy=c(5,6), chunksize=1000,
header=TRUE, localname="ltgppp")
K.all <- parKest(X=ltg)
```

The manager will read in a chuck of data of size **chunksize**, and send the appropriate rows of the data to each worker. The entire spatial window (user supplied or taken as the smallest bounding box of the data) is divided into strips either horizontally or vertically with the number of strips being equal to the number of workers. Each strip is also extended by the maximum search radius r to ensure that each worker has all the necessary data points to compute the K-estimate without having to communicate with neighbouring workers.

When chunking in data, load balancing is no longer supported since all workers will be required to have the entire dataset in its entirety between them. As such, chunking is beneficial when the original data cannot fit in memory of a single processor, whether it be a manager or a worker. Chunking also reduces the communication bandwidth required, but not necessarily the communication overhead.

3 Example: Ontario lightning data

A dataset of lightning strikes in the province of Ontario, Canada was obtained from the Ontario Ministry of Natural Resources. This data consists of 15.4 million lightning strikes from 1992-2010. In this example, an entirely enclosed square region is taken from the years with the least lightning strikes, 2003, and the year with many more lightning strikes, 2008. In each of these years, the number of data points in the defined region is too large for the **Kest** to compute efficiently if we wish to find edge corrected estimates. The following code snippet computes and plots the edge-corrected K-estimates using the **parKest** function assuming we have already initiated the workers as above. The resulting plots are given in Figure 1.

```
attach(lightning2003) #load the datasets
attach(lightning2008)
win <- owin(xrange=c(-84,-80), yrange=c(47,51)) #define obs. window
ppp.2003 <- ppp(x=lightning2003$V6, y=lightning2003$V5,window=win)
ppp.2008 <- ppp(x=lightning2008$V6, y=lightning2008$V5,window=win)
K.2003 <- parKest(X=ppp.2003, job.num=8*2)
K.2008 <- parKest(X=ppp.2008, job.num=8*2)
plot(K.2003)
plot(K.2008)
```

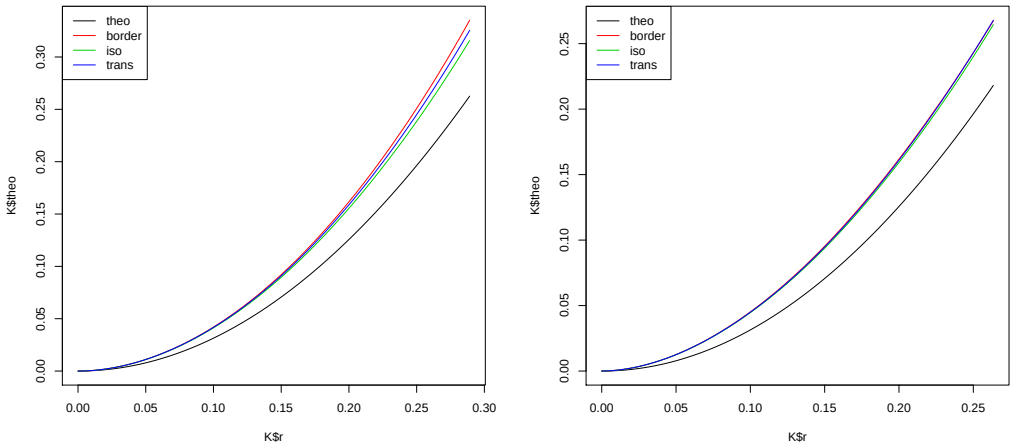

Figure 1: Plot of K-estimates of lightning strikes in a square region of Ontario in 2003 (left) and 2008 (right) using various border correction methods along with the theoretical line.

4 Conclusion

This short paper discussed some of the intensive computations that exist in the analysis of spatial point patterns and gave a short overview of the **parspatstat** R package. This package is able to take advantage of parallel computing using the **Rmpi** implementation of the **MPI** framework in order to speed up some of the functions in the popular **spatstat** package. In addition, it also makes possible the analysis of large spatial datasets that may not fit into memory. Both the function speed up and memory limitation increases almost linearly with the number of processors used in parallel.
As of writing, version 0.1-3 of **parspatstat** is available for download on CRAN. Future versions will add support for more functions.

References

[1] Baddeley, A. & Turner, R. (2005), "spatstat: An R Package for Analyzing Spatial Point Patterns." *The Journal of Statistical Software*, 12(6), 1-42.

[2] Baddeley, A., Kerscher, M., Schladitz, K. & Scott, B.T. (1999), "Estimating the J function without edge correction." *ArXiv Mathematics e-prints*, arXiv:math/9910011.

[3] Cressie, N. (1991), "Statistics for Spatial Data." New York, John Wiley & Sons.

[4] François, G. & Raphaël, P. (1999), "On explicit formulas of edge effect correction for Ripley's K-function." *Journal of Vegetation Science*, 10, 433-438.

[5] Osher, J. & Stoyan, D. (1987), "On the second-order and orientation analysis of planar stationary point processes." *Biometrical Journal*, 23: 523533. doi: 10.1002/bimj.4710230602.

[6] R Development Core Team (2010), "A language and environment for statistical computing." *R Foundation for Statistical Computing*, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

[7] Ripley, B. D. (1976), "The Second-Order Analysis of Stationary Point Process.", *Journal of Applied Probability*, 13, 255-266.

[8] Yu, H. (2002), "Rmpi: parallel statistical computing in R.", *R News*, 2(2), 10-14.