

Nature-Inspired Algorithms

by

Yves Beutler

Erstellt an der BFH Technik und Informatik
in teilweiser Erfüllung der Anforderungen an den Titel

Bachelor of Science BFH in Informatik

an der

BERNER FACHHOCHSCHULE

Juni 2019

© Yves Beutler, 2019. All rights reserved.

Autor.....

Yves Beutler
BFH Technik und Informatik
10. Juni, 2019

Betreut durch

Dr. Mascha Kurpicz-Briki
Betreuende Professorin

Nature-Inspired Algorithms

by

Yves Beutler

Erstellt an der BFH Technik und Informatik
am 10. Juni, 2019, in teilweiser Erfüllung der
Anforderungen an den Titel
Bachelor of Science BFH in Informatik

Abstract

Nature-Inspired Algorithms können dort eingesetzt werden, wo traditionelle Problemlösungsmethoden nicht funktionieren. Einige dieser Algorithmen haben sich als äusserst performant und robust erwiesen und werden heute zur Problemlösung in den unterschiedlichsten Anwendungsgebieten eingesetzt. Zu der Gruppe der Nature-Inspired Algorithms gehören beispielsweise Evolutionäre-Algorithmen, welche starke Ähnlichkeiten zu Darwins Evolutionstheorie mit seinem bekannten Zitat von 'Survival of the fittest aufweisen'. Weiter werden Schwarm- und Neuronale-Algorithmen thematisiert, wobei aktuell die letzteren, durch die neusten Grafikkartengenerationen herbeigeführt, einen erneuten Aufschwung erleben dürfen. Nach einer kurzen Übersicht der einzelnen Gruppen wird jede behandelte Unterart der Nature-Inspired Algorithms durch einen ausgewählten Algorithmus im Detail veranschaulicht.

Ziel dieser Arbeit ist es, eine Einführung in die Thematik von Nature-Inspired Algorithms und ihren Einsatzmöglichkeiten zu geben und die bekanntesten Unterarten mit ausgesuchten Algorithmen im Detail zu erläutern.

Betreuende Professorin: Dr. Mascha Kurpicz-Briki

Danksagung

Special thanks to Dr. Mascha Kurpicz-Briki for supporting me during the semester while I was working on this term paper and Natalie Kuster for providing me with additional information about genetic algorithms and especially data science related topics I covered in the next few pages.

Another thank you to Kieran Willis for proofreading my term paper.

Inhaltsverzeichnis

1	Einführung	7
1.1	Anwendungsgebiete	8
1.1.1	Optimierungen	8
1.1.2	Approximationen	8
2	Evolutionäre Algorithmen	9
2.1	Genetische Algorithmen	9
2.1.1	Initiale Population	10
2.1.2	Selektion	10
2.1.3	Kreuzung	11
2.1.4	Mutation	11
2.1.5	Terminierung	11
2.1.6	Beispiel Implementation	12
2.1.7	Vorteile	14
2.2	Genetic Programming	14
2.2.1	Darstellungsformen	14
2.2.2	Anwendungsfälle	15
2.3	Neuroevolution	15
2.4	Weitere Algorithmen	15

List of Figures

2-1	Population, Chromosome und Gene	10
2-2	Einfaches mathematisches Programm als Tree dargestellt	15

List of Tables

Kapitel 1

Einführung

Seit dem Anbeginn unseres Planeten sah sich die Natur mit mehr oder weniger komplexen Problemstellungen konfrontiert, um das Überleben ihrer Bewohner sicherzustellen. Der Schlüssel zum Erfolg ist es, sich am besten auf die naturgegebenen Umstände anzupassen. Ohne den Einklang mit der Natur wären die Überlebenschancen einer jeden Spezies schwindend gering wenn nicht gar inexistent. Seit Jahrzehnten adaptieren Menschen die Techniken aus der Natur und ihren Geschöpfen um moderne Technologien auf ein neues Level zu befördern. Flugzeuge werden nach den Flügelcharakteristiken von Zugvögeln konstruiert und neue Klebstoffe werden durch das Studium von Geckofüssen mit ihren unglaublichen Hafteigenschaften entworfen. [Cro14]

Wir dürfen nicht ausser Acht lassen, dass die Natur Millionen von Jahre in die Forschung nach dem Schlüssel zum Überleben investierte und in den meisten Fällen scheiterte. Doch die Natur gab nicht einfach auf, sondern justierte bestimmte Parameter um über die Zeit zunehmend besser zu werden. Durch sich ständig ändernde Umstände sucht sie auch heute noch nach Lösungen für zukünftige Spezies.

Wieso sollte die Menschheit nicht selbst von diesen Millionen von Jahren an Forschung profitieren?

1.1 Anwendungsgebiete

Nature-Inspired Algorithms werden meist dort eingesetzt, wo traditionelle Algorithmen keine oder keine brauchbaren Ergebnisse liefern können. Folgende Gegebenheiten können Gründe sein, um auf Nature-Inspired Algorithms zurückzugreifen:

1. Die Anzahl möglicher Lösungen im Suchraum ist so immens gross, dass die Suche nach der bestmöglichen Lösung viel zu aufwändig wäre.
2. Die Komplexität der Problemstellung ist derart hoch, dass nur vereinfachte Modelle bei der Problemlösung zum Einsatz kommen und dadurch keine aussagekräftigen Lösungen entstehen.
3. Die Evaluierungsfunktion zur Bewertung einer möglichen Lösung variiert mit der Zeit, so dass nicht nur eine sondern eine Vielzahl von Lösungen erforderlich ist.

[Bro11, Kap. 1.2]

1.1.1 Optimierungen

Häufige Anwendungsfälle dieser Algorithmen sind Optimierungen von Funktionen. Bei Optimierungen handelt es sich in der Regel um die Suche nach einer Parameterkombination für eine gegebene Funktion f um eine Kostenfunktion zu minimieren oder eine Wertefunktion zu maximieren.

1.1.2 Approximationen

Die Approximation beschreibt meist eine Funktion f , welche möglichst nahe an eine Zielfunktion angenähert werden möchte. Die approximierte Funktion f wird aus einem Set an Beobachtungen¹ generiert. Solche Approximationen werden häufig für Image Recognition verwendet und spielen ebenfalls eine wichtige Rolle bei der Klassifikation und dem Clustering von grossen Datenmengen.

¹oftmals auch als *training set* aus der Data Science bekannt

Kapitel 2

Evolutionäre Algorithmen

Diese Subkategorie von Algorithmen wurde durch den natürlichen Prozess der Evolution inspiriert. Charles Darwin begründete 1838 den Evolutionsprozess und das daraus resultierende Überleben der jeweils am besten angepassten Individuen mit der natürlichen Selektion. Diese stellt sicher, dass sich gut an den Lebensraum angepasst Lebewesen in der Natur durchsetzen können, währenddessen weniger gut angepasste Individuen aussterben. Zudem arbeitet die Natur nach dem 'Trial and Error' Prinzip. Dadurch werden Mutationen an Lebewesen ausprobiert und bei positivem Effekt auf das Überleben der Spezies beibehalten, bei einer Verminderung der Überlebenschancen jedoch wieder rückgängig gemacht. Dieser Prozess ist sehr zeitaufwändig, jedoch nachhaltig wirksam. Viele Algorithmen wie beispielsweise der Hill Climbing Algorithmus wenden ebenfalls dieses Prinzip an.

2.1 Genetische Algorithmen

Die Genetischen Algorithmen sind die bekanntesten Vertreter aus der Gruppe der Evolutionären Algorithmen. Sie orientierten sich an der Populationsgenetik und insbesondere an den Mendelschen Regeln. [McC00] Bei Genetischen Algorithmen werden häufig Begriffe aus der Biologie verwendet. Die Anzahl an möglichen Lösungen nennt man Population. Eine einzelne Lösungsvariante wird als Chromosom bezeichnet, wohingegen diese wiederum aus einer Kombination an Variablen, in der Biologie

Gene genannt, bestehen. Wie aus der Grafik 2-1 zu entnehmen ist, besteht ein Chromosom A_n aus mehreren Genen.

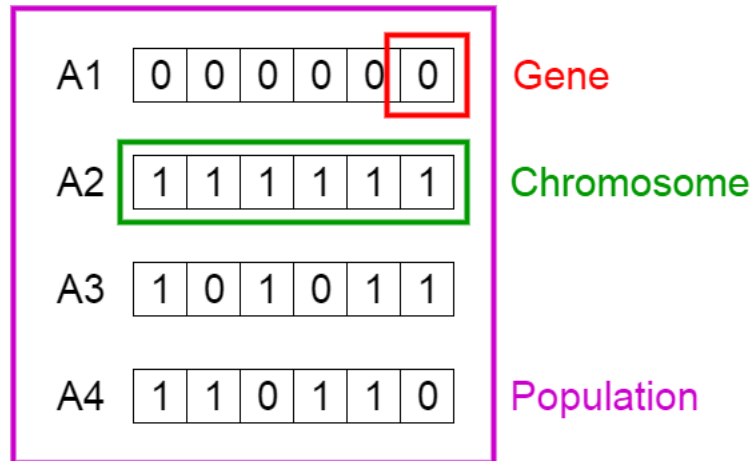


Figure 2-1: Population, Chromosome und Gene

Im folgenden werden die einzelnen Schritte erläutert, welche die Natur wie auch die Genetischen Algorithmen durchlaufen, um eine Anfangspopulation schrittweise weiterzuentwickeln. Die Natur sucht nach der erfolgsversprechendsten Lösung zum Überleben, wohingegen die Algorithmen optimale Lösungen für ganz unterschiedliche Problemstellungen suchen. [Mal17]

2.1.1 Initiale Population

Eine Population besteht aus mehreren unabhängigen Lösungen, im biologischen Kontext Chromosomen, um eine bestimmte Problemstellung zu lösen. Bei den meisten Implementationen von Genetischen Algorithmen wird ein Gen durch einen String repräsentiert. Bei Chromosomen eignen sich auch Binärwerte um das Vorhandensein eines Gens auszuzeichnen.

2.1.2 Selektion

Anhand einer sogenannten Fitnessfunktion wird für jedes Individuum eine Punktzahl berechnet. Die Punktzahl gibt darüber Auskunft, wie kompetitiv sich ein Individuum

gegenüber einem anderen verhält. Je höher dieser Wert, desto höher sind die Chancen, das besagte Individuum von der Selektion für die Reproduktion verwendet wird. Durch die Selektion können nur die fittesten Individuen ihre Gene an die nächste Generation weitergeben und den schwächeren Individuen wird der Reproduktionszyklus verwehrt.

2.1.3 Kreuzung

Der womöglich zentralste Bestandteil eines Genetischen Algorithmus ist die Kreuzung. Hier werden aus den zuvor selektierten Individuen, einer Ansammlung der vielversprechendsten Lösungen, zwei Individuen zur Paarung ausgewählt. Die beiden Ausgewählten tauschen ihre Gene bis zu einem bestimmten Punkt, dem sogenannten Crossover Point, aus. Die restlichen Gene hinter diesem Punkt werden nicht verändert. Durch diesen Prozess entstehen gleich zwei neue Individuen.

2.1.4 Mutation

Die Natur nutzt Mutationen aus, um die Artenvielfalt innerhalb einer Population zu gewährleisten. Trotz der geringen Wahrscheinlichkeit kommt es vor, dass einzelne Gene, welche durch die Kreuzung eigentlich vorhanden wären, nicht mehr vorhanden sind und umgekehrt. Ein Genetischer Algorithmus flippt beispielsweise einzelne Variablen seiner Nachkommen. Durch die Mutation wird verhindert, dass unser Algorithmus in einem lokalen Minimum stecken bleibt.

2.1.5 Terminierung

Ein Genetischer Algorithmus terminiert, sobald sich die nächste Generation nicht mehr merklich von ihren Vorgängern unterscheidet. Man spricht oftmals auch von Konvergenz. Es ist jedem Algorithmus selbst überlassen, wie lange ein Individuum in der Population überlebt. Häufig werden bei jedem Reproduktionszyklus ein oder mehrere Individuen mit den niedrigsten Fitnesswerten aus der Population gestrichen. In der Natur übernimmt der Tod diese Funktion.

2.1.6 Beispiel Implementation

Die einzelnen Schritte eines Genetischen Algorithmus sind in Listing 2.1 durch ein Python-Beispiel verdeutlicht. Die verwendete Fitnessfunktion ist sehr rudimentär gehalten und berechnet lediglich die Summe der vorhandenen Gene. Die Population ist konvergiert, wenn ein Individuum den Fitnesswert 5 erreicht, sprich alle Gene vorhanden sind.

Zweck dieses Listings ist es, den Ablauf eines Genetischen Algorithmus verständlich darzustellen. Für eine bessere Übersicht wurden dazu uninteressante Teile des Codes gänzlich weggelassen:

```
1 class Individual:
2     def __init__(self, genes):
3         self.genes = genes
4         self.fitness_function(genes)
5
6     # calculates fitness of each individual
7     def fitness_function(self, genes):
8         score = 0
9         for gene in genes:
10             if gene == 1:
11                 score += 1
12         self.fitness = score
13
14 class Population:
15     def __init__(self, individuals):
16         self.individuals = individuals
17
18     def crossover(self, parent_1, parent_2, cross_value=3):
19         offspring_1 = parent_1
20         offspring_2 = parent_2
21
22         # exchange genes below crossover point
23         for i in range(cross_value-1):
```

```

24         gen = offspring_1.genes[i]
25         offspring_1.genes[i] = offspring_2.genes[i]
26         offspring_2.genes[i] = gen
27
28     return [offspring_1, offspring_2]
29
30 def genetic_algorithm():
31     generation = 1
32     population = Population([
33         Individual([0, 1, 0, 1, 0]), Individual([1, 1, 0, 0, 0]),
34         Individual([0, 1, 1, 1, 0]), Individual([0, 0, 0, 1, 1]),
35         Individual([1, 0, 0, 0, 1]), Individual([1, 0, 1, 1, 0])])
36
37     while population.get_fittest().fitness < 5:
38         # selection
39         parent_1 = population.get_fittest()
40         parent_2 = population.get_fittest()
41
42         # crossover
43         offspring = population.crossover(parent_1, parent_2)
44
45         # mutation
46         offspring = population.mutate(offspring)
47
48         # add offspring, remove weakest individuals
49         population.grow(offspring)
50         population.kill_weakest()
51         generation += 1
52
53     print(f'The population converged after {generation} generations')
```

Listing 2.1: Genetischer Algorithmus (Python)

2.1.7 Vorteile

Genetische Algorithmen haben gewisse Vorzüge bei der Lösung von komplexen Problemstellungen gegenüber traditionellen Verfahren. Durch den modularen Aufbau mit den drei Hauptfunktionen Selektion, Kreuzung und Mutation, lassen sich diese Algorithmen einfach parallelisieren, was sich in einer besseren Rechenzeit bemerkbar macht. Weiter generieren sie nicht nur eine Lösung sondern eine Vielzahl an möglichen Lösungen, welche mit der Zeit immer bessere Ergebnisse liefern. [Gou19]

2.2 Genetic Programming

Eine weitere Unterart der Evolutionären Algorithmen ist Genetic Programming. Ähnlich wie bei den Genetischen Algorithmen ist Genetic Programming auch von den Mendelschen Regeln inspiriert worden. Die Population besteht jedoch nicht wie bisher gezeigt aus einzelnen Lösungen, sondern aus Programmen, welche eine Aufgabenstellung in unterschiedlicher Qualität lösen können. Ziel ist es, dass diese Programme durch den Vererbungsprozess von Generation zu Generation besser auf die jeweilige Problemstellung angepasst werden. [Sta19a]

2.2.1 Darstellungsformen

Es stellt sich die Frage, wie eine Applikation repräsentiert werden kann, um anschließend von Genetic Programming optimiert werden zu können. Es existieren nebst Tree- und Stack-basierten eine Lineare sowie eine Kartesische Darstellungsform. Im folgenden wird die Tree-basierte Form genauer erläutert.

Wie in Grafik 2-2 erkennbar ist, werden die einzelnen Operationen in der Tree-basierten Darstellungsweise durch Knoten repräsentiert. Die Operanden sind dabei jeweils die Endknoten. Der Vorteil von einem Tree ist, dass mittels Rekursion sehr simpel über alle Äste iteriert werden kann. Bei der Kreuzung innerhalb des Genetic Programming Algorithmus werden beispielsweise einzelne Äste zweier Bäume miteinander kombiniert um einen komplett neuen Baum zu kreieren. [Sta19b]

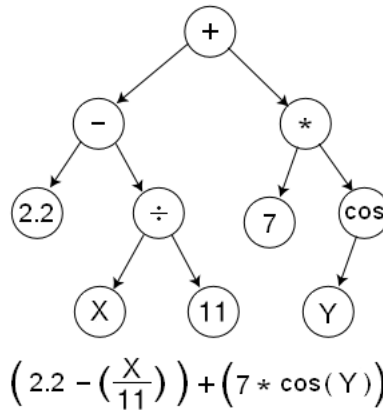


Figure 2-2: Einfaches mathematisches Programm als Tree dargestellt

2.2.2 Anwendungsfälle

Genetic Programming wird häufig als Machine Learning Tool eingesetzt. Es ist besonders nützlich, wenn eine Approximation der finalen Lösung genügt, da die Berechnung einer exakten Lösung zu aufwändig wäre. Ausserdem hilft es in Fällen, wo die genaue Form der Lösung zu Beginn unbekannt ist. Häufige Einsatzgebiete sind Data Modeling, Feature selection und Klassifikation.

Es existieren unzählige Beispiele, in denen mit Genetic Programming gleichwertige, wenn nicht sogar bessere Ergebnisse erzielt wurden als durch einen Menschen. Das Entwerfen von elektrischen Schaltplänen, KI in Computerspielen, Bilderkennung oder automatisiertes Bugfixing stellen nur einen kleinen Teil der Einsatzmöglichkeiten dar. [Koz10]

2.3 Neuroevolution

Hier könnte man bspw. NEAT erklären

2.4 Weitere Algorithmen

Weitere Unterarten von Evolutionären Algorithmen sind beispielsweise: - evolution strategy - neuroevolution (NEAT)

Bibliography

- [Bro11] Jason Brownlee. *Clever Algorithms: Nature-Inspired Programming Recipes*. LuLu, January 2011. <http://www.cleveralgorithms.com>.
- [Cro14] Alfred Crosby. Geckskin - Gecko-Like Adhesives, 2014. University of Massachusetts Amherst, <https://geckskin.umass.edu> (Zugriff am 20.04.2019).
- [Gou19] Rinu Gour. Python Genetic Algorithms With Artificial Intelligence, March 2019. <https://medium.com/@rinu.gour123/python-genetic-algorithms-with-artificial-intelligence> (Zugriff am 17.05.2019).
- [Koz10] John Koza. *Human-competitive results produced by genetic programming*. Springer US, May 2010. <https://doi.org/10.1007/2Fs10710-010-9112-3>.
- [Mal17] Vijini Mallawaarachchi. Introduction to Genetic Algorithms, July 2017. <https://towardsdatascience.com/introduction-to-genetic-algorithms-e396e98d8bf3> (Zugriff am 04.05.2019).
- [McC00] Phillip McClean. Mendel's First Law of Genetics (Law of Segregation), 2000. <https://www.ndsu.edu/pubweb/~mcclean/plsc431/mendel/mendel1.htm> (Zugriff am 16.05.2019).
- [Sta19a] Kai Staats. Generational GP Algorithm, 2019. <http://geneticprogramming.com/about-gp/gp-workflow> (Zugriff am 17.05.2019).
- [Sta19b] Kai Staats. Tree-based Genetic Programming, 2019. <http://geneticprogramming.com/about-gp/tree-based-gp> (Zugriff am 17.05.2019).