Simulated Annealing

Robin Schmidt, Johannes Piotrofski

29. Januar 2020

1 Grundlagen und Problemstellung

1.1 *Travelling Salesman Problem*

Die Problemstellung ist eine Abwandlung der klassischen Optimierungsaufgabe in der ein globales Maximum oder Minimum gefunden werden soll um einen Prozess möglichst effizient oder kostengünstig zu gestalten. Ein fiktiver Handelsreisender möchte in N Städten sein Geschäft betreiben und möchte dabei den kürzesten Weg durch diese Städte nehmen. Während das Problem für kleine N leicht zu lösen ist indem alle Möglichkeiten ($N!$) der Anordnung ausprobiert werden, wird der Raum der zu überprüfenden Kombinationen für große N schlicht zu groß um ihn zu durchsuchen.

Im Fall dieses Projektes wurden 40 deutsche Städte ausgewählt, die durch einen geschlossenen Pfad mit der kürzesten Länge verbunden werden sollen. Die geografischen Koordinaten der Städte waren gegeben.

1.2 *Simulated Annealing*

Beim *Simulated Annealing* handelt es sich um ein heuristisches Approximationsverfahren welches in der Lage ist näherungsweise Lösungen für Probleme der oben beschriebenen Art zu finden.

Die Idee ist, ausgehend von einem zufällig gewählten Startpunkt einen Schritt zu machen und danach zu evaluieren, wie es weiter gehen soll. Die Schritte werden im Fall des TSP realisiert durch das Vertauschen zweier zufällig gewählter Städte im Pfad oder durch das Vertauschen der Richtung eines Pfades zwischen zwei Städten. Wenn sich die Länge des gesamten Pfades nach dem Schritt verringert hat, so wird der neue Pfad als bessere Alternative zum letzten akzeptiert. Ist die Gesamtlänge größer geworden so wird zunächhst eine Wahrscheinlichkeit P in Abhängigkeit einer Temperatur T und der Längendifferenz ΔL des alten und neuen Pfades definiert:

$$p \sim e^{-\frac{\Delta L}{T}} \tag{1}$$

In einem Zufallsprozess wird nun entschieden, ob der längere Weg akzeptiert wird als neuer Ausgangspunkt für die nächste Iteration oder ob der letzte Pfad als bessere Alternative behalten wird. Der Parameter T klingt dabei exponentiell über die Gesamtheit aller Iterationen, also Schritte, des Pfades ab.

Man erlaubt also dem System *immer*, einen Schritt abwärts, also zu einer kürzeren Gesamtlänge des Pfades zu machen und man erlaubt ihm *manchmal* einen Schritt aufwärts zu einer größeren Gesamtlänge zu machen.

Ohne die Schritte aufwärts würde der Pfad nach wenigen Iterationen in einem lokalen Minimum der Pfadlänge landen und dort auch bleiben. Aufgrund der Größe des Raumes an verschiedenen Pfaden ist es jedoch sehr unwahrscheinlich, dass es sich bei dem lokalen Minimum schon um eine akzeptable Lösung handelt.

Die Idee hinter dem Algorithmus ist also die Ermittlung eines globalen Optimums bei gleichzeitiger Existenz vieler lokaler Optima indem dem System erlaubt wird, ein lokales Optimum mit einer gewissen Wahrscheinlichkeit wieder zu verlassen.

2 Aufgabenstellung

Die Aufgabenstellung dieses Projektes war es, *Simulated Annealing* als Algorithmus in einer beliebigen Programmiersprache zu implementieren und damit approximative Lösungen für das *Travelling Salesman Problem* zu finden.
Als Iteration des aktuellen Pfades im Laufe des Algorithmus sollten zwei verschiedene Methoden untersucht werden: das Vertauschen zweier zufällig ausgewählter Städte in der Reihenfolge des Pfades (*switching random towns, SRT*) sowie das Umdrehen eines zufällig ausgewählten Teilstücks (mit Mindestlänge 2) des Pfades (*reverting one path, ROP*) ist. Die Methoden der Iteration einer Kette von Elementen werden beispielhaft in Abb. 1 gezeigt.

Zusätzlich sollte die Abhängigkeit der Qualität der Lösungen von verschiedenen Kühlungsparametern α_i sowie die generelle mittlere Laufzeit t_{mean} des Algorithmus ermittelt werden.

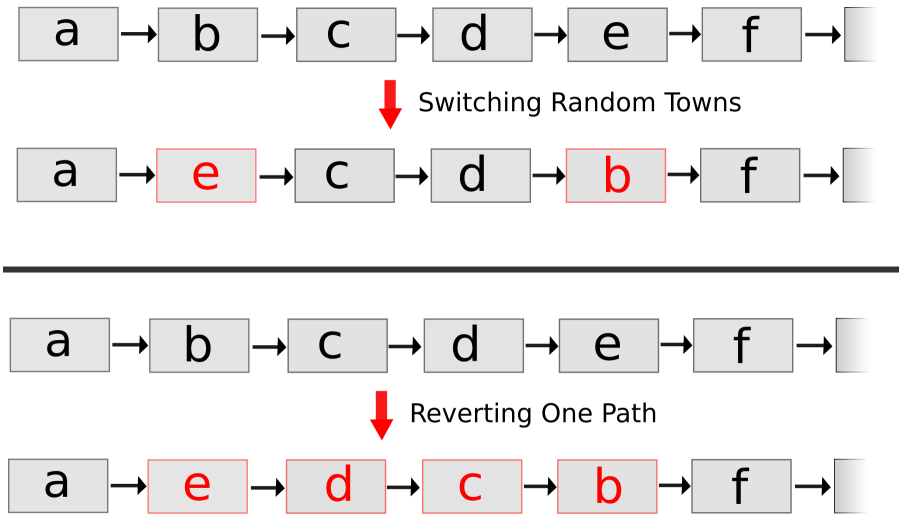

Abbildung 1: Beispielhafte Darstellung der am Pfad durchgeführten Iterationen

3 Ergebnisse

Der Algorithmus wurde in Python 3.6 implementiert, ein Link zu einem ausführbaren Beispiel kann im Anhang gefunden werden. Die Koordinaten der Städte werden aus gegebenen Dokumenten eingelesen und die Distanzen zwischen den Städten einmal berechnet mithilfe der *Haversine-Formel*. Die Distanzen werden in einer Matrix gespeichert und werden für die Berechnung der Länge eines Pfades nur noch abgerufen. Der exponentielle Abfall des Temperatur-Parameters wurde realisiert durch die Funktion $T(n) = (\alpha)^n \cdot 1000$ wobei n die Schrittzahl ist, die bis $n = 1000$ läuft.

$$\alpha_1 = 0.995 \qquad \alpha_2 = 0.991 \qquad \alpha_3 = 0.987$$

Für **jede** Temperatur werden wiederum $n_{Sub} = N^2 = 40^2$ Iterationen des Pfades durchgeführt. Insgesamt werden also in einem Durchlauf des Algorithmus für einen gegebenen Anfangspfad $n \cdot n_{Sub} = 1600000$ verschiedene Versionen dieses Pfades generiert und deren Länge verglichen.

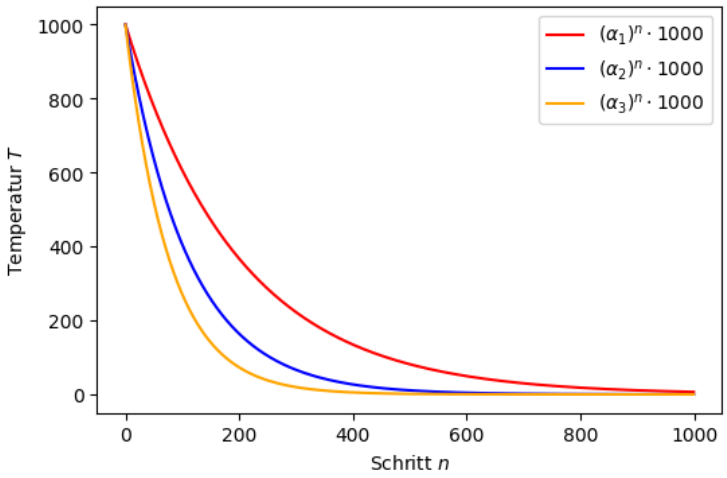

Abbildung 2: Darstellung der Verringerung der Temperatur in Abhängigkeit von verschiedenen Kühlungsparametern α_i

3.1 Tatsächlich kürzester Pfad

Der kürzeste Pfad P_1 wurde von Hand aus der geografischen Verteilung der 40 Städte ermittelt. Seine Länge beträgt 2783.6 Kilometer. Der Pfad ist in Abbildung 3 dargestellt. Um sicherzugehen, dass es sich dabei tatsächlich um den kürzesten Pfad handelt, wurde auch dieser noch mehrere Male durch den Algorithmus überprüft. Die Länge konnte jedoch nicht weiter verringert werden. Damit eignet sich P_1 als Maß der Qualität der vom Algorithmus berechneten Lösungen.

Als Startpunkt (und Endpunkt) wird zur Initialisierung des Algorithmus für die Vergleichbarkeit der Lösungen immer Rostock festgelegt. Danach werden 39 weitere Städte jeweils einmal zufällig aus einer Liste von allen denen gezogen, die auch in P_1 enthalten sind. Es ergibt sich eine zufällige Permutation von P_1, die als Ausgangspunkt für die Optimierung dient.

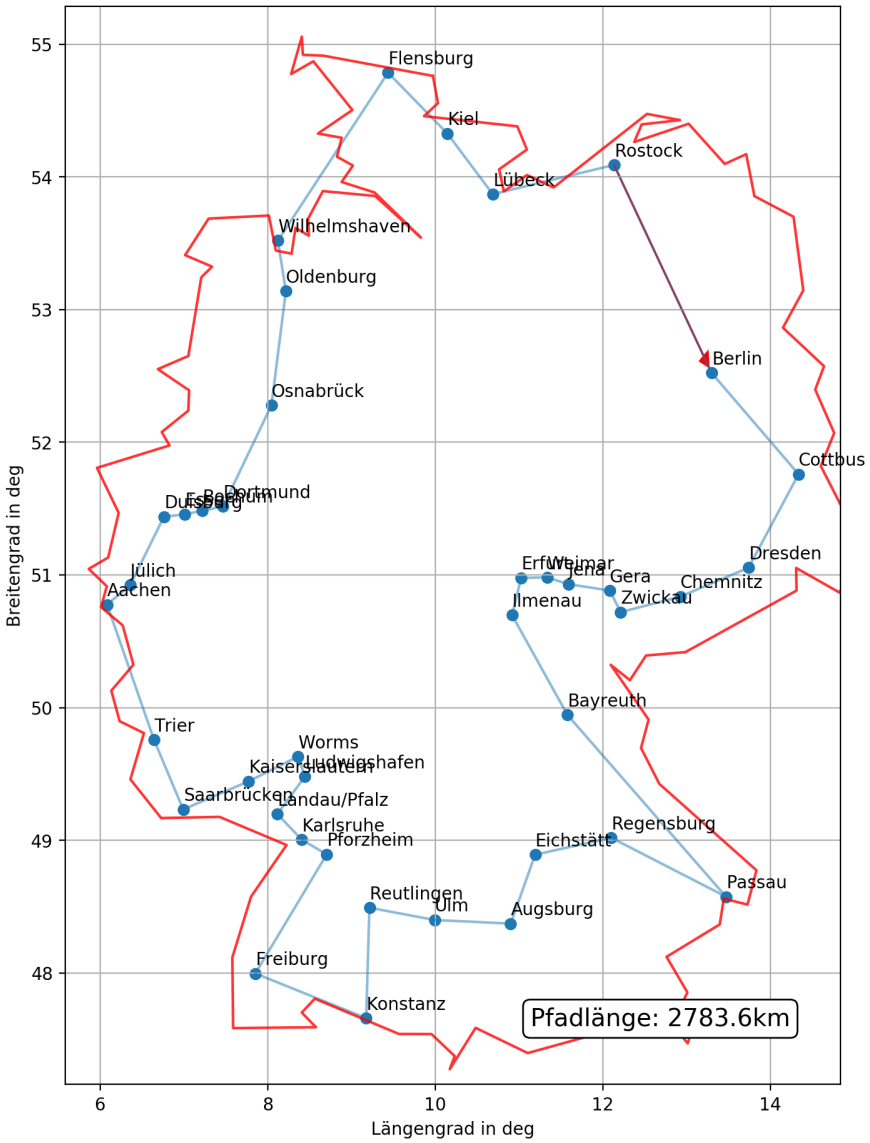

Abbildung 3: Darstellung des kürzesten Pfades P_1 durch 40 ausgewählte Städte. Start in Rostock. Die Umlaufrichtung wird durch den roten Pfeil angegeben.

Die durchgeführten Experimente, die im nachfolgenden Abschnitt dargestellt werden haben gezeigt, dass neben dem tatsächlich kürzesten Pfad P_1 noch ein weiterer in etwa 20% der Versuche gefunden wird. Dieser zweite Pfad, P_2, ist nicht optimal bezüglich der Gesamtlänge, unterscheidet sich jedoch nur geringfügig vom optimalen Pfad. Der relative Unterschied in der Gesamtlänge beträgt nur etwa 0.25%. Ein Vergleich der beiden Pfade ist in Abb. 4 gegeben.

Der geringe Unterschied in der Gesamtlänge zwischen P_1 und P_2 erklärt die Tatsache, dass auch der nicht-optimale Pfad P_2 regelmäßig gefunden wird. Sobald der Algorithmus einmal diesen Pfad gefunden hat, können auch mehrere Hunderttausend Iterationen des Pfades bei bereits geringer Temperatur (siehe auch Abb.5) keine bessere Lösung mehr produzieren, die Lösung ist also vergleichsweise stabil (zumindest für *ROP*, siehe nächsten Abschnitt.)

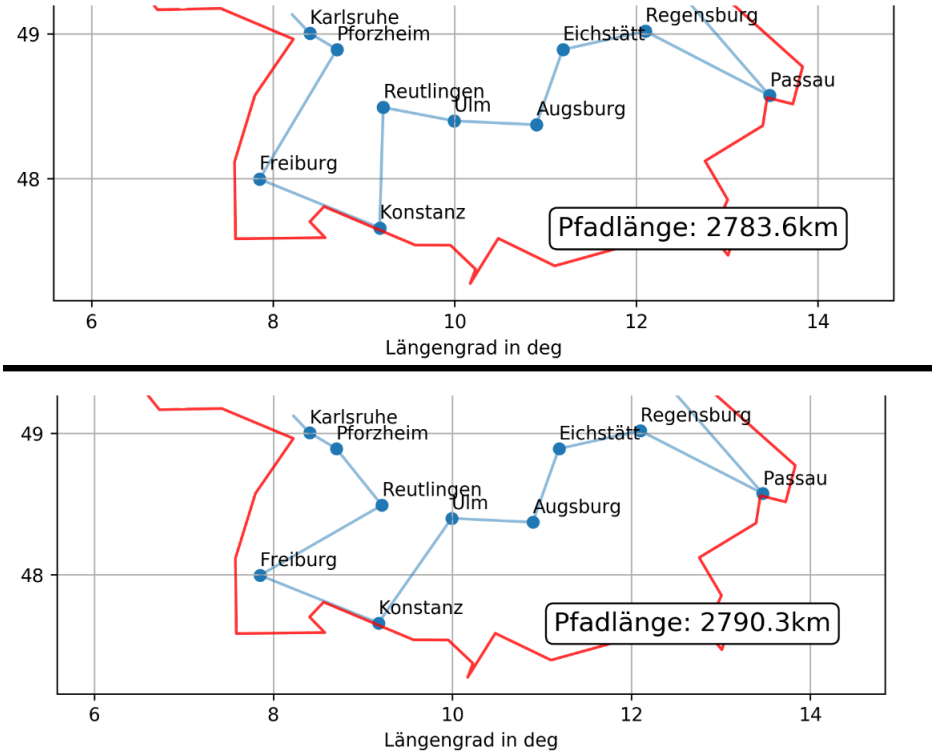

Abbildung 4: Vergleich des tatsächlich kürzesten Pfades P_1 (2783.6km, oben) mit dem nicht-optimalen anderen Pfad P_2 (2790.3km, unten), der auch regelmäßig als Lösung gefunden wird.

3.2 *Reverting One Path*

In Abbildung 5 sind links beispielhaft die Entwicklung der Pfadlänge von Initialisierung bis Konvergenz unter Benutzung der Invertierung eines mittleren Kettenstücks gezeigt. Es sind drei Anwendungen des Algorithmus auf einen gegebenen Anfangspfad (kurz: ein *Versuch*) dargestellt für verschiedene Kühlungsparameter α.

In den Darstellungen der Entwicklung der Pfadlänge zeigt sich ein deutlicher Unterschied für verschiedene Kühlungsparameter. Für den größten Parameter $\alpha = 0.995$ fällt die Pfadlänge im Verlauf aufeinanderfolgender Iterationen langsam ab. Im rechten Teil des Bildes zeigt sich eine verstreute Verteilung von den finalen Längen von 50 Versuchen, dargestellt als Histogramm sowie mit einer Approximation der Wahrscheinlichkeitsdichtefunktion (*kernel density estimate, KDE*). Das Maximum der Verteilung liegt deutlich bei Werten um 2790km. Das bedeutet, dass der absolute Großteil aller Versuche den kürzesten Pfad P_1 (bzw. die sehr naheliegende Lösung P_2) gefunden haben. Es ist ebenfalls zu beachten, dass selbst die in der Verteilung am weitesten rechts liegenden Lösungen mit etwa 2840km Gesamtlänge sich nur um etwa 2,5% von P_1 unterscheiden, also auch diese vergleichsweise qualitativ hochwertige Lösungen sind.

Für geringere Kühlungsparameter $\alpha = 0.991$ und $\alpha = 0.987$ zeigt sich links eine deutlich schnellere Konvergenz zu guten Lösungen. In den Verteilungen rechts im Bild kondensieren alle Versuche in zwei Werten, die exakt den Pfaden P_1 und P_2 entsprechen. Es wurde bereits erwähnt, dass es sich bei P_2 nicht um einen optimalen Pfad handelt, der Algorithmus also in diesem Fall nicht die beste Lösung liefert. Das ist jedoch der Tatsache geschuldet, dass P_1 von Hand gewählt wurde um die Vergleichbarkeit von Lösungen zu ermöglichen und dass rein zufällig eine sehr naheliegende Lösung P_2 existiert, die scheinbar auch für viele Iterationen mit bereits niedriger Temperatur stabil ist.

Der Grund für diese Stabilität liegt in der Methode der Iteration selbst. Bei Betrachtung der Abb. 4 kann man nachvollziehen, dass es nicht möglich ist, mit einer Anwendung von *ROP* auf P_2 den optimalen Pfad P_1 zu generieren. Stattdessen müsste man eine Kette von Anwendungen von *ROP* auf jeweils nächste Nachbarn anwenden, um bei P_1 anzukommen. Diese einzelnen Anwendungen hätten aber jeweils eine Verlängerung der Gesamtlänge zur Folge, die wiederum für klein werdende Temperaturen nur sehr wenig wahrscheinlich akzeptiert wird. Die Wahrscheinlichkeit, dass diese spezifischen Anwendungen konsekutiv geschehen ist also ebenfalls gering und somit kann der Algorithmus in manchen Fällen bei P_2 in einem lokalen Minimum verbleiben.

Allgemein zeigt Abb. 5, dass die Methode der Iteration *ROP* in der Lage ist qualitativ hochwertige Lösungen für die Problemstellung zu liefern und dass geringere Kühlungsparameter eine schnellere Konvergenz des Algorithmus bewirken kann.

3.3 *Switching Random Towns*

Abbildung 6 zeigt die Ergebnisse des Algorithmus bei Iteration durch Vertauschung zweier Städte.

Die Entwicklung der Pfadlänge zeigt für verschiedene Kühlungsparameter einen ähnlichen Verlauf wie in Abb.5. Kleinere α zwingen den Algorithmus bei einer deutlich geringeren Anzahl von Gesamtiterationen zu Lösungen von besserer Qualität. Die Histogramme rechts im Bild zeigen ein deutliches Maximum bei etwa 2800km Gesamtlänge, was dafür spricht, dass im Großteil aller Fälle P_1 bzw. P_2 gefunden werden.

Die Histogramme zeigen allerdings eine Veränderung für verschiedene Kühlungsparameter: die Streuung der Gesamtlängen der gefundenen Lösungen wird für kleiner werdende α größer. Das bedeutet, dass für kleinere α qualitativ schlechtere Lösungen wahrscheinlicher werden. Das steht im starken Kontrast zur Entwicklung der Histogramme für *ROP*, wo qualitativ bessere Lösungen mit kleiner werdendem α wahrscheinlicher werden.

Der Vorteil von *SRT* gegenüber *ROP* ist der, dass *SRT* wenig anfällig dafür ist im lokalen Minimum vom nicht optimalen Pfad P_2 steckenzubleiben. Der Grund dafür ist, dass *SRT* mit einer einzigen Anwendung problemlos den Pfad P_1 aus P_2 generieren kann. Die Wahrscheinlichkeit für diese Vertauschung ist im Verlauf von vielen Iterationen nicht gering. Damit wird es unwahrscheinlich, dass der Algorithmus für viele Iterationen bei P_2 verbleibt, so wie es bei *ROP* der Fall sein kann.

Die Iterationsmethode *SRT* ist also ebenfalls in der Lage, akzeptable Lösungen für die Problemstellung zu liefern, zeigt jedoch für kleiner Werdende Kühlungsparameter ein zur Methode *ROP* gegensätzliches Verhalten.

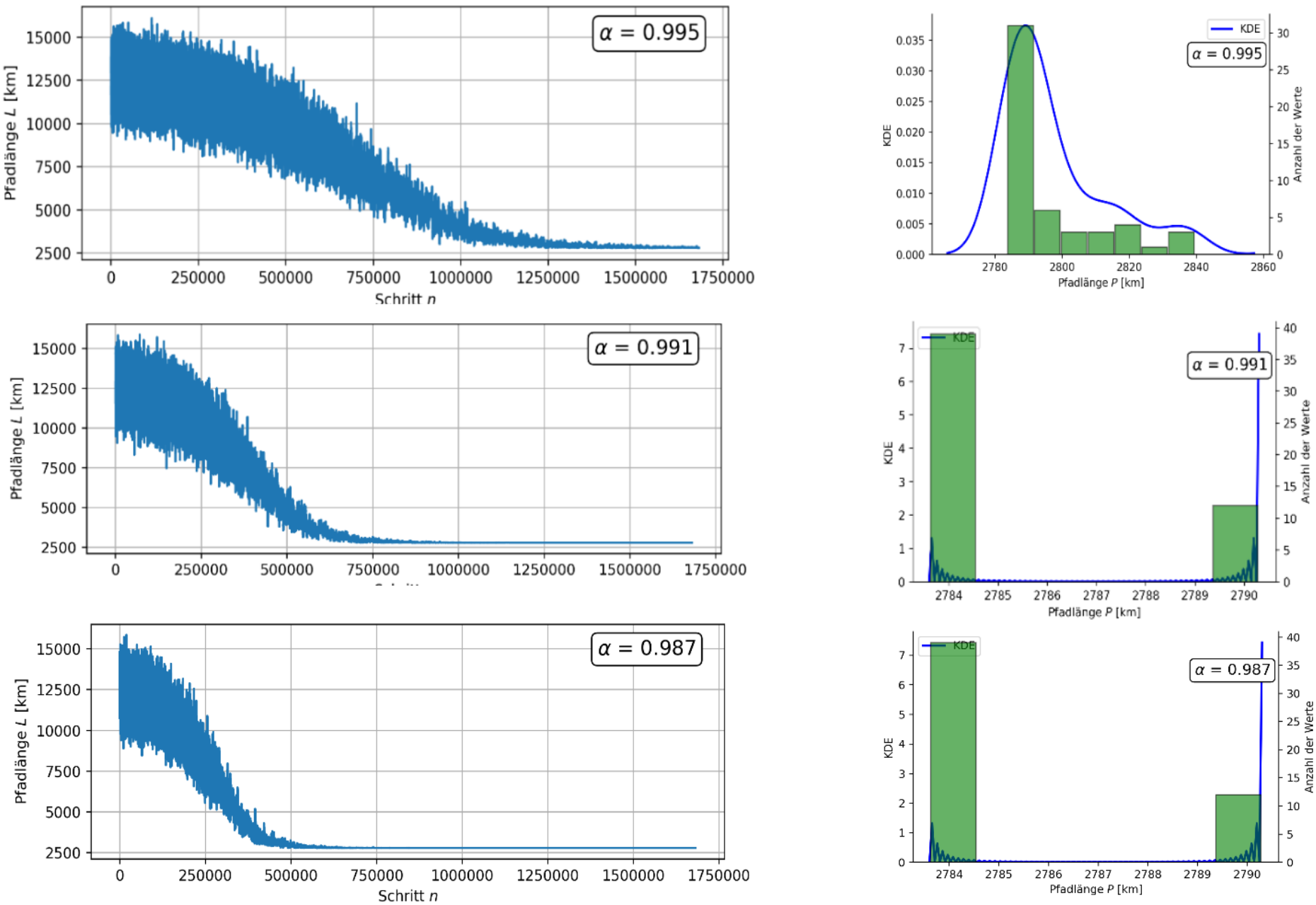

Abbildung 5: Entwicklungen der Pfadlänge einiger Versuche (links) und die finale Verteilung der Längen für 50 Versuche (rechts) in Abhängigkeit von verschiedenen α. **Iterationsmethode: Reverting One Path, ROP**

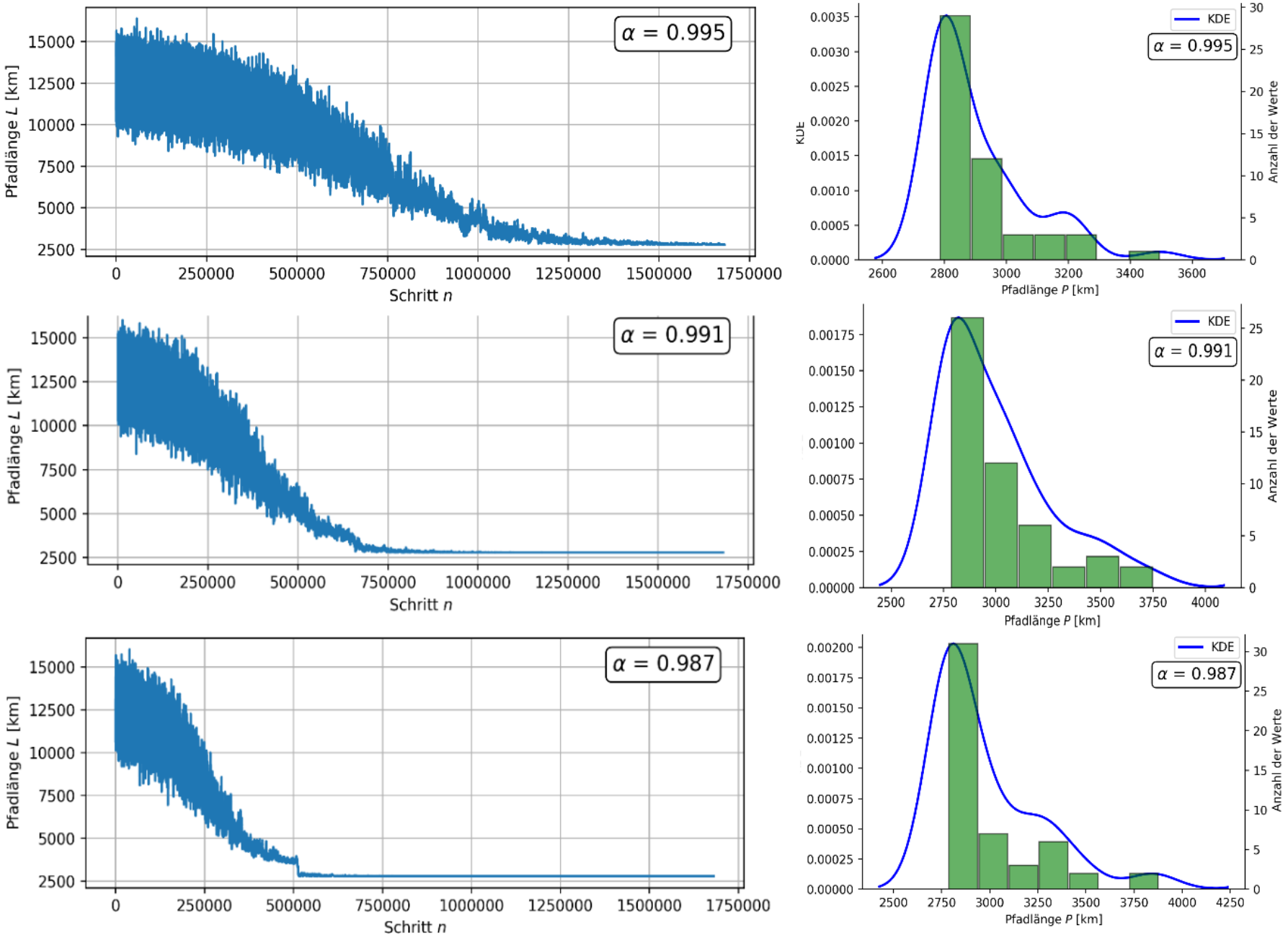

Abbildung 6: Entwicklungen der Pfadlänge einiger Versuche (links) und die finale Verteilung der Längen für 50 Versuche (rechts) in Abhängigkeit von verschiedenen α. **Iterationsmethode: Switching Random Towns, SRT**

3.4 Laufzeit bis zur Angabe einer Lösung

Abbildung 7 zeigt die Laufzeit des gesamten Programms in Sekunden bis zur Angabe einer Näherungslösung für die beiden verschiedenen Methoden der Iteration. Aufgetragen wurden jeweils die Mittelwerte der Laufzeit nach 50 Wiederholungen des Programmes für jedes n. Wie zu erwarten steigt die gesamte Laufzeit linear mit der Anzahl der Iterationen, es gibt keine großen Unterschiede zwischen den Arten der Iteration.

Zu bemerken ist, dass für $n = 600$ und $n = 700$ die Qualität der gelieferten Lösungen deutlich schlechter ist als jene, die in den vorherigen Abschnitten gezeigt wurden (wo $n = 1000$ genutzt wurde). Der Grund dafür ist, dass dem System nicht genügend Schritte erlaubt werden, um ein globales Minimum oder eine gute Näherung davon zu finden.

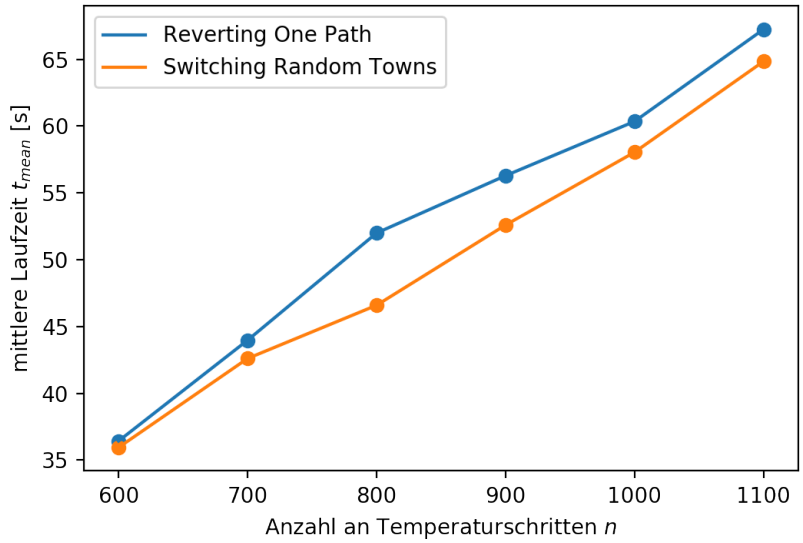

Abbildung 7: Abhängigkeit der mittleren Laufzeit des Programms von der Anzahl an Iterationen, $N = 40$ Städte

Der erkennbare Versatz der beiden Kurven in Abb. 7 lässt sich durch die verschiedenen Zeitkomplexitäten der Arten der Iteration erklären. Beide sind im Wesentlichen Manipulationen von Listen und bei beiden spielt somit die Länge der Liste, also die Menge an Städten N, die tragende Rolle.

Die meisten Listenmanipulationen haben in *Python* entweder die Komplexitätsklasse $\mathcal{O}(N)$ oder $\mathcal{O}(1)$.[1] Es ist also für Operationen wie das Zerschneiden, Invertieren und Zusammenfügen die in *SRT* und *ROP* genutzt werden eine lineare Abhängigkeit der Laufzeit eines Versuches von der Anzahl von Städten in der Kette zu erwarten. Abbildung 8 zeigt diese lineare Beziehung. Weil für einen Durchlauf des Programms die Listenoperation $n \cdot n_{sub} = 1000 \cdot N^2$ mal aufgerufen wird, muss in der Laufzeit für den Wert N^2 korrigiert werden, damit die lineare Abhängigkeit erkennbar wird.

Es ist ebenfalls erkennbar, dass *ROP* immer etwas mehr Zeit benötigt als *SRT*. Das kann dadurch erklärt werden, dass bei *ROP* mit dem Invertieren eines mittleren Kettenstücks eine Operation zusätzlich durchgeführt werden muss, die bei *SRT* nicht nötig ist. Der Unterschied zwischen *ROP* und *SRT* in der angepassten Laufzeit in Abb. 8 für $N = 40$ erklärt den Versatz der dargestellten Kurven in Abb. 7.

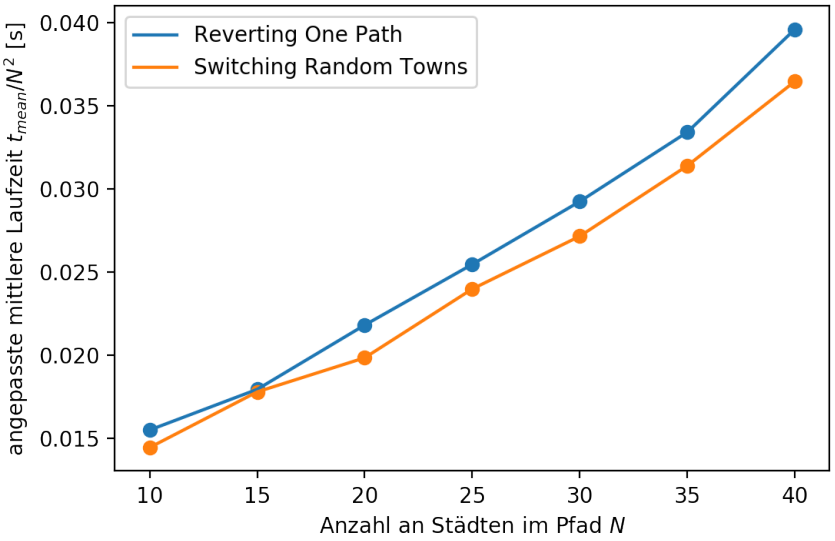

Abbildung 8: Abhängigkeit der angepassten mittleren Laufzeit des Programms von der Anzahl an Städten im Pfad, $n = 1000$ Temperaturschritte

[1] Quelle: https://www.ics.uci.edu/ pattis/ICS-33/lectures/complexitypython.txt

3.5 Fazit

Die in vorigen Abschnitten dargestellten Untersuchungen zeigen, dass *Simulated Annealing* einen guten Ansatz liefert um das hochkomplexe *Traveling Salesman Problem* näherungsweise zu lösen. Beide untersuchten Methoden der Iteration des Pfades konnten für dieses Beispiel innerhalb von Zeiten im Bereich von 60 Sekunden Lösungen liefern deren Pfadlänge nur noch einen Bruchteil der Ausgangslänge betrug.

Es wurde ermittelt, dass die Iterationsmethode *ROP* tendenziell eine höhere Wahrscheinlichkeit hat, hochwertige Lösungen zu liefern als *SRT*. Während bei *SRT* nur ein Teil aller gelieferten Lösungen eine hohe Qualität hat, ist das für *ROP* für kleiner werdende Kühlungsparameter *praktisch immer der Fall.*

In der Zeitkomplexität zeigt sich *SRT* etwas besser, also schneller, als *ROP*, wenn auch nur mit geringem Unterschied. Dieser Unterschied könnte eventuell auch nur ein Artefakt der hier gewählten Implementation der Iterationsmethode sein, also aus einer nicht-optimalen Programmierung resultieren. Entsprechend kann dieser Zeitunterschied möglicherweise durch bessere Implementationen verringert werden. Eine komplette Elimination des Zeitunterschiedes ist unwahrscheinlich, da wie im vorigen Abschnitt erwähnt bei *ROP* mit der Invertierung des Kettenstücks immer eine $\mathcal{O}(N)$-Operation mehr ausgeführt werden muss als bei *SRT*. Beide Methoden haben eine linear von N abhängende Zeitkomplexität, aber durch die zusätzliche Operation wird der Anstieg der Kurve bei *ROP* etwas stärker sein als bei *SRT*.

Der Grund für die größere Effizienz von *ROP* könnte darin liegen, dass bei der zufälligen Auswahl und der Invertierung eines Kettenstücks Pfaditerationen generiert werden, die im Konfigurationsraum weiter entfernt voneinander liegen, als wenn nur zwei Elemente der Kette vertauscht würden. Diese „größeren Sprünge im Konfigurationsraum“ könnten in Verbindung mit der veränderlichen Wahrscheinlichkeit, neue Lösungen zu akzeptieren und mit vielfacher Iteration vergleichsweise schneller dazu führen, dass qualitativ hochwertige Lösungen gefunden werden.

Ebenso kann vermutet werden, dass durch die „kleineren “ Sprünge von *SRT* weniger schnell gute Lösungen produziert werden. Wird dann auch noch der Kühlungsparameter verringert, also die Flexibilität der weiteren Iteration von Lösungen verringert, so könnte sich ein Verhalten ergeben, wie es in Abb. 6 zu sehen ist. Die Temperatur der Optimierung kühlt *zu schnell* ab, was in einer größeren Wahrscheinlichkeit von qualitativ schlechteren Lösungen resultiert.

4 Anhang

Link zum geschriebenen Programm (`.rar`-Datei): www.mega.co.nz

Inhalt:

- `SimulatedAnnealing.py` (ausführbar)
- `distances_and_other_functions.py` (liefert Funktionen die im anderen Programm genutzt werden)
- `Germany.dat` (Koordinatensatz um Grenzlinie von Deutschland zu plotten)