PrintPlacer: A Simple Application for Optimizing Printer Placement on a College Campus

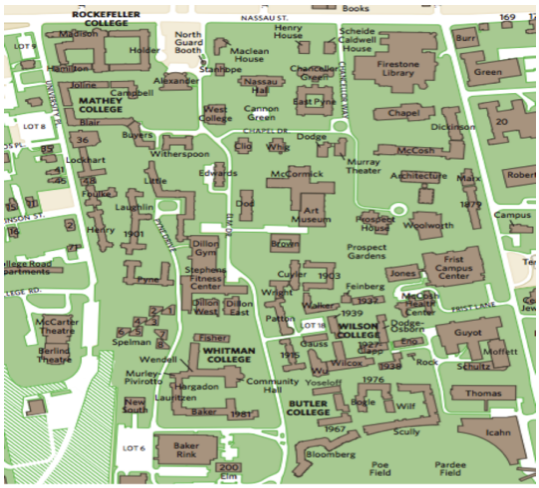
Gregory Jo, Rene Segura, Charles Peyser

January 14, 2014

1 Introduction

On any college campus with dormitories, there is a very population-dense environment in which students need to find printers to obtain study materials and write papers. It is very important for educational institutions to support the student body in educational ventures, and generally for institutions that serve a large amount of people to plan efficiently for the usage of their services. This type of policy would help with city-planning, especially in areas that have dynamic populations and also have outdated infrastructures.

The primary aim of this paper is to use there novel approach of using Monte Carlo simulation and regression techniques to find the optimal distribution of printers on the Princeton campus, and to serve as a proof of concept of this methodology. Therefore in this write-up, we focus on the algorithm we used for this project, and show that Monte Carlo simulation is an efficient way of finding the most optimal placement of multiple printers on the Princeton Campus as shown on this figure. Moreover, we attempt to generalize our findings from the data we found on the Monte Carlo simulation in the form of a regression analysis.



2 Related Work

In Monte Carlo simulation, it is very important that the randomness of the variables is preserved. This work shows some pitfalls in previous simulations where there were issues with the randomness of working with multiple variables in a Monte Carlo Simulation, such as the existence of pairwise correlation between the random independent variables (Ferrarini).

Moreover, for the Monte Carlo simulation in this project, it is necessary to place printers in random locations over the entirety of the Princeton campus. The authors here decided to use ordered random uniform variables for the independent variables, which is the definition of the beta distribution (McDonald and Xu).

For the regression methods used in the Monte Carlo simulations, we use this work on simulated annealing as a solution the traveling salesman problem as a general stencil (Brooks and Morgan). When we leverage this type of thinking to our problem of an optimal distribution of printers (an optimal placement of nodes) it becomes possible to leverage the Monte Carlo method by using repeated runs to measure the quality of a distribution, so that our results will provide an optimal solution within a certain confidence level.

3 Functionality

The purpose of this section is to acquaint the user with use of our program. The computation of our regression model is a three step process:

3.1 Generation of Raw Data

Substantial work is required to arrive at the formatted data required for the regression computation. In particular, the next step of data processing requires a comma-separated list of campus buildings which, at each line gives:

1) The building's name
2) The X-coordinate
3) The Y-coordinate
4) The number of print requests that stem from the building.

The second and third fields are populated by manual inspection of the buildings on a campus map; it is the fourth that is less obvious. For random data sets used in development and testing, this field was populated with arbitrary but reasonable values. For real printer data, however, this value can be approximated by, for each printer, equally distributing that printer's print count for a certain period of time to each building in the set of buildings which are closer to that printer than to any other printer.

3.2 Monte Carlo Generation of Undesirability Data

In order to compute a regression equation, we require a set of data in the form of random values of independent values together with some scalar "undesirability" metric that gives the quantity which we seek to minimize. We again choose the comma-separated value format, with in each line gives first "undesirability" of the printer distribution, and then the X- and Y-coordinates in turn of each printer in the distribution.

For and "undesirability" metric, we have chosen to use total print-distance summed over all buildings. The "print-distance" of a building is defined as the distance between that building and the nearest printer times the print request count of that building. It is meant to reflect the degree to which users located in buildings around the campus are inconvenienced in walking to pick up their prints.

The computations necessary to create a table of random data are done by 'createTable.py', which is a Python script which can be easily changed to create a table of arbitrary length and for an arbitrary number of printers. The script relies on the Python module 'campus.py', which defines the various data structures used in the computation.

3.3 Computation of Regression Equation

The final step of the computation is to use the randomly generated data to produce a regression equation. This task is accomplished by the printerRegression.m script, which uses a second degree model. That is, for dependent variable

y and independent variables $x_1, ..., x_n$, the model approximates

$$y \approx \alpha + \sum_{i=1}^{n} \beta_i x_i + \sum_{i=1}^{n} \gamma_i x_i^2 + \sum_{i=1}^{n} \sum_{j=1}^{n} \delta_{ij} x_i x_j$$

The script produces the optimized values of the obtained equation as list of X- and Y-coordinates for each printer.

4 Design

The purpose of this section is to familiarize the reader with the workings of those parts of the code that are not entirely obvious.

4.1 campus.py

The process of converting between a list of building locations and print requests to a set of data suitable for regression requires the encapsulation of data into a number of relevant types. For this reason, the campus.py module contains a number of Python class definitions which both allow the storage of information into convenient data structures and computation on those data structures. We define the following classes:

4.1.1 Buildings

A *Buildings* object encapsulates the data in the provided input, and provides the interface between this step of the computation and the one before it (see functionality section above). It contains four Python lists, which contain building names, X-Coordinates, Y-coordinates, and print requests respectively. It contains a constructor which populates the list from the appropriate csv file, as well as testing method that prints all four lists.

4.1.2 Printer

A *Printer* object encapsulates data relevant to a single printer on a college campus. It contains instance variables that contain its coordinates and the print-distance that has been assigned to the printer in the model. It also contains a *distTo()* method that computes the distance from the printer to some pair of coordinates - this method is used in the eventual allocation of print-distance to the printer. The constructor is simple; it stores coordinates, initializes print distance to zero, and stores the distance from the printer to the origin for eventual purposes of sorting.

4.1.3 Printers

A *Printers* object is essentially a list of individual *Printer* objects, and is meant to encapsulate all of the printers on a campus. The reason that this class must

be defined (instead of simply using Python's built-in list data structure) is that we must be able to easily sort a list of printers according to their distance to the origin. The reason for this is ensure that identical distributions of printers are not represented differently simply because of the order of the printers. For example, for the regression to be valid we could not have two printers located at (1,1) and (0.5, 0.5) represented differently based on which one was designated as "printer 1" and which as "printer 2".

To accomplish this sorting, a *sortByDistToOrigin()* method is included in the *Printers* class that uses a lambda of the *distToOrgin()* method in the *Printer* class to sort.

A *Printers* object is more than just an array of objects of the type *Printer*. It is a randomly assigned set of printers to coordinates that provides the basis for Monte Carlo generation of data. As such, it is construction involves two calls to Python's $random.uniform()$ function, which generates two pseudorandom numbers to act as coordinates.

4.1.4 Distribution

A *Distribution* object is meant to represent the way that printers are distributed among buildings. It thus contains a *Buildings* object and a *Printers* object. The *Buildings* object is provided to the constructor, while the *Printers* object is generated randomly.

The *Distribution* class contains two crucial instance methods. The first, *allocateBuildingsToPrinters*, does the job of populating the *printDist* property of the *Printer* objects that had been initialized to zero. It does this by iterating through each building in the included *Buildings* object, finding the closest *Printer* in the included *Printers* object, and computing print distance.

The second method, *totalPrintDist*, adds the print distances of all the printers and returns it. This value acts as the "undesirability" metric that we seek.

4.1.5 dataGenerator

The *dataGenerator* data type serves the purpose of putting together all the other classes defined in the module. It accepts the file produced in Step 1 of the computation as well as a desired number of printers and data points in its constructor. The *generateDataPoint* method is then called repeatedly to populate an instance variable list of lists called "data", where each secondary list constitutes one line in the Monte Carlo output. The *generateDataPoint* functions by initializing a new *Distribution* object and calling its *totalPrintDist()* method. The *getData* method returns the list of lists to the client code (in this case, the module *createTable.py*), which handles formatting into the csv file required for Step 3.

4.2 Regression and Plotting

Step 3 of the computation involves the use of a pair of MATLAB scripts, one for the regression computation and a second which creates a three-dimensional surface plot of the desirability of a single printer.

4.2.1 PrinterRegression.m

This script creates first and second degree terms for the model, and uses the *mvregress* function to compute the coefficients of the least-squares regression equation. It then uses the *fmincon* function to optimize the equation, producing the ideal locations for printers on the campus.

4.2.2 PrinterRegressionNoInteraction.m

As with PrinterRegression, but without interaction terms.

4.2.3 surfaceplot.m

This script creates a surface plot that demonstrates the effect of printer movement on undesirability.

5 Results

Trial runs were performed to test the regression and optimization functions being used to select optimal printer locations. This was done using the Princeton campus with random printer loads as shown in the attached "results" document. The functions were then tested to solve for optimal locations for two, three, four, and five printers. Three trials were done with different random samples for each number of printers.

The results given from these data trials can show us a variety of things. The minimization of the total printer distance, which was the dependent variable of our regression function, is achieved with the function created. The function is able to return optimal printer locations in a small amount of time. Trials were additionally run for 25 and 50 printers, and the function was able to return optimal printer locations.

For the case of a single printer, the regression and optimization functions work extremely well to find the optimal printer location. All trials return nearly identical locations, with slight changes due to the random sampling. For two printers, the same can practically be said. This begins to change with more than two printers, as there are more and more combinations of printers that can be optimal locations. Different trials begin to return different optimal locations. There are still some common locations that appear in the trials even though not all the points are the same.

The key issue is that the minimization values change according to the set of random data collected. Additionally, it seems to be the case that as the number of printers increases, the higher chance that the printer locations that minimize the regression function are found on the borders of the campus. This is most likely due to the fact that borders result in values of zero for all the interaction variables that include that printers x or y location. Initial trials with an alternate version of the regression function without interaction variables, called PrinterRegressionNoInteraction.m, were done with the same random data to find the optimal location for 5 printers. These data points were also included below. These optimal locations were less likely to have printers found on the borders of the campus, suggesting that what is stated before is possibly true.

Still, the case can be made that the initial points still create a good distribution of the printers on the campus map, especially with the given initial building data. Trials would need to be run with several campus styles to test whether the regression and optimization functions truly achieve optimal printer locations. Additionally, the case may be that the variable being used to determine optimality, total print distance, may not be the best way to determine optimal printer locations. Further studies would have to see whether any of these possibilities is correct.

6 Works Cited

[1] Ferrarini, A., 2011 A fitter use of Monte Carlo simulation in regression models, Computational Ecology and Software, 1(4): 240-243

[2] McDonald, James B., Xu, Yexiao J., A generalization of the beta distribution with applications, Journal of Econometrics, Volume 66, Issues 12, MarchApril 1995, Pages 133-152.

[3] Brooks, S.P., Morgan, B.J.T., Optimization Using Simulated Annealing, Journal of the Royal Statistical Society. Series D (The Statistician) , Vol. 44, No. 2 (1995) , pp. 241-257.