Evolutionäre Optimierungsalgorithmen

Federico Ramírez Villagrana

Universität Hamburg
Methodenkompetenz - Einführung in das wissenschaftliche Arbeiten
Dozent: Dr. Andreas Günther

Zusammenfassung

In diesem Paper wird darüber diskutiert, sowohl was Optimierung ist und warum ist es schwer, als auch was evolutionärer Algorithmen (EA) sind und wie, warum, und wann sind sie hilfreich in dem Optimierung-Bereich. Es wird auch im Detail erklärt, wie EA funktionieren und warum sind sie als teil des künstlichen Intelligenz betrachtet. Es werden auch einige bekannte EA erwähnt und oberflächlich erklärt. Am ende wird darüber diskutiert, was für Vor- und Nachteile die EA haben.

I. Einleitung

Das Problem des Handlungsreisenden (engl. traveling salesman problem oder TSP) ist ein sehr bekanntes und studiertes Problem in dem Informatik-Bereich und es geht wie folgendes:

Es muss eine Reihenfolge für den Besuch mehrerer Orte so gewählt sein, dass keine Station außer der ersten mehr als einmal besucht wird, die gesamte Reisestrecke des Handlungsreisenden möglichst kurz, und die erste Station gleich der letzten Station ist. [wikc]

Sei n die Anzahl der Stationen, es gibt $(n-1)!$ mögliche Lösungen für das TSP. Das heißt, dass für $n = 4$ es 6 mögliche Lösungen gibt. Es ist nicht schwer einen brute-force [1] Ansatz zu verfolgen für ein Problem, das nur 6 Lösungen hat. Aber wenn $n = 50$, gibt es circa $6,1 \times 10^{62}$ Lösungen.

Folgendes ist hilfreich, um diese Anzahl in einer Perspektive zu setzen: Das Universum ist circa 15 Milliarde Jahre alt, das ist $4,7 \times 10^{17}$ Sekunden. Wenn es eine Billion Rechner gäbe, die jede einzeln seit dem Anfang des Universums eine Billion Lösungen pro Sekunde berechnete, bisher wären nur $4,7 \times 10^{41}$ Lösungen berechnet worden.

Das TSP ist nur ein Beispiel von vielfältigen Probleme, die zu den kombinatorischen Problemen gehören, das heißt, Probleme für die einfach kein brute-force Ansatz möglich ist. Es ist in diesem Fall, dass evolutionärer Algorithmen (EA) hilfreich sind.

EA sind ein geeignetes Werkzeug, um gute Lösungen für kombinatorische Probleme zu finden. Natürlich können wir nicht sicher sein, dass wir die beste Lösung gefunden haben, außer wenn wir jede mögliche Lösung berechnen haben. Aber wie es mit dem TSP-Beispiel gezeigt ist, es ist nicht immer möglich, alle mögliche Lösungen (in angemessener Zeit) zu berechnen.

II. Stand der Forschung

Todo

[1] Auch Exhaustionsmethode, ist eine Lösungsmethode für Probleme, die auf dem Ausprobieren aller möglichen (oder zumindest vieler möglicher) Fälle beruht. [wika]

III. Optimierung

In dem Mathematik-Bereich, Optimierung bedeutet die Findung von Parametern eines Systems, die ein bestmögliches Ergebnis erzielen. [wikb] Wir können diese Findung von Parametern auch wie folgendes definieren: Die Auswahl der bestmöglichen Lösung eines Problems von einer Menge mögliche Lösungen.

In Optimierungsprobleme wird eine Zielfunktion (engl. objective function) definiert, die entweder maximiert oder minimiert werden soll. Die Zielfunktion wird „Kosten-Funktion“ oder „Fitness-Funktion“ in Minimierung- und Maximierungsprobleme bzw. genannt.

Sei $f(x)$ ein Zielfunktion, x ist ein Vektor und wird „Entscheidungsvariable“ genannt. Die Anzahl von Elementen in x wird die „Dimension“ des Problems genannt. Die Domäne der Zielfunktion repräsentiert die mögliche Lösungen des Problems.

Das Ziel der Optimierung ist es denn, die bestmögliche Lösung des Problems zu finden, das heißt, einen Wert für die Entscheidungsvariable zu finden, sodass die Zielfunktion einen maximal bzw. minimal Wert ergibt.

Es gibt verschiedene Klassifizierungen oder Arten von Optimierung:

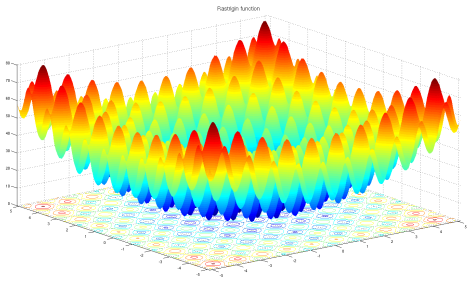
- Eingeschränkt Optimierung.
  Die Entscheidungsvariable kann nicht irgendwelchen Wert aus der Domäne der Zielfunktion nehmen. Es gibt Einschränkungen dafür.

- Multi-objective Optimierung.
  Das Problem besteht aus vielfältigen, voneinander unabhängigen Zielfunktionen.

- Multi-modal Optimierung.
  Die Zielfunktion(en) hat/haben mehrere Minima bzw. Maxima. Abbildung 1 zeigt eine multi-modale Funktion.

Eine der wichtigsten und schwersten Teile der Optimierung ist es, eine geeignete Zielfunktion zu definieren, die alle die wichtige zu optimieren Faktoren berücksichtigt. Probleme im wirklichen Leben sind normalerweise eingeschränkt, multi-Objective und multi-modale, oder mit einer hoch Anzahl von Dimensionen. Wegen diesen Eigenschaften der Zielfunktionen und der Optimierungsprobleme ist es schwer eine gute Lösung mittels traditioneller Vorgänge zu finden. Im Laufe der Zeit haben EA sich als gutes Werkzeug zur Lösung dieser Art von Probleme in angemessener Zeit erwiesen.

Abbildung 1: *Rastrigin function. Quelle:* `https://commons.wikimedia.org/wiki/File:Rastrigin_function.png`



IV. Was ist ein evolutionärer Algorithmus?

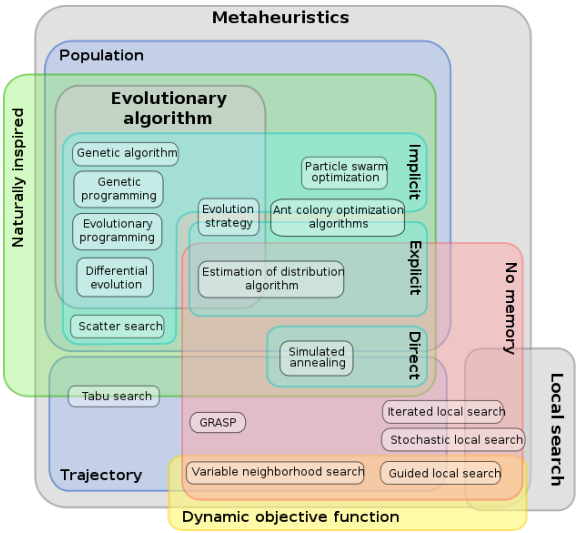
Der EA-Bereich ist ziemlich neu, deswegen gibt es bisher keine allgemein akzeptierte Definition von evolutionäre Algorithmus.

EA sind zwar als Teil des künstliche Intelligenzes (KI) betrachtet, aber genau wo sie in Verbindung mit anderen KI-Methoden stehen, und was der EA-Bereich beinhaltet, ist von dem Autor abhängig. Abbildung 2 zeigt eine von mehrere mögliche Klassifikationen von KI-Methoden in Bezug auf Metaheuristics (welche über den Rahmen dieses Papiers hinausgehen).

In Abbildung 2 kann es auch erkannt werden, dass der Autor Particle Swarm Optimization (PSO) und Ant Colony Optimization (ACO) nicht als EA betrachtet, trotzdem gibt es andere Autoren (wie z. B. [Sim13]), die diese Algorithmen genau für EA halten.

In diesem Paper wird folgende Definition für evolutionärer Algorithmus genommen: Ein

Abbildung 2: *Klassifikation von Metaheuristics. Quelle:* `https://en.wikipedia.org/wiki/File:Metaheuristics_classification.svg`



Algorithmus, der die Lösung eines Problems durch viele Iterationen entwickelt .

i. Eigenschaften der evolutionären Algorithmen

Folgende sind die Hauptmerkmale der EA:

- Eine Zielfunktion.
- Eine - so genannte - Bevölkerung von mögliche Lösungen.
  Eine Menge von Darstellungen der Lösungen (Elemente der Domäne der Zielfunktion) des Problems, die mittels der Algorithmus und mehrere Iterationen (auch „Generationen“ genannt) verarbeitet und hoffentlich verbessert werden.
- Vorgänge, die die Elementen der Bevölkerung betreffen und verändern.

Die meisten EA sind aus der Natur inspiriert, das heißt, dass Wissenschaftler Ereignisse, Systeme, und Mechanismen der Natur beobachten und danach versuchen einigermaßen, sie mit Algorithmen zu reproduzieren. EA werden „evolutionär“ genannt, denn sie sind ursprünglich auf die Evolution basiert. [Hol75]

ii. Eigenschaften der Intelligenz

Wie vorher gesagt, EA sind als Teil des KI betrachtet, weil sie Eigenschaften intelligenter natürlicher Systeme emulieren.

Nach [Sim13] sind die folgende Eigenschaften notwendig, um ein System als intelligent zu betrachten:

- Adaptation.
  Ein intelligentes System muss in hohem Maße an unvorhersehbare Änderungen anpassbar sein. Lernen ist daher unerlässlich.
- Zufälligkeit.
  Obwohl Zufälligkeit meistens als eine schlechte Eigenschaft betrachtet wird, es ist in gewissem Maße notwendig, damit neue Lösungen eines Problems gefunden sein können.
- Kommunikation.
  Zwar ist eine einzige Ameise nicht wirklich als intelligent beurteilt, aber eine Ameisenkolonie ist (dank Kommunikation durch Pheromone) als eine super-intelligent Einheit überlegt.
- Rückmeldung.
  Ein System kann sich nicht anpassen und verbessern, wenn es seine Umgebung nicht erkennen und darauf reagieren kann. Auch um zu lernen, muss ein System seine Fehler (mittels Rückmeldung) erkennen.
- Erkundung.
  Neue Lösungen finden, neue Wege erkunden, neue Ideen erzeugen. Am meistens eng mit Zufälligkeit verbindet.
- Ausbeutung.
  Im Gegenteil zu Erkundung, Ausbeutung heißt, schon bekannte Lösungen und Wege (oder Kenntnisse) ausnutzen.

In nächster Sektion werden verschiedene Beispiele von EA gezeigt und in jeden Beispiel wird es erwähnt, welches Teil des Algorithmuses welche Eigenschaft des Intelligenzes versucht zu emulieren. Es ist auch wichtig zu nennen, dass nicht jede Eigenschaft des Intelligenzes von jeden EA repliziert ist.

V. EA Beispiele

Heutzutage gibt es viele EA und jedes Jahr werden neue entwickelt. In dieser Sektion werden nur ein paar der bekanntesten und wichtigsten erwähnt und oberflächlich erklärt.

i. Genetic Algorithmen - GA

Genetic Algotihmen (GA) sind ursprünglich in [Hol75] präsentiert und danach in [Gol98] weiter entwickelt. Der Name „Genetic Algorithms" bezieht sich mehr auf eine Familie von Algorithmen als auf einen einzigen Algorithmus.

GA sind die erste vorgestellte, bekannteste, und am meistens verwendete EA. Ursprünglich waren sie dafür entwickelt, um adaptierbare Systeme zu studieren. Sie sind Simulationen der natürlichen Selektion. Daher sind sie genau auf die Evolution basiert.

Es gibt sechs Hauptteile der GA:

1. Codierung der Lösungen.

   Als erstes muss man eine geeignete Codierung für die mögliche Lösungen finden. Das heißt, eine Darstellung für die Lösungen finden, die einfach zu verarbeiten ist. Eine die am meistens verwendete Codierungen ist eine Bit-Folge. In dieser Codierung jedes Bit wird „Gen" genannt.

2. Ursprüngliche Bevölkerung.

   Eine ursprüngliche Bevölkerung muss initialisiert werden. Diese Initialisierung kann entweder zufällig oder auf Vorkenntnis basiert sein. Wie viele Elementen die Bevölkerung hat, ist ein wichtiger Parameter. Je mehrere Elemente desto eine bessere Wahrscheinlichkeit, um gute Lösungen zu finden.

3. Fitnessfunktion.

   Die Zielfunktion des Problems.

   Der Wert, den die Zielfunktion ergibt, wenn sie für den Wert ausgewertet wird, den ein Element der Bevölkerung darstellt, wird die „Fitness" des Elements genannt.

4. Auswahl von Einzelnen.

   Zwei Elemente der Bevölkerung werden ausgewählt, um ein neue Element herzustellen. Das verfolgte Verfahren, um die Elemente auszuwählen, kann entweder zufällig sein, oder auf die Fitness der Elemente basiert. Die ausgewählte Elemente werden „Eltern" genannt.

5. Crossover.

   Dies ist das wichtigste Teil der GA. Es definiert wie das neue Element aus die Eltern erstellt wird. Das heißt, welche Gene wird das neue Element von jeden Elternteil erben.

6. Mutation.

   Einige zufälligerweise-ausgewählte neuerstellte Elemente werden mutiert sein. Das heißt, dass einige Bits gekippt werden. Welche Elemente werden mutiert und wie viele und welche Bits werden gekippt, kann mittels verschiedener Verfahren bzw. entscheidet werden.

Abbildung 3: *Wesentlich genetic Algorithmen pseudo-Code. Quelle: [Sim13]*

```
Parents ← {randomly generated population}
While not(termination criterion)
    Calculate the fitness of each parent in the population
    Children ← ∅
    While |Children| < |Parents|
        Use fitnesses to probabilistically select a pair of parents for mating
        Mate the parents to create children c1 and c2
        Children ← Children ∪ {c1, c2}
    Loop
    Randomly mutate some of the children
    Parents ← Children
Next generation
```

EA insgesamt sind Problem-unabhängig, das heißt, dass wenn es eine geeignete Darstellung der Lösungen und eine geeignete Zielfunktion

gibt, kann theoretisch jedes Problem mittels EA optimiert werden. Aus diesem Grund gibt es bei jeden EA ein Element der Adaptation.

In Teil 2 sowie in Teile 4 bis 6 kann Zufälligkeit verwendet werden, je nachdem welche Verfahren bzw. ausgewählt werden.

In dem Mutation-Teil, eine hoch Wahrscheinlichkeit, um Elemente zu mutieren, heißt viel Erkundung und weniger Ausbeutung, und umgekehrt.

ii. Particle Swarm Optimization - PSO

Particle Swarm Optimization (PSO) ist ursprünglich in [JK95] und in [YS98] präsentiert.

PSO ist auf menschlichen sozialen Verhalten basiert. [JK01] Genauer, PSO basiert auf der Beobachtung von Gruppen von Individuen, die zusammenarbeiten, um nicht nur ihr kollektives Verhalten in einer bestimmten Aufgabe zu verbessern, sondern auch ihr individuelles Verhalten zu verbessern.

Die Grundideen hinter PSO sind folgende:

- Es gibt eine Menge Partikeln, die sich mit einer beliebigen Geschwindigkeit durch den Suchraum bewegen.
- Der Suchraum ist in „Nachbarschaften" mit einer beliebigen gleichen Größe unterteilt.
- Jeder Partikeln speichert sowohl die beste Position, die sie im Suchraum besucht hat, als auch die beste Position, die ihre Nachbarn erreicht haben.
- Beide die Partikels beste Position und ihre Nachbarns beste Positionen beeinflussen die Partikels Geschwindigkeit.

In PSO können die Geschwindigkeiten der ersten Bevölkerung zufällig initialisiert werden. Daher haben wir ein Element der Zufälligkeit.

Da die Partikeln die beste bisher erreichte Positionen ihrer Nachbarn kennen müssen, gibt es auch Kommunikation.

Die Tatsache, dass die beste bisher erreichte Positionen die Geschwindigkeit der Partikel beeinflussen, zeigt Rückmeldung. In welche Ausmaß wird die Partikel beeinflusst, kann entweder Erkundung oder Ausbeutung fördern.

Abbildung 4: *Wesentlich PSO Algorithmus pseudo-Code. Quelle: [Sim13]*

Initialize a random population of individuals $\{x_i\}$, $i \in [1, N]$
Initialize each individual's n-element velocity vector v_i, $i \in [1, N]$
Initialize the best-so-far position of each individual: $b_i \leftarrow x_i$, $i \in [1, N]$
Define the neighborhood size $\sigma < N$
Define the maximum influence values $\phi_{1,\max}$ and $\phi_{2,\max}$
Define the maximum velocity $v_{\max}$
While not(termination criterion)
    For each individual x_i, $i \in [1, N]$
        $H_i \leftarrow \{\sigma$ nearest neighbors of $x_i\}$
        $h_i \leftarrow \arg\min_x \{f(x) : x \in H_i\}$
        Generate a random vector ϕ_1 with $\phi_1(k) \sim U[0, \phi_{1,\max}]$ for $k \in [1, n]$
        Generate a random vector ϕ_2 with $\phi_2(k) \sim U[0, \phi_{2,\max}]$ for $k \in [1, n]$
        $v_i \leftarrow v_i + \phi_1 \circ (b_i - x_i) + \phi_2 \circ (h_i - x_i)$
        If $|v_i| > v_{\max}$ then
            $v_i \leftarrow v_i v_{\max} / |v_i|$
        End if
        $x_i \leftarrow x_i + v_i$
        $b_i \leftarrow \arg\min\{f(x_i), f(b_i)\}$
    Next individual
Next generation

iii. Differential Evolution - DE

Es wurde erstmals in [Sto96a] und [Sto96b] über differential Evolution (DE) diskutiert. Aber die erste vielfach gelesene DE-Veröffentlichung war [KP97].

Im Gegensatz zu fast alle andere EA, DE ist nicht auf der Natur inspiriert. Eine die wichtigste Merkmale der DE ist, dass es einfach ist. Dies Merkmal ist wichtig, da es erlaubt Benutzer aus andere Bereiche (d. h. nicht Informatikern/inen oder Mathematiker/inen), es umzusetzen.

Die wesentliche Schritte eines DE Algorithmuses sind die folgende:

1. Die Bevölkerung eines DE Algorithmus besteht auf Vektoren mit n Elemente. Wo n ist auch die Dimension des Problems. Weiterhin, jeder Vektor selbst stellt eine mögliche Lösung des Problems dar.

2. Zwei Vektoren $x_{r2}, x_{r3} \mid r2 \neq r3$ werden ausgewählt.

3. Die Differenz zwischen x_{r2} und x_{r3} wird berechnet.

4. Die skalierte Differenz wird zu einem dritten Vektor $x_{r1} \mid r1 \notin \{r2, r3\}$ addiert. Das wird eine neue mögliche Lösung v_i ergeben.

5. v_i wird mit einem Vektor $x_i \mid i \notin \{r1, r2, r3\}$ kombiniert.

6. x_i wird gegen u_i vergleicht und nur der beste wird behaltet.

Abbildung 5 zeigt Schritte 2 bis 4 und Abbildung 6 zeigt einen ganze DE Algorithmus pseudo-Code.

Abbildung 5: *Grundlegende Idee von DE. Quelle: [Sim13]*

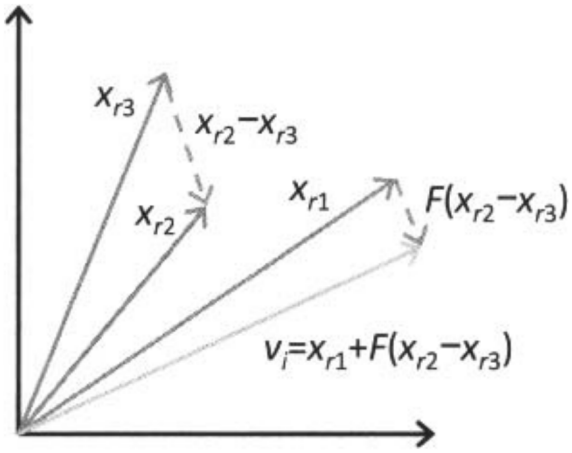


Abbildung 6: *Wesentlich DE Algorithmus pseudo-Code. Quelle: [Sim13]*

F = stepsize parameter $\in [0.4, 0.9]$
c = crossover rate $\in [0.1, 1]$
Initialize a population of candidate solutions $\{x_i\}$ for $i \in [1, N]$
While not(termination criterion)
    For each individual x_i, $i \in [1, N]$
        $r_1 \leftarrow$ random integer $\in [1, N] : r_1 \neq i$
        $r_2 \leftarrow$ random integer $\in [1, N] : r_2 \notin \{i, r_1\}$
        $r_3 \leftarrow$ random integer $\in [1, N] : r_3 \notin \{i, r_1, r_2\}$
        $v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3})$ (mutant vector)
        $\mathcal{J}_r \leftarrow$ random integer $\in [1, n]$
        For each dimension $j \in [1, n]$
            $r_{cj} \leftarrow$ random number $\in [0, 1]$
            If $(r_{cj} < c)$ or $(j = \mathcal{J}_r)$ then
                $u_{ij} \leftarrow v_{ij}$
            else
                $u_{ij} \leftarrow x_{ij}$
            End if
        Next dimension
    Next individual
    For each population index $i \in [1, N]$
        If $f(u_i) < f(x_i)$ then $x_i \leftarrow u_i$
    Next population index
Next generation

Der Algorithmus, den die Abbildung 6 beschreibt, ist „DE/rand/1/bin“ genannt und es ist die einfachste Variante der DE Algorithmen. Andere Varianten sind z. B. „DE/best/1/L“, „DE/best/2/bin“, „DE/rand/1/exp“, u. v. a. m.

Jedes Teil dieser Namen bezieht sich auf die Verfahren, die für bestimmte Schritte der DE Algorithmen verwendet werden.

Zum Beispiel der „rand/best“-Teil[2] beschreibt wie Vektoren x_{r1} und x_{r2} im 2. Schritt ausgewählt werden. „rand“ bedeutet, dass die Vektoren zufälligerweise ausgewählt werden, während „best“ heißt, dass die beste Vektoren eine höher Wahrscheinlichkeit haben, um ausgewählt zu sein.

In [EMM06] werden die Leistungen mehrerer Varianten von DE Algorithmen vergleicht.

Wenn Vektoren x_{r1} und x_{r2} zufällig ausgewählt werden, gibt es ein Element der Zufälligkeit. Wenn nur die beste Vektoren ausgewählt werden, wird Ausbeutung gefördert. Die Tatsache, dass nur die beste Vektor im 6. Schritt behaltet wird, zeigt Rückmeldung. Die Kombination, die im 5. Schritt stattfindet, könnte auch als Rückmeldung und Ausbeutung beurteilt werden.

VI. Diskussion

Todo

VII. Fazit

Todo

Quellenverzeichnis

[EMM06] Carlos A. Coello Coello Efrén Mezura-Montes, Jesús Velázquez-Reyes. A comparative study of differential evolution variants for global optimization. *GECCO '06 Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 485 – 492, 2006. Seattle, Washington, USA.

[Gol98] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine*

[2] „best“ und „rand“ sind nicht die einzige Varianten.

Learning. Reading. MA: Addison-Wesley Professional, 1998.

[Hol75] John Henry Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[JK95] Russell C. Eberhart James Kennedy. *Particle Swarm Optimization*, volume IV. Proceedings of IEEE International Conference on Neural Networks, 1995. pp. 1942 - 1948.

[JK01] Russell C. Eberhart James Kennedy. *Swarm intelligence*. Morgan Kaufmann Publishers Inc., 2001.

[KP97] Rainer Storn Kenneth Price. Differential evolution: Numerical optimization made easy. *Dr. Dobb's Journal*, pages 18 – 24, 1997.

[Sim13] Dan Simon. *EVOLUTIONARY OPTIMIZATION ALGORITHMS - Biologically-inspired and Population-based Approaches to Computer Intelligence*. John Wiley and Sons, Inc., 2013.

[Sto96a] Rainer Storn. Differential evolution design of an iir-filter. *IEEE Conference on Evolutionary Computation*, pages 268 – 273, 1996. Nagoya, Japan.

[Sto96b] Rainer Storn. On the usage of differential evolution for function optimization. *Conference of the North American Fuzzy Information Processing Society*, pages 519 – 523, 1996. Berkeley, California.

[wika] Brute-force-methode. `https://de.wikipedia.org/wiki/Brute-Force-Methode`. Accessed: 05.01.2019.

[wikb] Optimierung. `https://de.wiktionary.org/wiki/Optimierung`. Accessed: 03.01.2019.

[wikc] Problem des handlungsreisenden. `https://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden`. Accessed: 03.01.2019.

[YS98] Russell C. Eberhart Yuhui Shi. *A modified particle swarm optimizer*. Proceedings of IEEE International Conference on Evolutionary Computation, 1998. pp. 69 - 73.