Data Debugging Report

Dimitar Gochev

September 11, 2012

1 Introduction

Software bugs are a well-known problem, and substantial research effort has been expended developing tools which encourage the writing of correct programs. However, most programs operate on data supplied by users, and even a perfectly good program would produce the wrong results if it is given incorrect input. Error checking of inputs is generally limited to ensuring the correct data type and formatting as required by the program. However, data is important and there should be better ways of checking its correctness.

One could imagine a tool which automatically audits data, similar to the ones which check for typographical errors in text. However, numerical data is much more unstructured than text. There are no rules, such as grammar, governing how numerical data should look, and no predefined set of allowable values such as the set of words in a language. There may, however, exist some ways of narrowing the scope of the problem to make it more manageable.

If we look at how data is most commonly stored nowadays, we see that it is in spreadsheets, such as Excel files or databases, and this can provide some degree of structure for approaching the problem. For that reason, we attempted to create a tool which would automatically detect numerical errors in spreadsheets. We have developed a Microsoft Excel extension, or add-in, written in C#, which uses cross-validation techniques and perturbation analysis to look for numerical values that appear suspicious. Values that appear to be potential errors are highlighted proportionally to the likeliness that they are incorrect, as judged by our analysis – the more likely a value is to be wrong, the brighter the highlighting.

2 Motivation and Related Work

The most commonly used software for editing spreadsheets, Microsoft Excel, has close to no built-in error checking. Aside from some type verification, the user simply has to be careful in their work, which is frequently not the case [15].

Spreadsheet languages allow users to specify their own mechanisms for error checking, which can be effective if done properly. Users can define assertions with filters or IF statements making sure values contained in cells or regions fit certain criteria. Combined with the AVERAGE and STDEV (standard deviation) functions, one is able to remove the outliers in a range by checking if their values fall within x standard deviations from the average. Such approaches to error reduction are rarely put to use, however, and automatic auditing methods are desired.

As a result of this necessity, significant efforts have been spent by researchers trying to address the issue of spreadsheet validity. However, the multitude of usage scenarios and the absence of any predefined structure in spreadsheets contribute greatly to the difficulty of this problem. Because users are free to arrange spreadsheet data in any way they please (vertically, horizontally, or in any arbitrary fashion), creating a tool that would work in the general case is a major undertaking. While per-company auditing solutions can be developed with relative ease because a standardized format can be agreed upon from

the start, adopting such standards on a large scale is near impossible. Nevertheless there have been proposed general methodologies for spreadsheet organization [10]. In the same way regular programming can be improved by paradigms such as object-orientation and modularity, spreadsheet development can also benefit from similar ideas. However, as is the case with other types of programming, following these principles typically reduces errors, but does not eliminate them altogether. Also, while they may help individuals learn how to manage spreadsheets in ways that will reduce the likelihood of errors, these strategies are not widely used and therefore do not provide any significant assistance in the creation of automatic auditing tools. Some work has also gone into classifying the types of errors that may appear in spreadsheets. While this taxonomy is useful in pointing out the kinds of errors that may need to be addressed and how often they show up in practice, the ways in which they can be dealt with is not established [12].

Some semi-automatic tools have been created which require that users specify the range of cells they want to audit, and possibly include additional ranges to be used for reference [4]. While this is in fact a useful tool which is able to extract relationships between cells which may not necessarily be numeric, unassisted discovery of regions is the natural step forward towards automatic error detection [6]. This is what our approach aims to achieve when dealing with numerical values.

It is worth noting that errors in computational logic are close to impossible to pinpoint because one cannot know the user's intent.

3 Approach

3.1 Building a Dependence Graph

A single spreadsheet typically contains multiple different sets of data, and each of them will have its own properties. For instance, the range of numerical values may differ greatly between sets within the same spreadsheet. This is important because when we are looking for errors, values have to be analyzed within their proper context, and values from different datasets will not be logically compatible. Therefore in order to be able to detect errors, one has to start by identifying the contiguous sets of data within the spreadsheet. Usually a person is able to do this with relative ease by looking at the visual layout of a spreadsheet and using clues such as names of columns or rows to break apart the spreadsheet into its composing sets. However, spreadsheets provide an extremely flexible way of storing data. There are hardly any organizational restrictions to the user and this lack of constraints makes it quite difficult to automatically extract the sets of related data by examining the layout. Even though many spreadsheets are laid out in some structured way, there are certainly some that are counterintuitive, and the variability among layouts is immense.

Regardless of how a spreadsheet is laid out, however, its elements remain the same. That is, all the data items, graphs, and formulas can be moved around without affecting the content. The underlying logic and interrelationships between these components is what is important. Therefore we can abstract away the visual layout of a spreadsheet and represent its structure as a dependence graph where the vertices will be cells, graphs, and other spreadsheet elements, and the edges will represent the directed flow of information between the nodes. [For instance, if the cell B1 contains the formula f(x)=A1+A2, the dependence graph for that spreadsheet would look as shown in Figure 1.]

[Figure 1]

Constructing a dependence graph requires that we extract all dependencies between the elements of a spreadsheet. We have to examine every component and see if it relies on any other piece of data in the spreadsheet. For charts this is fairly easy because their inputs can be accessed through the API. This is not the case with formulas, so references to other cells or ranges have to be extracted from the formulas text via parsing. This is done through the use of regular expressions, recognizing range references first, and stripping them out before moving on to recognizing single cell references. Naturally, there is no use

in constructing such a graph for spreadsheets with no underlying dependencies because there will be no edges between the vertices.

3.2 Perturbation Analysis / Sensitivity Analysis / Cross-Validation

Constructing the dependence graph is useful for two main reasons. First, the process of extracting dependencies can provide very useful information about the datasets in the spreadsheet without needing complex layout analysis. That is because whenever we identify a reference to a range of cells in a charts inputs or a formula, we know that the cells in that range belong to the same dataset. In other words the creator of the spreadsheet has already identified this range as containing contiguous data, and its cells are treated with no distinction by the chart or formula. As mentioned earlier, identifying these sets of contiguous data is of great importance because it provides the necessary context for further analysis.

The second important benefit of the dependence graph is that it allows us to pinpoint the spreadsheet elements that are most likely to be important to the user. These are the end results of the underlying computations. By looking for nodes in the graph that do not feed into any other nodes we can identify the components of the spreadsheet most likely to be the final outputs desired by the user.

Having knowledge knowledge of the datasets in a spreadsheet enables the use of traditional statistical methods for outlier detection. However, these methods typically require that the datas distribution is known, which is generally not the case. There are also some parametric methods which are independent of the distribution, but they are less sensitive.

Instead of employing statistical methods, we put to use our additional knowledge of the outputs in the spreadsheet. We are able to perform perturbation (sensitivity) analysis via leave-one-out cross-validation and measure the influence of cells on the final outputs. The cross-validation process works by sequentially replacing a value in the range with every other value in the range and computing the cumulative effect on the output. This allows us to calculate the importance, or influence, of that particular value on the value of the output. The effect on the output is calculated by summing the absolute difference from the original after each substitution:

$$\left| \frac{output_{old} - output_{new}}{output_{old}} \right|$$

An important benefit of this type of analysis is that we do not have to understand the underlying logic in the spreadsheet. Instead we simply perturb the inputs and look at the outputs. This makes it possible to analyze spreadsheets containing user-defined macros and functions.

The perturbation analysis is less clear when dealing with outputs which are not numeric. In the case of text outputs in cells, the change is treated as a boolean. That is, if perturbing a value causes a change in the text output, we add 1 to the total influence; otherwise it does not change. In the case of visual outputs, such as charts, it is more difficult to measure the effect. For certain kinds of charts, such as bar graphs or pie charts, we are inclined to expect to see equally distributed data (bars and wedges of approximately the same size). Therefore in those cases we measure the perturbation effect by the amount the result deviates from equilibrium.

4 Limitations

Our approach relies on the presence of formulas or charts to establish some relationships between the data. However, we observe that in the spreadsheet corpus, which contains a collection of near 5000 spreadsheets from various sources, about 40% of them contain formulas, giving reason to believe that formulas are commonplace and we can expect to be able to rely on them [5].

References

[1] Aurigemma, S., Panko, R.R. The Detection of Human Spreadsheet Errors by Humans versus Inspection (Auditing) Software. CoRR, 2010

[2] Ayalew, Y., Clermont, M., Mittermier, R. Detecting errors in spreadsheets, Proceedings of the European Spreadsheet Risks Interest Group Annual Conference University of Greenwich, London, 2000, pp. 51-62.

[3] Brown, P. S., Gould, J. D. An Experimental Study of People Creating Spreadsheets, ACM Transactions on Office Information Systems (5:3) July 1987, pp. 258-272.

[4] Crivat, B., MacLennan, J. (2004). Detect Anomalies in Excel Spreadsheets. 'ACCESS VB SQL Advisor Magazine'.

[5] Fisher II, M., Rothermel, G. The EUSES Spreadsheet Corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. Proceedings of the 1st Workshop on End-User Software Engineering, pages 47-51, St. Louis, MO, USA, May 2005.

[6] Fisher II, Marc, et al. Scaling a Dataflow Testing Methodology to the Multiparadigm World of Commercial Spreadsheets. ISSRE 2006: 13-22

[7] Hellman, Z. 'Breaking Out of the Cell: On The Benefits of a New Spreadsheet User-Interaction Paradigm'. Proceedings of CoRR. 2008.

[8] Jafry, Y.,Sidoroff, F.,Chi, R. 2006. 'A Computational Framework for the Near Elimination of Spreadsheet Risk'

[9] Morochove, R. Tech at Work: Spreadsheet Slipups Cause Financial Errors, PCWorld, 2006

[10] Panko, R. What we know about spreadsheet errors. Journal of End User Computing: Special issue on Scaling Up End User Development, 10(2):15-21, Spring 1998.

[11] Panko R. R (2008) Revisiting the Panko-Halverson Taxonomy of Spreadsheet Errors http://arxiv.org/abs/0809.3613 , EuSpRIG 2008 Annual Conference, In pursuit of Spreadsheet Excellence, 199-220

[12] Rajalingham, K. (2005, July). "A Revised Classification of Spreadsheet Errors," Proceedings of the 2005, European Spreadsheet Risks Interest Group, EuSpRIG 2005, Greenwich, London, 185-199.

[13] Rothermel, G., et al. (1998), 'What You See Is What You Test: A Methodology for Testing Formbased Visual programs', 20th International Conference on Software Engineering, 198-207.

[14] Rothermel, Gregg (2001). 'A Methodology for Testing Spreadsheets', ACM Transactions on Software Engineering and Methodology, 10 (1), 110 - 47.

[15] Wailgum, T. Eight of the Worst Spreadsheet Blunders, CIO, 2007.