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 zweidimensionaler 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

2.1 Algorithm description

Der Ablauf der Transformation in in Algorithmus 1 dargestellt. Als Input werden folgende Parameter erwartet:

- **iterations** n**:** Das ist die Gesamtanzahl der Iterationsschritte die der Algorithmus durchläuft. Eine größere Zahl bedeutet im Allgemeinen auch ein besseres Ergbenis.
- **inital dataset** D**:** Das ist der Initiale Datensatz, mit dem der Algorithmus startet. Die statistischen Eigenschaften dieses Datensatzes sollen auch versucht beibehalten zu werden.
- **target dataset** T**:** Der Zieldatensatz gibt die Form in Punkten an, in welche der initiale Datensatz transformiert werden soll. Dieser soll ähnliche statistische Eigenschaften, wie der initiale Datensatz, aufweisen.
- **temperature** t**:** Die Temperatur ist notwendig für die Simmulated Annealing. Anhand des Iterationsschritts sowie der Temperatur kann bestimmt werden, ob ein Punkt zufällig akzeptiert wird, auch wenn wir keine Nährung an die Optimallösung haben.
- **error** e**:** Mittels des Errors legen wir die obere Schranke fest, um wie viel die statistischen Eigenschaften zweier Datensätze voneinander Abweichen dürfen.

Die Ausgabe des Agorithmus ist der transformierte Datensatz, welcher bis auf einen Fehler die gleichen statistischen Eigenschaften aufweist, jedoch visuell dem Targetdatensatz ähnelt.

Der Grundlegende Ablauf eines Iterationsschrittes ist dabei wie folgt. Zunächst wird ein zufälliger Punkt aus dem initialen Datensatz ausgewählt. Das Ziel ist, dass dieser so verschoben wird, dass die statistischen eigenschaften beibehalten werden, er jedoch näher an dem Targetdatensatz liegt. Demnach wird der Punkt zufällig in eine Richtung geshifted. Dann wird kontrolliert, ob das Shiften den Punkt näher an den Targetdatensatz gebracht hat. Ist er nicht näher, hat er trotzdem die Chance zufällig akzeptiert zu werden. Die Wahrscheinlichkeit wird durch die Temperatur gesteuert. Wird er auch da nicht akzeptiert, dann wird er erneut zufällig geshifted.

Dies wird solange wiederholt, bis mindestens eine der beiden Bedingungen erfüllt ist. Nun wird kontrolliert, ob der Datensatz mit dem neuen geshifteten Punkt immernoch, bis auf einen Fehler, die gleichen statistischen Eigenschaften aufweist, wie der initiale Datensatz. Wenn nein, dann wird der Punkt verworfen und der nächste Iterationsschritt wird gestaret. Wenn die Eigenschaften übereinstimmen, dann wird der neue geshiftet Punkt Teil des Datensatzes und der nächste Iterationsschritt wird gestartet.

Algorithm 1 Transform Dataset

```
function TRANSFORM(iterations n, inital dataset D, target dataset
T, temperature t, error e)
    for i = 1, ..., n do
        p ← getRandomPoint(D)
        while minDist(p', T) ≥ minDist(p, T) do
            p' ← randomShift(p)
            if allowBreak(i, t) then
                break
            end if
        end while
        D' ← D ∪ {p'} − {p}          ▷ Ersetze p durch p' in D
        if sameStats(D, D', e) then
            D ← D'
        end if
    end for
    return D
end function
```

Zusamenfassend erreicht der Algorithmus, dass in fast jedem Interationsschritt ein Punkt des ursprünglichen Datensatzes so verändert wird, dass die statistischen Eigenschaften beibehalten werden, jedoch die Form sich ein kleines Stück in Richtung des Targetdatensatz bewegt. Über eine große Anzahl von Iterationsschritten wurde ein neuer Datensatz mit ähnlichen statistischen Eigenschaften generiert.

2.2 Statistical Properties

Wie im Verfahren erläutert, werden einige statistische Eigenschaften von Datensätzen betrachtet. Dabei wurden folgende Parameter verwendet:

- Mittelwert
- Standardabweichung
- Korrelationskoeffizient

Im folgenden werden die Eigenschaften genauer beschrieben.

Mittelwert. Der Mittelwert bestimmt das arithmetische Mittel aller Punkte des Datensatzes für jede Dimension. Dieser wird dabei wiefolgt berechnet:

$$\overline{D} = \begin{pmatrix} \overline{x} \\ \overline{y} \end{pmatrix} = \frac{1}{N} \sum_{i=1}^{N} \begin{pmatrix} x_i \\ y_i \end{pmatrix}.$$

In der visuellen Darstellung bietet der Mittelwert einen Punkt, um welchen sich die Daten erstrecken.

Standardabweichung. Die zweite statistische Eigenschaft ist die Standardabweichung. Sie ist ein Maß für die Streuung eines Datensatzes. Dabei wird in jeder Dimension der quadratische Abstand aller Punkte zum Mittelwert akkumuliert und gemittelt. Wie Berechnungsvorschrift ist wie folgt:

$$\sqrt{Var(D)} = \sqrt{\begin{pmatrix} Var(x) \\ Var(y) \end{pmatrix}} = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} \begin{pmatrix} (x_i - \overline{x})^2 \\ (y_i - \overline{y})^2 \end{pmatrix}}.$$

In der visuellen Betrachung kann man mithilfe des Mittelwertes und der Standardabweichung bereits einiges Aussagen. Somit kann nicht nur den Mittelpunkt der Daten betrachten, sondern auch wie weit sich die Daten von diesem Mittelpunkt in alle Dimesionen erstreckt. Um zuletzt noch die Form der Daten zu betimmen wird die dritte statistische Eigenschaft verwendet.

Korrelationskoeffizient. Als Letzes wird der Korrelationskoeffizient herangezogen. Dieser zeigt den linearen Zusammenhang der x- und y-Werte eines Datensatzes auf. Das ist ein Wert im Bereich $[-1, 1]$. Bei einem großen absoluten Wert (1 oder -1) liegt eine vollständige Korrelation vor. Bildlich würde es bedeuten, dass die Daten auf einer Geraden ausgerichtet sind. Ein Wert von 0 würde bedeuten, dass wir eine perfekte symmetrische Verteilung der Daten auf den zwei Dimensionen haben, da die Daten unkorreliert sind. Beispielsweise würde ein zweidimensionaler Datensatz mit normalverteilten x- und y-Werten einen niedrigen absoluten Korrelationskoeffizienten generieren. Die mathematische Berechnungsvorschrift lautet wie folgt:

$$Korr(x, y) = \frac{\sum_{i=1}^{N}(x_i - \overline{x}) * \sum_{i=1}^{N}(y_i - \overline{y})}{N * \sqrt{Var(x)} * \sqrt{Var(y)}}.$$

Mittels dieser statistischen Eigenschaften lassen sich einige Aussagen über bildliche Darstellungen der Daten im zweidimensionalen Koordinatensystem treffen. Somit lässt sich vermuten, dass Datensätze, welche bis auf einen kleinen Fehler (z.B. bis auf die zweite Nachkommastelle genau), identische Mittelwerte, Standardabweichung und Korellationskoeffizienten haben, sehr ähnlich in der bildlichen Darstellung aussehen. Wie sich in den Ergebnissen zeigen lässt, ist dies jedoch nicht immer der Fall. Dabei muss erwähnt werden, dass der Korrelationskoeffizient nur den linearen Zusammenhang aufzeigt. Nichtlineare Abhängigkeiten sind jedoch häufig schwer korrekt abzubilden, sodass eine vereinfachte lineare Abbildung häufig bereits eine gute Approximation ist.

TODO: Simmulated Annealing allowBrak() Umsetzung

2.3 Optimizations

TODO: Wie wurde die Parallelisierung durchgeführt

3 EXPERIMENTS

Im folgenden Abschnitt wird nun betrachtet, welche Auswirkungen die Optimierung des Algorithmus auf die Laufzeit hat.

Verglichen werden dabei folgende Fälle:

- der serielle Python Algorithmus aus der Publikation von Matejka und Fitzmaurice, welches die Implementierung ist, auf welcher die neue Umsetzung basiert.

- der Algorithmus umgeschrieben in C++ und seriell ausgeführt. Somit kann betrachtet werden, welchen Unterschied die Verwendung einer anderen Programmiersprache macht.
- der gleiche Algorithmus in C++, jedoch nun parallelisiert mit den Optimierungen aus Abschnitt 2.3 ausgeführt.

Die Experimente wurden dabei auf TODO: Spezifikationen (Prozessor) ausgeführt. Somit kann eine Vergleichbarkeit erreicht werden.

Experiment 1. Im ersten Experiment wurden alle drei Fälle über unterschiedlich vielen Iterationen ausgeführt. Dabei wurden 100'000, 200'000, 500'000 und 1'000'000 Iterationen als Ausgangslage verwendet. Die Zeiten wurden in Sekunden gemessen. Durch die große Anzahl der Iterationen wird gewährleistet, dass alle Algorithmen zu einer guten Lösung kommen. Bei dem Fall *C++/parallel* wurde die beste parallele Zeit verwendet, die erreicht werden konnte, ohne dass größere Verminderungen des Ergebnisses zur Folge gekommen sind. In diesem Fall wurden TODO: Anzahl Threads verwendet. Die Ergenisse für dieses Experimente sind in Tabelle 1 zu erkennen.

TODO: Auswertung serieell para Tabelle

Allgemein lässt sich erkennen, dass allein die Verwendung einer Programmiersprache, wie C++ die Laufzeit enorm verkürzt. Durch Compileroptimierungen, sowie der generellen effizienten Speicheroptimierung ist sie bei berechnungsintensiven Anwendungen klar im Vorteil. Durch die zusätzliche Verwendung von OpenMP und die Aufteilung des Problems in Subprobleme kann eine weitere Beschleunigung erreicht werden, welche jedoch erst bei vielen Iterationen und großen Datensätzen eine sinnvolle Anwendung findet.

Table 1: Vergleich der Algorithmen und Ausführungsart über 100'000, 200'000, 500'000 und 1'000'000 Iterationen. Alle Angaben in Sekunden.

Iterations	Python/serial	C++/serial	C++/parallel
100,000	0	0	0
200,000	0	0	0
500,000	0	0	0
1,000,000	0	0	0

Experiment 2. Im zweiten Experiemnt wurde nun der parallele C++ Algorithmus aufgegriffen und mit unterschiedlich vielen Threads ausgeführt. Auch hier wurden erneut 100'000, 200'000, 500'000 und 1'000'000 Iterationen als Ausgangslage verwendet. Mithilfe der zusätzlichen Benutzung der Threads soll betrachtet werden, in wie weit ein Speedup erreicht werden kann. Erwartet wird, dass der inhärent serielle Anteil, also der Bestandteil des Algorithmus, welcher schlecht bzw. gar nicht parallelisierbar ist, niedrig ist. Somit sollte die Verwendun mehrerer Thread zu einem größeren Performance-Gewinn führen. Die Ergenbisse des zweiten Experiments sind in Tabelle 2 aufgelistet.

TODO: Auswertung para Tabelle

Die Anzahl der Threads hat somit einen Einfluss auf die Laufzeit des Algorithmus. Jedoch muss beachtet werden, dass dies nur sinnvoll ist, wenn die Datensätze groß sind. Durch die verringerte Anzahl der Freiheitsgrade bei der Berechung der statistischen Eigenschaften sind die Ergebnisse bei kleinen Datensätzen häufig nicht zufriedenstellend oder benötigen noch einige zusätzliche Iterationen, welche seriell nicht benötigt werden.

Table 2: Vergleich der des parallelen C++ Algorithmus mit unterschiedlich vielen Threads über 100'000, 200'000, 500'000 und 1'000'000 Iterationen. Alle Angaben in Sekunden.

Iterations	2 Threads	4 Threads	8 Threads
100,000	0	0	0
200,000	0	0	0
500,000	0	0	0
1,000,000	0	0	0

Somit bestätigen die Experimente die Ziele eine effizientere Generierung eines Datensatzes mit ähnlichen statistischen Eigenschaften zu ermöglichen.

4 CONCLUSIONS

REFERENCES

[1] Felicitas Macgilchrist. 2014. *Academic writing.* UTB.