Visualization of Population, Income and Business Data in Phoenix

Hong Wang, Lei Chen, Megha Tiwari, Raghav Raja

**Abstract**—this project deals with visualizing the data related to population, income, crime and businesses in Phoenix. The different visualizations explored include choropleth maps, chernoff faces, parallel coordinate plots, time charts, mosaic plots and tag clouds. The project also explores the data extraction, data cleaning, data preprocessing and data analysis techniques that need to be done before visualization. The work also focuses on providing interaction to the visualization, making them dynamic with focus and context features. The goal of the project is to develop a system that can be used to explore how the crime, income and population of the region affect the businesses.

**Index Terms**—*Choropleth, Tag Cloud, Mosaic Plot, Chernoff faces, Yelp.*

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

—————————— ◆ ——————————

# 1 Introduction

V

isualization is defined as the use of computer supported, interactive visual representations of data to amplify cognition [1]. The process is subdivided into several stages. First, we extract the data needed for visualization. Then, we process the data. This data preprocessing and cleaning helps in transforming the raw data into a format that can be readily used for visualization. Once this is taken care of, we apply different visualizations on the data and analyze them to gain information about the data sets. The visualizations applied on the data sets are the Choropleth Map, Time Charts, Tag Cloud, Mosaic Plot, Chernoff faces and Parallel coordinate plot. The Choropleth Map allows the user to compare the population, crime, income and reviews among all the regions in and around Phoenix. The time charts are used to visualize how the crime affects the reviews of a restaurant in a particular place. The Parallel Coordinate plot gives an overview of crime, population, income, reviews and rating of a particular census tract and its comparison with the average and the median values for all the census tracts. The Chernoff faces can be used by the user to have an overview of all the data encoded on the face and maybe look for a particular kind of face on the map. The Mosaic plot can be used to determine the top 5 restaurants in the census tract and the tag cloud can be used to understand the overview of reviews of a particular restaurant.

# 2 Related Work

# 3 System

## 3.1 Data Collection

## 3.1.1 Crime Data

The crime data is obtained from spotcrime.com. The website doesn’t offer any API. We used a scraper to extract crime records from the source code of the html pages. Each original crime record contains the following columns:

<Type> <Date Time> <Address>

In order to overlay each crime onto the Phoenix map, we need to get the latitude and longitude of the crime. We used Google’s geocoding API to do the translation. After replacing the address with a latitude longitude coordinate, we use R to do a point-in-polygon operation to get the geoid in which the crime happened. The final crime record is in the following form.

<Type> <Date Time> <geoid>

## 3.1.2 Population Data

We obtained the 2012 population data of Phoenix from census.gov. The data is downloadable in CSV form through the FactFinder page in census.gov. We chose Census Tract as the geographical unit, as it is the smallest unit we can find on the website for 2012.

The data fields are as follows:

*GEO.id*

*GEO.id2*

*GEO.display-label*

*HD01\_VD01*

*HD02\_VD01*

For example:

*1400000US04013010101*

*4013010101*

*Census Tract 101.01, Maricopa County, Arizona*

*4344*

*594*

GEO.id and GEO.id2 both contain the geoids for each record. Geoids are used by the Census Bureau to identify different areas on the maps; it is used for all census data including shape files. Thus it is important to have it in order to incorporate the data with the shape files. The Geo.id field contains the full length id and it is what we are going to use. The Geo.display-label field contains the Tract number which is what we are going to use to label our map. The HD01\_VD01 field stores the actual population in each Census Tract, and HD02\_VD01 is the margin of error of the population estimation. Thus the actual population in one census tract should be Population ± margin of error. In our example above, the population of Census Tract 101.01 is 4344 ± 594. However, in our visualization, we may need to ignore the margin of error for consideration of simplicity.

## 3.1.3 Income Data

The income data for year 2012 can also be obtained from census.gov. We used median household income because it is less affected by the outliers if any. Similarly, the population data, the income data uses Census Tract as the geographical tracking unit.

The income data contains many fields, but we will narrow it down to the following:

*GEO.id*

*HC01\_EST\_VC13*

*HC01\_MOE\_VC13*

*HC01\_EST\_VC15*

*HC01\_MOE\_VC15*

For example:

*1400000US04013010101*

*98211*

*10779*

*135719*

*22881*

As mentioned before, GEO.id field stores the full length geoids for each record. The HC01\_EST\_VC13 and HC01\_MOE\_VC13 fields store the median household income and its margin of error, respectively. The HC01\_EST\_VC15 and HC01\_MOE\_VC15 fields store the average household income and its margin of error, respectively. We will ignore the margin of error for consideration of simplicity as well.

## 3.1.4 Yelp Data

We used Yelp API v2.0 to extract the data of the restaurants in Phoenix. The API keys were obtained and they were used along with the search term, “restaurant” and location. Being a Restful API, Yelp API enables calls that have a URL and search parameters embedded. Data received from Yelp API is in JSON format. So, it has to be processed by a JSON parser to convert to a CSV format.

The data obtained from Yelp is also filtered to remove any unnecessary fields. After filtering, the name of the business, ID of the business, review count, rating, address location and category of the business are restored.

The records are stored in the following format:

*<Name>*

*<ID>*

*<Review Count>*

*<Rating>*

*<Address>*

*<Category>*

For example:

*LosDosMolinos*

*los-dos-molinos-phoenix*

*204*

*3.5*

*8646 S Central Ave Phoenix AZ 85042*

*Mexican*

Also, for the most reviewed restaurants, we are also interested in the trend of their review numbers. Since the revenue of a restaurant is confidential, we have to use another method to estimate it. Since there is a connection between review number, customer flow and revenue, we use the review numbers per week and the average rating to represent how the business of a restaurant goes. The records are stored in the following format:

*<ID>*

*<Review#>*

*<Review Date>*

*<Useful Review Votes>*

For example:

*WIcDFpHEnC3ihNmS7-6-ZA*

*17*

*2011-02-11*

*1*

## 3.2 Data Analysis

We analyzed the temporal and geographical distribution of crime and review data. On one hand, we found that the crime data has very consistent temporal and geographical patterns. The crimes happened mostly in the Phoenix downtown and business areas. The trend of crime numbers of months in 2013 is almost the same as ones in 2012. This is also true for day of month, weekdays, and even hours of a day. On the other hand, review data show no clear pattern. When a new restaurant started, it may cause the review number of its area to increase. Throughout the year, the review numbers for each month are relatively stable compared to crime data. And our conclusion is that there is no clear connection between crime and review data.

We also did an analysis on population, income and crime. The result is also not clear. There seems to be a very weak connection between income and crime, but it’s so weak that it can be neglected.

As a result, we chose not to tell a story, but offer a visualization tool and let users to explore the data sets, and try to find interesting facts by themselves.

## 3.3 System Design

After a brainstorming session, we decided to use choropleth map as a base and add some visualization graphs on top of that to offer an insight to the details.

First step is the choropleth map. We use a dropdown box to let the user choose one of the 4 data sets to plot on the choropleth map. For each data set, we use a dropdown box to let the user choose the classification method, equal interval or quantile interval. For the time range selection, a double ends slider bar is used. All these components are located at the left top part of the screen.

Next step is Chernoff faces. We will draw a face for each of the census tracts on the choropleth map. Since the Chernoff faces may cause some distraction, we used a checkbox to let the user to enable or disable.

After a time range is selected, the first graph to be updated is the time chart graph. The beginning and end point are decided by the left and right bar on the time slider. The two lines on the graph are the crimes and reviews. The purpose of this time chart graph is to compare the trends, rather than the absolute values. So the two lines are both scaled on the y-axis to fit into one graph.

After a census tract is selected, the parallel coordinates plot is used to show its 6 variables: population, income, crimes, violent crimes, reviews and average rating. There are 3 lines on the parallel coordinates plot, current selection, average and median. The average and median line are calculated based on all the geoids. The mosaic plot is also corresponded to the census tract selection. It will list the top 5 most reviewed restaurants in a census tract. We choose 5 because a higher number may make each one too small and hard to be selected, and people are more likely to be interested in the popular restaurants. We use vertical split by reviews first, and then horizontal split by ratings second. For the ratings from 1 to 5, we use sequential color scheme, from light to dark. If a restaurant has a small number of reviews and therefore a very narrow width, we use a popup text to show its name.

The final step is the tag cloud. Once a restaurant is selected in the mosaic plot, in the tag cloud we will show the top 20 most used words by reviewers. The size of each word is decided by its frequency; the higher the bigger.  
Choropleth map being at the center, all other components are placed around to offer an intuitive control flow.

## 3.4 Data Preprocessing

To make the data look-up and update simple and fast, instead of using a database, we choose to do the data preprocessing and create several search tables.

The first table uses geoid as its key. When the user selected a geoid, its population, income, crimes, v-crimes, reviews and average rating are needed to be updated. Therefore, for each month in the 2 year timespan, we calculated the above 6 values and saved them in the table. For a lookup operation, we only need to read and add up all the months within the selected time range. To get the average rating, instead of saving the rating itself, we saved the number of stars. It’s because the average rating for each month may have a different weight due to its different number of reviews. Having the total stars, we can get the average rating by divided by the number of reviews, which is saved as well.

The second table is businessID based. For each restaurant, it has a unique id assigned by Yelp. Using that id as a key, we create a table for mosaic and tag cloud graph which contains two sets of data. First set of data is the total stars for each month which is similar to the one in the 2nd table. Second set of data is the top 20 most used words in the review for each month.

The third table is for the time chart plot, and is time based. Using each day as a row and each geoid as a column, we calculate and save the crimes and reviews.

Tag Cloud required the information to be extracted the Yelp API and the Yelp website. The restaurant information that is used in the project is readily provided by the Yelp API v2.0. The problem with the API, however, is that it returns only 20 values at a time for particular latitude and longitude values. Hence, a Java Program that iteratively loops by sending requests to the API is used, having the southwest and northeast boundaries of Phoenix. The information returned is in a JSON format, where each string corresponds to each restaurant. Later, a JSON parser program was used to convert this JSON file into a .csv file. When this was done, only the fields of interest were kept and the other fields were discarded. The street addresses of all the restaurants present in this .csv file were later converted into latitude and longitude values and geoids to represent them on the map. The Google API was used to do it.

The next step was to extract the reviews for all the restaurants present in the .csv file. The Yelp API does not have any support for doing this and just returns the top reviews for a place but not all the reviews for a place. Hence, we had to write a Scraper program to send requests to each page and extract the data from the page, retaining the business id, the rating of the review, the date of the review and the review content. The program returned the reviews that were stored in a text file. These reviews extended from 2005 to 2014. Later, this file was cleaned to only have the values of reviews in 2012 and 2013. The total reviews amounted to close to a hundred thousand spread over 4000 restaurants.

These reviews were then grouped by a particular business id for each month using a Java program, which resulted in reducing the number of rows in the data to at most the number of businesses rather than the number of reviews. Since, the tag cloud takes the inputs as business id and time period, having the reviews grouped based on the business id would make the work easier. Also, this program grouped the reviews by month and spread across the 24 months. Now, this review content is run through a word count program, which gave word counts of all the reviews per business per month. In this program, the prepositions and other commonly used words that don’t need to be part of the tag cloud are removed. Now, the data is stored in <word>\_<count> format in a decreasing order for each business per month. These files are then combined into a huge table with business id and the 24 columns, each representing the most frequent words for that particular month. This file is converted into .js file and is read into the JavaScript file that is used to draw the tag cloud.

## 3.5 Task Breakdown

Our tasks were initially broken down into 4 parts:

1) Time selectable choropleth map and time chart

2) Chernoff face

3) Parallel coordinate plot and mosaic plot

4) Tag Cloud

Task 1 was assigned to Hong. Task 2 was assigned to Leo. Task 3 was assigned to Megha. Task 4 was assigned to Raghav.

## 3.6 Implementation

## 3.6.1 Time Selectable Choropleth Map

The choropleth map needs to give an overview of population, income, crime number and review count in the Phoenix area. It also must allow the user to zoom in and zoom out.

To simplify the task, we chose to use Leaflet Library, which is an open source javascript library that allows us to create the map with relative ease.

We downloaded the shapefile for Arizona from census.gov and then joined the shapefile with the table we created, which contains the population, income, crime, and review number for each geoid. We then converted the shapefile to geojson format, and loaded it into the library. The library automatically creates tiles based on the geojson file.

The library makes it very easy to add interactions to the map. So we created an info panel which displays information of a census tract whenever the user hovers her mouse over the corresponding tile. Also whenever the user clicks on a tile, the geoid of the corresponding census tract will be sent to other modules for synchronization.

Afterwards, we implemented a menu selection which allows the user to select among the four data sets. Each dataset has a different color. We also allow the user to choose between equal interval and quantile color schemes. Both equal and quantile schemes use 7 bins, and they have to use a sequential color scheme. Equal interval color scheme works very well for population and income data, not for crime and review count data, because for both crime and review count data, the value in one census tract is much larger than the rest, which drags up the scale and makes most of the census tract fall into the lower levels of the interval. By switching to quantile color scheme, the problem no longer exists.

To enable time selection, we decided to use a slider bar. We created a range slider using the Jquery library. The user is able to move the two slides on the slide bar left or right to select certain time range. We will then calculate the sum of crime number and review count in each census tract over that selected time range. The map will then update the color of each tile based on the calculated sum and current selected color scheme.

However, there’s one problem we encountered during the implementation. Initially, we calculated the sum of crime number and review count for each census tract over the full time range, and then use the results to calculate the intervals in the legend for both equal and quantile color scheme. So the distribution of colors was based on that initial legend. The problem occurs when the user selects a very small time range, for example, one month, then the initial legend no longer work very well. Because the sum over the selected time range is significantly smaller than the sum over the full time range, most of the census tract will fall into the lower levels of the interval, making the tiles non-uniformly distributed again. So we realized we must update the legend dynamically when the user moves the slides in order to fix the problem. But at the same time, we also noticed the fixed legend is pretty useful too because it allows user to easily see how the data changes over time. So finally we decided to give the user an option to choose between fixed and dynamic legend. When fixed legend option is selected, the legend will not change when user move the slider in the slide bar, and the user can notice the color of the tiles in the map will gradually become lighter as she narrow down the slide bar; when dynamic legend option is selected, the legend will update dynamically as the user move the slides in the slide bar, and the color of tiles will remain uniformly

## 3.6.2 Time Chart

The time chart will need show the change of crime number and review count over certain period of time for the selected census tract. So the time chart must be synchronized with the mouse down event in the map. Each time when the user click on a tile on the map, the time chart will update based on the information in the corresponding census tract. The time chart also must be synchronized with the time selection. Whenever the user moves the slider, the time chart will update based on the selected time range.

Inside the time chart, the red line represents the crime number and green line represents the review count. We used a library called Rickshaw, which is built on top of d3, to plot the lines. We build a table which contains the daily crime numbers and review counts for every geoid. Based on the user selection described above, we then dynamically extract a portion of the data from the table and feed it into the library to draw the lines.

However, it turns out even if we use only a portion of the data in the table, there will still be too many data points to plot, making the graph extremely cluttered. It’s because we used the daily data, so the number of data points to plot for each geoid will range from 30 - 731. To fix the problem, we decide to take only 20 sample points from the selected data, and the value on each sample point will be the average of data within the intervals of these sample points. Also the length of intervals between the sample points will adjust dynamically based on the selected time range, to make sure there will only be 20 sample points to plot.

By doing this, each sample points will no longer have its original value, but the trend of the data will be maintained, and that’s what we want to keep. We want the user to be able the compare the trend of crime number and review count to find out if there’s any correlation between those two.

Because the magnitude of crime number and review count is very different, so we used two separate y-axises for them. The left y-axis is for crime number and the right one is for review count. The scale of both y-axises are calculated based on the min and max value of the corresponding data. Linear scale is used for all axises here. Also, in order to make sure both crime and review lines are always present in the graph ( except when the value on the selected sample points are 0 ), the scale of both y-axis will be adjusted dynamically based on user selection.

## 3.6.3 Chernoff Faces

We create canvas objects in html and use javascript to draw faces on them. Then we need to add canvas objects on top of the map. There are several ways to do so, and we chose to use leaflet library’s api to add a custom layer on top of the map. When the user changes the zoom level of the map, the Chernoff faces need to scale accordingly. We tried to get the zoom level and use a linear equation to get the needed size. However, we found that the zoom level is not linear itself. Therefore, a fixed search table is used to decide the size of faces under certain zoom level.

For each census tract, it has 6 values that can be encoded by Chernoff faces: population, income, crimes, violent crimes, reviews and average rating. The encoding strategy we used is face size, face color, eye size, eyebrow angle, mouth size, mouth curveness respectively. We didn’t use nose which is also a common encoding element because it is difficult to draw a nice-looking nose in javascript and when the face is small ,which is true for most of the time, it is hard to see the nose.

During the presentation, most of the feedback is given to Chernoff faces. One good suggestion is a legend to show the encoding, and it’s has been added now. Another question is that if a census tract has both high crimes number and average rating, its face should not look happy. After some discussion and research, our conclusion is that Chernoff faces do not necessarily need to have connections between data and human feeling.

## 3.6.4 Parallel Coordinate Plot

Parallel coordinates plot is to show the relative position of the selected census tract in the range of all. Additionally, average and median are calculated and added to the plot as well. We chose to use javascript to draw the plot directly instead of using the d3 library. The reason is that our needs are fairly simple and the parallel coordinates offered in d3 offers unneeded features that makes it slower. We added a canvas object in html on which we use javascript to draw the plot. The parallel plot is updated when the selection of time range or census tract is changed.

## 3.6.5 Mosiac Plot

Similar to parallel coordinates plot, the Mosaic plot is also implemented using javascript. The rectangle area is first divided vertically by each restaurant’s total review number, and then horizontally by 5-star rating number. When a restaurant has only a small number of total reviews, its name becomes too small to read. We use popup name to solve this problem. After the presentation, we feel that a legend is always needed for user to read the plot, therefore a simple legend is added to each color rectangle to indicate the rating number it represents.

## 3.6.6 Tag Cloud

The first method that we used to create the tag cloud was to process the data on the go from the review contents for a selected business and time interval and this took a lot of time to render. Hence, we had to fall back on creating this huge table described above and then convert it into a .js file. The inputs to this cloud are from the time slider and the selection of the restaurant from the mosaic plot. Both of them generate an onClick event that calls this tag cloud function. The parameters passed are the business id, startIndex and the stopIndex of the months.

A function initially calculates the top twenty words according to the given parameters aggregating all the word counts according to that particular time frame and return the most frequently used words in a decreasing order to another function called the draw() function, which will eventually draw the tag cloud on the screen. The function that calculates the top 20 words is constructed using a Hash table, that is actually used to store all the words used in that particular time range from the .js file and sort them in the descending order. Hash table has the words as the key and the count as the value. The frequency of the word actually is encoded in the size of the word of the tag cloud. Size is a pre-attentive cue that can easily be noticed by the viewer. The tag cloud function is similar to the function available on d3.js but the function on d3.js doesn’t encode any visual variables into the tag cloud. A few changes had to be made to that function to encode the frequency of the word to the font size. Then, the tag cloud is drawn printing out each word perpendicular to the next starting from the center of the screen.

## 3.7 Integration

After all the modules are done, it’s time to put them together. We used a 1024 x 768 space, and every module is scaled down to fit into the space. We initially thought about using tabs, but later decided it’s better to put everything in the same space because it allows more robust interactions.

Our design strictly follows the mantra: “overview first, then zoom in and filter, details on demand.” So the control flow of our program starts from the choropleth map. The choropleth map gives an overview of the data, also allow the user to zoom in and filter. We give user an option to enable or disable the chernoff face on the map. The chernoff face also gives a very good overview of the data, since it encodes all data sets in one face. The user can use the chernoff face to gain a general impression of each census tract and can roughly tell which one have more favorable status than the others.

If the user wants more details about one particular census tract, she can click on it, then all the other modules will be synchronized to visualize information for that particular census tract: the parallel coordinate plot gives a comparison of each data sets; the time chart compares the trend of change for crime number and review count; the mosaic plot lists the 5 restaurants with the most reviews, along with their ratings; lastly, the tag cloud will show the most common used words, all for the selected census tract. Additional interactions are implemented in the mosaic plot so that whenever the user clicks one restaurant, the tag cloud will be synchronized to display the most commonly used words for reviews on that particular restaurant.

As discussed in previous sections, we also allows time selection aside from the area selection, and it is realized by the slide bar. So all the modules in the program, including the choropleth map, are synchronized with the slide bar, namely each one of them will be updated in real time based on the selected time range.

One interesting thing to point out is that we found it was not so easy to put the chernoff faces on the map, because the faces were drawn by html canvas, separately from the library we use to create the map. Luckily we found that the leaflet library allow us to overlay html divs on the map. So we created a series of divs containing the chernoff faces and then put them on the map one by one. We then hard coded all zoom-in and zoom-out features for the faces. It is not a very efficient way to do this, so we could list this as one of the tasks we need to do for future improvements.

## 3.8 Test & Debug

We have encountered various bugs during the implementation, most of them involve scaling. We have mentioned that we hard coded the zoom in and zoom out feature for the chernoff faces. We did it merely by getting the zoom value of the map and use it to control the size of the face. However, during the implementation, we realized the map didn’t use a linear scale for the zoom value. So we end up having to hardcode the size of face for each particular zoom value using a look-up table. Also for the time chart, we tried to use the d3 library to control the scaling of y-axis. We’ve spent quite some time trying to figure out how the scaling function works in the library.

There are also various bugs regarding the control event that we had to fix.

# 4 Results

# 5 Conclusion and Future Work

**Acknowledgment**

**References**

1. J.S. Bridle, “Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition,” *Neurocomputing—Algorithms, Architectures and Applications,* F. Fogelman-Soulie and J. Herault, eds., NATO ASI Series F68, Berlin: Springer-Verlag, pp. 227-236, 1989. (Book style with paper title and editor)
2. W.-K. Chen, *Linear Networks and Systems.* Belmont, Calif.: Wadsworth, pp. 123-135, 1993. (Book style)
3. H. Poor, “A Hypertext History of Multiuser Dimensions,” *MUD History,* http://www.ccs.neu.edu/home/pb/mud-history.html. 1986. (URL link \*include year)
4. K. Elissa, “An Overview of Decision Theory,"unpublished. (Unpublished manuscript)
5. R. Nicole, "The Last Word on Decision Theory," *J. Computer Vision,* submitted for publication. (Pending publication)
6. C. J. Kaufman, Rocky Mountain Research Laboratories, Boulder, Colo., personal communication, 1992. (Personal communication)
7. D.S. Coming and O.G. Staadt, "Velocity-Aligned Discrete Oriented Polytopes for Dynamic Collision Detection," *IEEE Trans. Visualization and Computer Graphics*, vol. 14,  no. 1,  pp. 1-12,  Jan/Feb  2008, doi:10.1109/TVCG.2007.70405. (IEEE Transactions )
8. S.P. Bingulac, “On the Compatibility of Adaptive Controllers,” *Proc. Fourth Ann. Allerton Conf. Circuits and Systems Theory*, pp. 8-16, 1994. (Conference proceedings)
9. H. Goto, Y. Hasegawa, and M. Tanaka, “Efficient Scheduling Focusing on the Duality of MPL Representation,” *Proc. IEEE Symp. Computational Intelligence in Scheduling* *(SCIS ’07)*, pp. 57-64, Apr. 2007, doi:10.1109/SCIS.2007.367670. (Conference proceedings)
10. J. Williams, “Narrow-Band Analyzer,” PhD dissertation, Dept. of Electrical Eng., Harvard Univ., Cambridge, Mass., 1993. (Thesis or dissertation)
11. E.E. Reber, R.L. Michell, and C.J. Carter, “Oxygen Absorption in the Earth’s Atmosphere,” Technical Report TR-0200 (420-46)-3, Aerospace Corp., Los Angeles, Calif., Nov. 1988. (Technical report with report number)
12. L. Hubert and P. Arabie, “Comparing Partitions,” *J. Classification,* vol. 2, no. 4, pp. 193-218, Apr. 1985. (Journal or magazine citation)
13. R.J. Vidmar, “On the Use of Atmospheric Plasmas as Electromagnetic Reflectors,” *IEEE Trans. Plasma Science*, vol. 21, no. 3, pp. 876-880, available at http://www.halcyon.com/pub/journals/21ps03-vidmar, Aug. 1992. (URL for Transaction, journal, or magzine)
14. J.M.P. Martinez, R.B. Llavori, M.J.A. Cabo, and T.B. Pedersen, "Integrating Data Warehouses with Web Data: A Survey," *IEEE Trans. Knowledge and Data Eng.*, preprint, 21 Dec. 2007, doi:10.1109/TKDE.2007.190746.(PrePrint)

**First A. Author** Allbiographies should be limited to one paragraph consisting of the following: sequentially ordered list of degrees, including years achieved; sequentially ordered places of employ concluding with current employment; association with any official journals or conferences; major professional and/or academic achievements, i.e., best paper awards, research grants, etc.; any publication information (number of papers and titles of books published); current research interests; association with any professional associations. Author membership information, e.g., is a member of the IEEE and the IEEE Computer Society, if applicable, is noted at the end of the biography.

**Second B. Author Jr.** biography appears here.

**Third C. Author** biography appears here.