Generierung von visuell unterschiedliche 2D Datensätzen mit ähnlichen statistischen Eigenschaften

Algorithm Engineering 2023 Project Paper

Eric Kaufmann
Friedrich Schiller University Jena
Germany
eric.kaufmann@uni-jena.de

Benjamin Schneg
Friedrich Schiller University Jena
Germany
benjamin.schneg@uni-jena.de

ABSTRACT

The five-finger pattern [1]:

(1) **Topic and background:** What topic does the paper deal with? What is the point of departure for your research? Why are you studying this now?
(2) **Focus:** What is your research question? What are you studying precisely?
(3) **Method:** What did you do?
(4) **Key findings:** What did you discover?
(5) **Conclusions or implications:** What do these findings mean? What broader issues do they speak to?

KEYWORDS

entity resolution, data cleansing, programming contest

1 INTRODUCTION

In der Statistik ist die Analyse von Datensätzen maßgeblich. Bei besonders großen bzw. komplex strukturierten Datenmengen ist es häufig schwierig die Daten sinnvoll zu visualisieren. Demnach wird der Datensatz nur von Eigenschaften, wie dem Mittelwert oder der Standardabweichung, klassifiziert. Dabei können Datensätze, die sich visuell kaum ähnlich sind, dennoch nahezu identische statistische Eigenschaften aufweisen.

1.1 Background

Um Datensätze zu generieren, die bis auf einen bestimmten Fehler, die gleichen statistischen Eigenschaften aufweist, wird der Ansatz des Simmulated Annealing verwendet. Die Idee ist dabei iterativ ein Problem näher an seine Optimallösung zu bringen. Es wird dabei akzeptiert, dass man am Anfang größere Fehler zulässt, diese Toleranz (Temperatur) jedoch über die Interationen nachlässt. Das heißt, das nach jeder Iteration die erlaubte Veränderung gedämpft wird, sodass man exakter die Lösung approximieren kann.

1.2 Related Work

Die Idee basiert auf auf der Veröffentlichung *Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing* von Matejka und Fitzmaurice. Dort wurde ein zweidimensaionaler Datensätze mittels Simmulated Annealing auf eine Zielstruktur transformiert. Sowohl der initiale als auch der transformierte Datensatz wiesen dabei einen ähnlichen Mittelwert, eine ähnliche Standwardabweichung und ähnlichen Korrelationskoeffizient auf. Ähnlich wurde dort mit einer Gleichheit auf von bis zu zwei Nachkommastellen definiert.

1.3 Our Contributions

In dieser Veröffentlichen wurde nun die Idee von Matejka und Fitzmaurice aufgegriffen und optimiert. Anstatt den neuen Datensatz mittels Python zu gegerieren, wurde in dieser Arbeit versucht durch die Verwendung von C++ und OpenMP eine effizientere Lösung zu finden.

2 THE ALGORITHM

Algorithm 1 An algorithm with caption

Require: $n \geq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  $X \leftarrow x$
  $N \leftarrow n$
  while $N \neq 0$ **do**
    if N is even **then**
      $X \leftarrow X \times X$
      $N \leftarrow \frac{N}{2}$ ▷ This is a comment
    else if N is odd **then**
      $y \leftarrow y \times X$
      $N \leftarrow N - 1$
    end if
  end while

2.1 Internal Representation of Mock Labels

In Figure 1 we convert the mock labels to sorted integer sets.

2.2 Efficient Preprocessing of Input Data

The following findings are important to speed up preprocessing of the input data:

- Reading many small files concurrently, with multiple threads (compared to a single thread), takes advantage of the internal parallelism of SSDs and thus leads to higher throughput [2].
- C-string manipulation functions are often significantly faster than their C++ pendants. For example, locating substrings with `strstr` is around five times faster than using the C++ `std::string` function `find`.

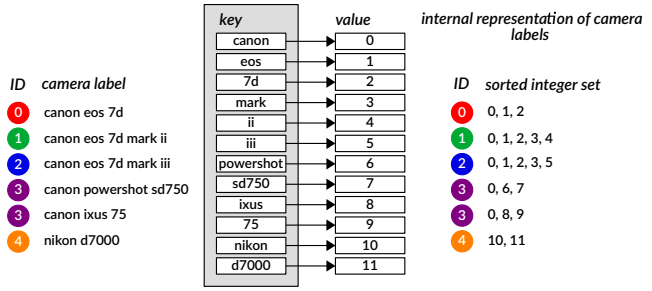

Figure 1: Conversion of mock camera labels to sorted integer sets. We map each unique token (key) in camera labels to a unique value. Based on these key-value-mappings, we convert camera labels to sorted integer sets. A camera can have different names in different countries. Therefore, repeating IDs reference the same cameras (see, for example, ID=3).

- Hardcoding regular expressions with *while, for, switch* or *if-else* statements results in faster execution times than using standard RegEx libraries, where regular expressions are compiled at runtime into state machines.
- Changing strings in place, instead of treating them as immutable objects, eliminates allocation and copying overhead.

3 EXPERIMENTS

Table 1 shows the running times of the resolution step of the five best placed teams.

Table 1: Comparison of the F-measure and the running times of the resolution step of the five best placed teams. The input data for the resolution step consisted of 29,787 in JSON formatted e-commerce websites. Measurements were taken on a laptop running Ubuntu 19.04 with 16 GB of RAM and two Intel Core i5-4310U CPUs. The underlying SSD was a 500 GB 860 EVO mSATA. We cleared the page cache, dentries, and inodes before each run to avoid reading the input data from RAM instead of the SSD.

Team	Language	F-measure	Running time (s)
PictureMe (**this paper**)	C++	0.99	**0.61**
DBGroup@UniMoRe	Python	0.99	10.65
DBGroup@SUSTech	C++	0.99	22.13
eats_shoots_and_leaves	Python	0.99	28.66
DBTHU	Python	0.99	63.21

4 CONCLUSIONS

REFERENCES

[1] Felicitas Macgilchrist. 2014. *Academic writing*. UTB.
[2] Zhenyun Zhuang, Sergiy Zhuk, Haricharan Ramachandra, and Badri Sridharan. 2016. Designing SSD-Friendly Applications for Better Application Performance and Higher IO Efficiency. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*. IEEE. https://doi.org/10.1109/compsac.2016.94