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

Autor .....
Yves Beutler
BFH Technik und Informatik
10. Juni, 2019

Betreut durch .....
Dr. Mascha Kurpicz-Briki
Betreuende Professorin

Abstract

Nature-Inspired Algorithms können dort eingesetzt werden, wo traditionelle Problemlösungsmethoden nicht funktionieren, etwa bei Optimierungsproblemen mit einer viel zu grossen Anzahl an möglichen Lösungen oder bei Problemen mit einer zeitabhängigen Bewertungsfunktion der Lösungen. 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 und seiner These "Survival of the fittest" aufweisen.

Nebst den Evolutionären Algorithmen werden Schwarm-Algorithmen thematisiert, welche sich an einer Kollektiven Intelligenz bereichern, um komplexe Problemstellungen zu lösen. Im Tierreich überleben viele primitive Tierarten nur durch einen Tierverbund. Nach einer kurzen Einführung in die einzelnen Gruppen wird jede behandelte Unterart der Nature-Inspired Algorithms durch ausgewählte Algorithmen 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. Nebst den hier thematisierten Evolutionären- und Schwarm-Algorithmen existiert eine Vielzahl an interessanten Untergruppen wie etwa den Neuronalen- oder den Physikalischen Algorithmen.

Betreuende Professorin: Dr. Mascha Kurpicz-Briki

Danksagung

Einen ganz besonderen Dank an Frau Dr. Mascha Kurpicz-Briki für Ihre Unterstützung während dem Semester bezüglich Umfang und Struktur dieser Seminararbeit. Weiter möchte ich mich bei Natalie Kuster für das passende Informationsmaterial zum Thema Genetische Algorithmen bedanken. Für die Beantwortung aller meiner Fragen rund um diese Thematik bin ich Ihr natürlich ebenfalls sehr dankbar. Zu guter letzt noch ein grosses Dankeschön an mein Fehlerteufelchen Corina für Ihre Korrekturlesung und Ihre ständig motivierende Art.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

Seit 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 an 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. Häufig scheitern altbekannte Lösungsmodelle wenn mit riesigen Datenmengen gearbeitet wird. Folgende Gegebenheiten können weitere 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]

Eine weitverbreitete Eigenschaft von Nature-Inspired Algorithms ist die Kombination aus Regeln und Zufälligkeiten, um natürliche Phänomene zu imitieren [NH15]. Die Einsatzmöglichkeiten scheinen unendlich zu sein. Häufig werden solche Algorithmen dort eingesetzt, wo für die Problemstellung zwar bereits Lösungen existieren, diese aber noch viel Verbesserungspotential hinsichtlich Zeitdauer, Ressourcenverbrauch oder Genauigkeit aufweisen.

Konkrete Anwendungsgebiete werden bei den entsprechenden Ausprägungen der einzelnen Algorithmen im Detail behandelt.

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 Parame-

terkombination 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 Beobachtungen[1] generiert. Solche Annäherungen werden häufig für Image Recognition verwendet und spielen ebenfalls eine wichtige Rolle bei der Klassifikation und dem Clustering von grossen Datenmengen.

[1] oftmals 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 durch die natürliche Selektion [Wikd]. 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 [Anr18] 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.

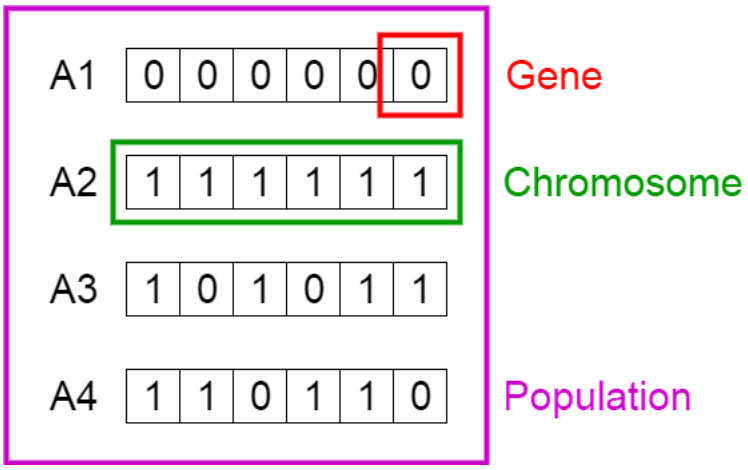

Abbildung 2-1: Population, Chromosome und Gene [Mal17]

2.1.1 Funktionsweise

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 erfolgversprechendsten Lösung zum Überleben, wohingegen die Algorithmen optimale Lösungen für ganz unterschiedliche Problemstellungen suchen. [Mal17]

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.

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 besagtes 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.

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.

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.

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.

2.1.2 Beispiel Implementation

Die einzelnen Schritte eines Genetischen Algorithmus sind in Listing A.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 die uninteressanten Teile des Codes gänzlich weggelassen.

2.1.3 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]

2.1.4 Anwendungsgebiete

Es existieren schier endlose Einsatzmöglichkeiten für Genetische Algorithmen. Sie werden unter anderem zum Trainieren von Neuronalen Netzwerken eingesetzt, zur Berechnung von Fahrplänen wie auch den zu fahrenden Routen (analog dem Traveling Salesman Problem), zur Konstruktion von Aerodynamischen Fahrzeugchassis in der Luftfahrt, in der Automobilbranche und häufig auch in der Finanzwirtschaft wie beispielsweise zur dynamischen Preisberechnung [Tut].

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.

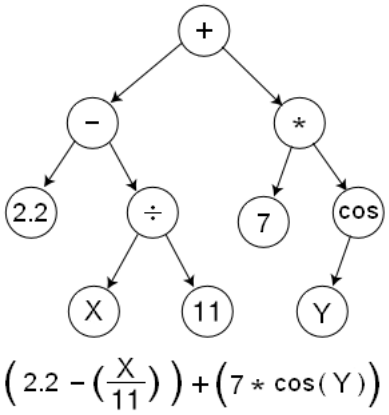

Abbildung 2-2: Einfaches mathematisches Programm als Tree dargestellt [Wike]

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]

2.2.2 Anwendungsgebiete

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 Weitere Algorithmen

Nebst den beiden behandelten Klassen von Genetischen Algorithmen und dem Genetic Programming existieren noch weitere Unterarten von Evolutionären Algorithmen. Diese werden hier lediglich aufgelistet, da eine detaillierte Differenzierung zu den beiden bereits bekannten Algorithmen aus dieser Gruppe den Rahmen dieser Seminararbeit sprengen würde. Weitere Evolutionäre Algorithmen sind:

- Evolution Strategies
- Grammatical Evolution
- Learning Classifier System

Kapitel 3

Schwarm-Algorithmen

In der Natur ist es nicht ungewöhnlich, dass sich eine Vielzahl an unabhängigen Individuen derselben Spezies zu einer Gruppe zusammenschlisst, um überlebenswichtige Aufgaben wie Nahrungssuche, Umsiedlung oder Fortpflanzung erfolgreich zu meistern. Diese Tierverbände kennen wir unter vielen verschiedenen Bezeichnungen: Herde, Rudel, Kolonie, Rotte, Kultur, Schule, Sippe oder Schwarm. Ein Schwarm ist der Inbegriff für eine riesige Ansammlung, zum Beispiel Heringe im Atlantik oder ein Meer aus Monarchfaltern auf ihrem Weg quer durch Mittelamerika.

Was alle Schwärme auszeichnet ist ihre Schwarmintelligenz. Durch die Kooperation jedes einzelnen, unabhängigen Individuums wird eine kollektive Intelligenz geschaffen, um auch mit scheinbar unlösbaren Aufgaben fertig zu werden. Dabei besitzen die einzelnen Individuen bei weitem nicht einen Bruchteil des Denkvermögens, welches durch den Verbund geschaffen wird. Das durch den Schwarm aufgebaute Wissen wird durch seine einzelnen Individuen oder die jeweilige Umgebung präserviert [Bro11, Kap. 6.1].

Aufgrund der relativ einfachen Verständnisweise sowie der hohen Effizienz sehen Computerwissenschaftler grosses Potential in Schwarm-Algorithmen. Daher ist es auch nicht verwunderlich, beteiligen sich seit den 2000er Jahren viele Wissenschaftler an der Erforschung von Schwarmintelligenz [LL18].

3.1 Ant System

Der italienische Wissenschaftler Marco Dorigo hat sich mit dem Schwarmverhalten von Ameisenkolonien bei ihrer täglichen Futtersuche beschäftigt und 1991 den ersten Ameisenalgorithmus namens Ant System vorgestellt [Wika].

Sobald Ameisen auf ihrer Futtersuche Nahrung gefunden haben und diese zurück zum Nest transportieren, beginnen sie, ein bestimmtes Pheromon auszusondern. Der Duftstoff legt sich entlang des zurückgelegten Weges Richtung Nest ab. Anfänglich schwärmen Ameisen in alle möglichen Richtungen aus. Kürzere Wege zwischen Futterquelle und Nest sind von den Ameisen stärker frequentiert und weisen eine höhere Pheromonkonzentration auf als längere Alternativrouten. Zudem neigen Ameisen bei der Wahl des Wegs dazu, den stärker duftenden Weg zu wählen. Dadurch kreiert die Ameisenkolonie durch ihre Schwarmintelligenz einen sehr kurzen Pfad. Diesen Weg nehmen wir Menschen meist als Ameisenstrasse wahr [Bro11, Kap. 6.3].

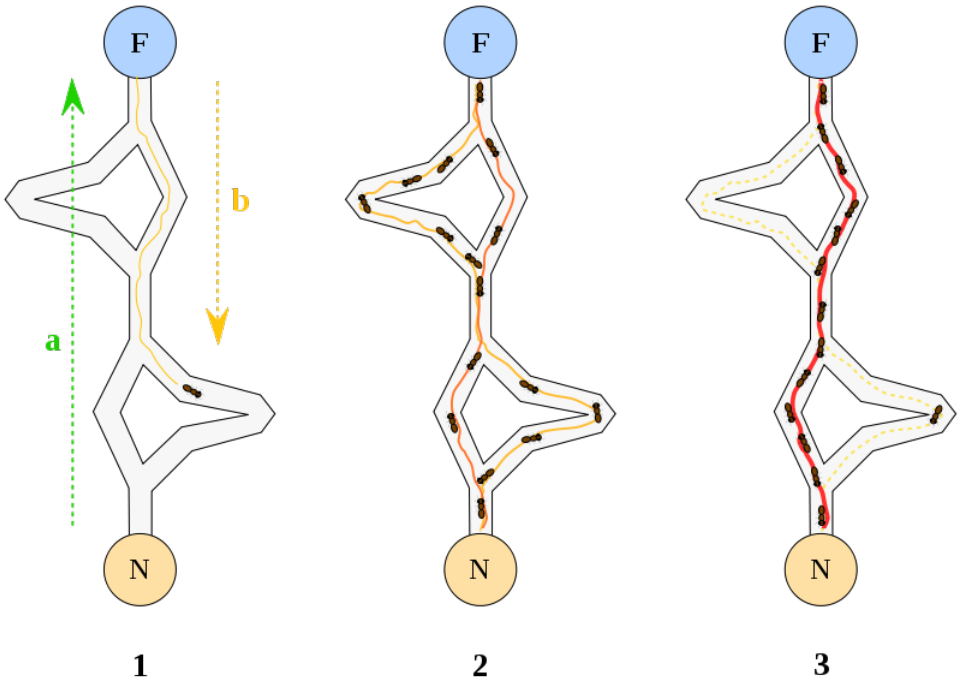

Abbildung 3-1: Funktionsweise des Ant Systems [Wika]

In Grafik 3-1 können wir erkennen, dass die Ameisen nach einer gewissen Zeit die Strecken mit den höchsten Duftstoffkonzentrationen bevorzugen.

3.1.1 Funktionsweise

Möchte man den Ant System Algorithmus implementieren, müssen bestimmte Parameter berücksichtigt werden. Die Wahrscheinlichkeiten, dass sich eine Ameise am Knotenpunkt i entscheidet nach j zu krabbeln, kann durch Formel 3.1 berechnet werden [Blu03].

$$P_{ij}(t) = \begin{cases} \frac{[T_{ij}(t)]^{\alpha} \cdot [N_{ij}]^{\beta}}{\sum_{j \in allowed} [T_{ij}(t)]^{\alpha} \cdot [N_{ij}]^{\beta}} & \text{wenn } j \in allowed \\ 0 & \text{sonst} \end{cases} \tag{3.1}$$

Der interessanteste Part dieser Formel stellt die Pheromonverteilung [3.2] dar. Sie liefert den Wert für die Pheromonkonzentration zwischen Knoten i und j. Die Pheromonintensität ergibt sich aus dem vorgängigen Durchlauf $(t-1)$ und den aktuell durchgelaufenen Ameisen $\Delta T_{ij}(t, t+1)$. Sie unterliegt einem zeitlichen Zerfall, auch Evaporation genannt. Dieser Koeffizient ρ stellt sicher, dass die Pheromonkonzentration nicht ins Unendliche ansteigen kann; Dementsprechend sollte auch $\rho < 1$ sein. Die Pheromonwerte jeder Teilstrecke werden nach jedem Durchgang mittels (3.2) neu berechnet.

$$T_{ij}(t+1) = \rho \cdot [T_{ij}(t)] + \Delta T_{ij}(t, t+1) \tag{3.2}$$

Nebst der Pheromonverteilung existiert noch eine weitere Komponente, nämlich die Sichtbarkeit der einzelnen Knoten N_{ij}. Ameisen richten sich nicht nur nach der Duftintensität der Strecke, sondern bevorzugen nebenbei auch kürzere Wege. Die Sichtbarkeit kann ganz simpel als $\frac{1}{d_{ij}}$ berechnet werden. Dieser Ansatz ist Greedy [Wikf] und könnte auch durch eine ausgereiftere Heuristik ersetzt werden.

Um nicht die gleichen Knoten mehrfach zu besuchen, wird eine Tabu-Liste eingesetzt. Diese Liste existiert für jede einzelne Ameise und wird in jedem Durchlauf mit dem aktuell besuchten Knoten erweitert. Folglich berechnet sich die Liste der erlaubten Knoten aus der Differenz der Liste aller Knoten und der Tabu-Liste, in

(3.1) als *allowed* gekennzeichnet.

Durch die Parameter α und β können die Pheromonverteilung und die Sichtbarkeit gewichtet werden. Dieses Feintuning kann das Ant System zu einem reinen Greedy-Algorithmus ($\alpha = 0, \beta = 1$) oder zu einem Zufallsalgorithmus ($\alpha = 1, \beta = 0$) verändern.

3.1.2 Anwendungsgebiete

Der Ant System Algorithmus wurde zum Lösen von Kombinationsproblemen entwickelt. Bei diesem Problemtypen besteht die Schwierigkeit darin, aus mehreren Teillösungen die optimale Kombination zu finden. Die Summe an möglichen Lösungen übersteigt somit die Kompetenz von traditionellen Pfadsuch-Algorithmen. Wie anfänglich in Kapitel 1 behandelt, eignen sich Nature-Inspired Algorithmen für diese Aufgaben besonders gut. Da es sich bei diesen Problemen um heuristische Optimierungsverfahren handelt, kann der Ant System Algorithmus aber nicht garantieren, jeweils den bestmöglichen Lösungsweg zu finden [Wika].

Mit dieser Art von Kombinationsproblemen beschäftigen sich beispielsweise Logistikunternehmen, um eine optimale Transportroute zu finden. Weiter kann das Ant System auch zur Konstruktion von Halbleiterplatten oder bei der Kantendetektion in der Bildverarbeitung eingesetzt werden [Wikb].

Travelling Salesman Problem (TSP)

Das Problem des Handelsreisenden ist ein bekanntes Beispiel von kombinatorischen Optimierungsproblemen [Wikh]. Man versucht eine festgelegte Anzahl Städte zu besuchen, wobei keine Stadt bis auf die Anfangsstadt mehrfach besucht werden darf. Ziel ist es, einen optimalen Weg zu finden, bei welchem der Handelsreisende die kleinstmögliche Distanz zurücklegen muss. Das TSP gehört zu den NP-vollständigen Problemen, daher existiert kein Algorithmus, welcher den optimalen Weg in polynomieller Laufzeit bestimmen kann [Wikg].

3.1.3 Erweiterungen

Das 1991 vorgestellte Ant System wurde mehrfach weiterentwickelt, um sowohl Laufzeit, als auch Effizienz zu verbessern. Die populärsten Weiterentwicklungen werden hier kurz erläutert.

Elite Ant System (1991)

Das Elite Ant System unterscheidet sich nur geringfügig vom ursprünglichen Ant System. Es verwendet eine angepasste Pheromonverteilungsfunktion [MZ14]. Anders als beim Ant System, bei welchem jede Ameise die gleiche Menge an Pheromonen ausströmt, existieren sogenannte Elite-Ameisen, welche eine grössere Menge an Duftstoffen ausströmen und so die Kolonie bei der Pfadsuche wesentlicher beeinflussen. Der nach jeder Iteration kürzeste Pfad wird als Elitepfad angesehen und die darauf wandernden Ameisen sind folglich die Elite-Ameisen [BHS99].

Rank-based Ant System (1997)

Eine weitere Erweiterung des Ant System stellt das Rank-based Ant System dar. Nach jeder Iteration werden die Ameisen anhand von ihrem zurückgelegten Pfad der Länge nach sortiert. Ameisen mit den kürzeren Pfaden tragen anschliessend verhältnismässig mehr zur Pheromonverteilung bei als solche mit längeren. Zusätzlich werden nur die Anzahl ω besten Ameisen für die Berechnung verwendet. Dies verringert die Gefahr, dass eine hohe Pheromonkonzentration durch viele Ameisen auf dem gleichen sub-optimalen Pfad entsteht [BHS99].

Max-Min Ant System (1999)

Anders als beim Ant System legt nur die beste Ameise ihre Pheromonspur. Des Weiteren werden die Konzentrationen pro Teilpfad mit einem Maximum- und Minimum-Wert versehen. So können keine gänzlich unattraktiven und auch keine zu stark konzentrierten Pfade entstehen. Wenn die beste Ameise eine Teilstrecke ij nicht passiert hat, ist der dazukommende Pheromonwert als $\Delta T_{ij}^{\text{best}} = 0$ definiert [Dor07].

3.2 Bees Algorithm

Eine Kolonie Honigbienen schwärmt bei ihrer Suche nach Nektar in einem Radius von über 10km in alle Richtungen aus. Blumenfelder mit einer grossen Menge an Nektar oder Pollen für die Nachkömmlinge, sollen von einer grösseren Menge Bienen angeflogen werden, als kleinere Felder mit weitaus weniger Nektar oder Pollen. Sogenannte Erkundungsbienen, welche zur Klasse der Arbeiterbienen gehören, fliegen zufällige Blumenfelder an. Zurück im Bienenstock evaluieren die Erkundungsbienen die gefundenen Blumenfelder anhand verschiedener Kriterien wie Nektarmenge oder Distanz zum Bienenstock. Mit einem Wackeltanz kommunizieren die Bienen ihre Ergebnisse mit der restlichen Kolonie. Aus dem Wackeltanz können die Bienen die Richtung zum Blumenfeld, die zurückzulegende Distanz und die ungefähre Nektarmenge ableiten [Bro11]. Diese einzigartige Fertigkeit lässt die Bienenkolonie immer die gerade passende Anzahl an Arbeiterbienen an Blumenfelder in ihrem Suchradius ausschwärmen.

3.2.1 Funktionsweise

Wie in Listing A.2 zu erkennen ist, wird über eine bestimmte Anzahl Erkundungsbienen iteriert, welche zufällig in der Landschaft platziert werden. Jede so besuchte Blumenwiese wird mit einem Wert versehen, welcher Auskunft über ihre Attraktivität liefert. Dies kann mit dem Fitnesswert aus Kapitel 2-1 verglichen werden. Eine Blumenwiese steht für eine mögliche Lösung, welche jedoch noch optimiert werden kann. Solange die Abbruchbedingung noch nicht erreicht ist, werden Arbeiterbienen im Recruiting-Prozess für das Sammeln von Nektar und Pollen rekrutiert. Je attraktiver die Blumenwiese, desto mehr Bienen sendet der Bienenstock in diese Richtung aus.

Wie in Zeile 7 dargestellt, werden nur die besten Blumenwiesen weiter behandelt. Die dafür rekrutierten Bienen werden in einem gewissen Radius um die Blumenwiese platziert. Dies entspricht einer lokalen Suche. Landet nun eine Biene auf einer Blumenwiese mit besserem Rating, so übernimmt diese Biene neu die Rolle als Erkun-

dungsbiene. Wenn keine Biene eine attraktivere Blumenwiese (oder Lösung) findet, so wird der Suchradius um die initiale Blumenwiese verkleinert. Dieser Prozess nennt sich auch Neighbourhood Shrinking. Wenn die Blumenwiese inklusive dem ihr zugehörigen Suchradius keine brauchbaren Lösungen bereithält, wird das gesamte Gebiet nach einer gewissen Zahl an Iterationen komplett verlassen (Zeile 9).

Damit der Bees Algorithm nicht in lokalen Minima stecken bleibt, werden die Erkundungsbienen, welche die am wenigsten attraktiven Blumenwiesen gefunden haben, von diesen abkommandiert und abermals zufällig in der Landschaft platziert. Dadurch führt der Algorithmus neben einer lokalen auch noch eine globale Suche durch [Wikc].

3.2.2 Anwendungsgebiete

Der Bee Algorithm kann für Optimierungsprobleme wie das TSP [Wikh] verwendet werden. Bemerkenswert ist die Tatsache, dass der Algorithmus für das Load Balancing grosser Serverfarmen von Web Hosting Firmen eingesetzt wird. So werden mit dem Bees Algorithm einer Region genauso viele Server zur Verfügung gestellt, wie der Hoster entbehren kann. Dabei ist die Wichtigkeit der Region, welche hier als Blumenwiese agiert, das Hauptmerkmal zur Verteilung. Die einzelnen Server übernehmen dabei die Rolle der Bienen [You].

Appendix A

Code-Listings

A.1 Genetischer Algorithmus

```
class Individual:
  def __init__(self, genes):
      self.genes = genes
      self.fitness_function(genes)

  # calculates fitness of each individual
  def fitness_function(self, genes):
      score = 0
      for gene in genes:
          if gene == 1:
              score += 1
      self.fitness = score

class Population:
    def __init__(self, individuals):
        self.individuals = individuals

    def crossover(self, parent_1, parent_2, cross_value=3):
        offspring_1 = parent_1
        offspring_2 = parent_2

```

```
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'Population converged after {generation} generations')
```

Listing A.1: Genetischer Algorithmus (Python)

A.2 Bees Algorithm

```
for i=1 to num_of_scouts:
    scout[i] = initialize_scout()
    flower_patch[i] = initialize_patch(scout[i])

    while stop_condition=FALSE:
        recruitment()
        for i=1 to num_of_best_sites:
            flower_patch[i] = local_search(flower_patch[i])
            flower_patch[i] = site_abandonment(flower_patch[i])
            flower_patch[i] = neighbourhood_shrinking(flower_patch[i])

        for i=num_of_best_sites to num_of_scouts:
            flower_patch[i] = global_search(flower_patch[i])
```

Listing A.2: Bees Algorithm (Pseudocode) [Wikc]

Bibliografie

[Anr18] Dr. Bernhard Anrig. *BTI7282 Computer Perception and Artificial Intelligence - Searching*. October 2018. https://www.cpvrlab.ti.bfh.ch/bachelor/BTI7282/ai/private/ai/arb1/search_handout.pdf Benötigt BFH-Authorisierung; (Zugriff am 22.05.2019).

[BHS99] Bernd Bullnheimer, Richard Hartl, and Christine Strauss. A new rank based version of the ant system - a computational study. *Central European Journal of Operations Research*, 7:25–38, 01 1999.

[Blu03] Daniel Blum. *Ant Colony Optimization (ACO)*. Universität Dortmund, May 2003. http://ls11-www.cs.tu-dortmund.de/lehre/SoSe03/PG431/Ausarbeitungen/ACO.pdf.

[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).

[Dor07] M. Dorigo. Ant colony optimization. *Scholarpedia*, 2(3):1461, 2007. http://doi.org/10.4249/scholarpedia.1461.

[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.

[LL18] Siew Mooi Lim and Kuan Yew Leong. *A Brief Survey on Intelligent Swarm-Based Algorithms for Solving Optimization Problems*. IntechOpen, July 2018. https://doi.org/10.5772/intechopen.76979.

[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).

[MZ14] Omolbani Mohammadrezapour and Mohammad Javad Zeynali. *Comparison of ant colony, elite ant system and maximum – minimum ant system algorithms for optimizing coefficients of sediment rating curve (case study: Sistan river).* University of Zabol, Iran, October 2014. http://jap.haraz.ac.ir/article-1-47-en.pdf.

[NH15] Siddique Nazmul and Adeli Hojjat. *Nature Inspired Computing: An Overview and Some Future Directions.* Springer US, November 2015. https://doi.org/10.1007/s12559-015-9370-8.

[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).

[Tut] Tutorials Point: Genetic Algorithms - Application Areas. https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_application_areas.htm (Zugriff am 22.05.2019).

[Wika] Wikipedia: Ameisenalgorithmus. https://de.wikipedia.org/wiki/Ameisenalgorithmus (Zugriff am 22.05.2019).

[Wikb] Wikipedia: Ant colony optimization algorithms. https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms#Applications (Zugriff am 25.05.2019).

[Wikc] Wikipedia: Bees algorithm. https://en.wikipedia.org/wiki/Bees_algorithm#Algorithm (Zugriff am 10.06.2019).

[Wikd] Wikipedia: Charles Darwin. https://en.wikipedia.org/wiki/Charles_Darwin (Zugriff am 22.05.2019).

[Wike] Wikipedia: Genetic programming. https://en.wikipedia.org/wiki/Genetic_programming (Zugriff am 22.05.2019).

[Wikf] Wikipedia: Greedy Algorithm. https://en.wikipedia.org/wiki/Greedy_algorithm (Zugriff am 25.05.2019).

[Wikg] Wikipedia: NP (complexity). https://en.wikipedia.org/wiki/NP_(complexity) (Zugriff am 25.05.2019).

[Wikh] Wikipedia: Travelling Salesman Problem. https://en.wikipedia.org/wiki/Travelling_salesman_problem (Zugriff am 25.05.2019).

[You] Youtube: The Honey Bee Algorithm. https://www.youtube.com/watch?v=elTfueXcYaU (Zugriff am 10.06.2019).