Ausarbeitung zum Thema

Nature Patterns

In der Informatik angewandte Patterns aus der Natur

Software Design Patterns

Boris Wüst
Valentin Schwind

Inhalt

I. Einleitung 3
II. Allgemeines zur Bionik 4
III. Evolutionäre Algorithmen 8
IV. Neuronale Netze 13
V. Schwarmintelligenz 20

I. Einleitung

Schon seit Jahrhunderten nutzt der Mensch Ideen aus der Natur für die Lösung technischer Probleme. Der Patent-Fundus der Natur scheint unendlich und bietet ein riesiges Reservoir für technologische Zukunftsaspekte. Allgemein bekannte Beispiele sind Fische, die Paten der strömungsgünstigen Fahrzeuge oder das Blatt der Lotuspflanze, das der Nanotechnologie hilft unverwüstliche und schmutzabweisende Oberflächen zu entwickeln.

In den vergangenen Jahren hat sich daraus eine eigene Wissenschaft entwickelt: Die *Bionik* (zusammengesetzt aus **Bio**logie und Tech**nik**). Sie beschäftigt sich mit der Suche und Analyse natürlicher Vorbilder, die im Laufe der Evolution entstanden sind, sowie ihrer Umsetzung in technischen Entwicklungen.

Wissenschaftler, Ingenieure und Vertreter vieler unterschiedlicher Fachrichtungen arbeiten hierbei zusammen, um systematisch von der Natur zu lernen. In den Blickpunkt rücken dabei nicht nur Konstruktionen, Strukturen und Materialien, sondern auch immer mehr organisatorische Verfahren und Strategien, sowie natürliche Intelligenzen, die in der sogenannten *Informationsbionik* Anwendung finden.

In dieser Ausarbeitung zur Vorlesung *Software Design Patterns* wollen wir anhand von Beispielen zunächst die allgemeinen Prinzipien der Bionik erklären, um dann im Speziellen auf konkrete Anwendungen in der Informationsbionik einzugehen. Unser Fokus ist hierbei auf eine bekannte (und der wohl verbreitetsten Form) einer natürlichen Intelligenz gerichtet: Der *Schwarmintelligenz*.

Der Begriff „*Nature Patterns*“ wurde hierbei von uns kreiert um den Zusammenhang zwischen der (Informations-)Bionik und den Design Patterns aus der Softwaretechnik zu verdeutlichen und umfasst somit alle Software Design Patterns, die aus der Natur oder der Bionik abgeleitet sind.

II. Allgemeines zur Bionik

„*Von der Natur lernen*“ heißt der Leitgedanke der Bionik und ist häufig zu hören und zu lesen, wenn es um das Grenzgebiet zwischen Biologie und Technik geht. Zunächst sollte man sich aber vor Augen führen, dass die Natur keinen Katalog an Blaupausen für die Technik bereitstellt, sondern dass jede Idee auf ihre Übertragungsmöglichkeit hin neu erforscht werden muss. Das einfache Kopieren der Natur durch die Bionik ist leider nicht möglich, doch sie bietet Anregungen und Impulse für Entwicklungen in den unterschiedlichsten Bereichen. Daher sind die wichtigsten Aspekte der Bionik: systematische Erlangung des vollständigen Wissens und Verständnisses einer natürlichen Vorlage, die Abstrahierung und Umsetzung (oder Problemlösung) in einer ihrer Zieldisziplinen wie z.B.:

- Material- & Werksstoffforschung
- Konstruktionsforschung (z.B. in Flugzeugbau, Autobau, etc.)
- Robotik- & Prothetik (u.a. auch Kybernetik & Medizin)
- Klima- & Energetobionik
- Architektur & Design
- Kinematik & Dynamik
- Neuro-, Evolutions-, Prozess- & Organisationsbionik
- etc.

Die *Klima*- und *Energetobionik* befasst sich zum Beispiel primär mit passiver Lüftung, Kühlung und Heizung. Dazu gehört die Idealausrichtung zu Sonne und Wind, Dachformen, Erdnischen, ideale Unterkellerung und Luftführung vom kühlen Erdbereich in die sommerwarmen Räume, Luftumwälzung mit Gasaustausch unter Verwendungen poröser Materialien, Energiespeicherung in Wärme aufnehmenden System und vieles mehr. Mit der Übernahme solcher natürlichen Prinzipien, wie sie beispielsweise die Termiten verwirklichen, können bis zu 80% der elektrischen Energie zur sommerlichen Kühlung und 40-60% der Energie zur Winterheizung eingespart werden.

Nicht nur natürliche Konstruktionen können als Vorlage dienen, sondern auch Verfahren, mit denen die Natur Vorgänge steuert. Mit der Übernahme solcher Verfahren befasst sich die *Prozessbionik*. Eines der wesentlichsten Vorbilder hierbei ist die Photosynthese im Hinblick auf zukünftige Wasserstoff-Technologien zur Energiegewinnung. Zudem können Aspekte der ökologischen Umsatzforschung mit großem Gewinn im Hinblick auf die Steuerung komplexer industrieller und wirtschaftlicher Unternehmungen untersucht werden. Schließlich sind es die natürlichen Methoden, des vollständigen Recyclings und der Vermeidung von Deponiematerial, wert in allen Details auf eine Übertragbarkeit hin untersucht zu werden.

Nature Patterns

Komplexes Management ist heute noch immer nicht in der Lage, vorausschauend allen Anforderungen eines auch nur mittleren Industriebetriebes gerecht zu werden. Im Gegensatz dazu laufen Organisationsfragen in der belebten Welt, sei es ein Einzelorganismus (die Gesamtkomplexität einer Fliege ist größer, als die der gesamten deutschen Volkswirtschaft), als auch in Organismen-Systemen und schließlich in ökologischen Systemen äußerst störungsarm ab. Die funktionellen Querbeziehungen von Ökosystemen, beispielsweise des Waldrands, sind bereits um einiges komplexer als die eines größeren Industriebetriebes. Die *Organisationsbionik* versucht aus der Art und Weise, wie die Natur Informationen organisatorisch zusammenfasst und verwendet zu lernen und diese analog in der Technik und der Verwaltung einzusetzen. Dieser Bereich der Bionik ist sehr zukunftsträchtig, wird jedoch im Moment nur sehr zögerlich von der Forschung aufgegriffen. Ein entscheidender Grund dafür könnte die Komplexität solcher Organisationsstrukturen sein.

Man kann erkennen, dass die Bionik schon jetzt alle nur denkbaren Forschungs-, Industrie- und Wirtschaftsbereiche durchdringt und in der Zukunft noch mehr an Bedeutung gewinnen wird. Zudem kann sie als Bindeglied zwischen den Naturwissenschaften gesehen werden.

Allgemein lässt sich die Bionik folgendermaßen definieren:

> *"Bionik als wissenschaftliche Disziplin befasst sich mit der technischen Umsetzung und Anwendung von Konstruktions-, Verfahrens- und Entwicklungsprinzipien biologischer Systeme".*
>
> Neumann, D. (ed) (1993): Technologieanalyse Bionik. Analysen + Bewertungen zukünftiger Technologien.

Die drei Prinzipien nochmal zusammengefasst:

1. (Bau-)Konstruktionen der Natur:	*Konstruktionsbionik*
2. Vorgehensweisen & Verfahren der Natur	*Prozess-, Verfahrens- und Organisationsbionik*
3. Entwicklung & Evolutionsprinzipen der Natur:	*Informationsbionik*

Um diese Prinzipien während des Entwicklungsprozesses zu übertragen werden zwei verschiedene Verfahren angewandt. Es wird zwischen dem *Top-Down*- und dem *Bottom-Up-Prozess* unterschieden, die sich durch die Ausgangslage und Herangehensweise an eine technische Entwicklung voneinander unterscheiden. Beide Prozesse müssen bei Neuentwicklungen jedesmal neu durchlaufen werden.

Der Top-Down-Prozess

Der Top-Down-Prozess wird in der Bionik angewendet, um ein Problem zu lösen, das durch eine bestehende technische Entwicklung, oder nur durch eine Idee entstanden sein kann. Hierbei wird in der Natur nach einer Lösung gesucht und im Detail analysiert. Wenn eine ähnliche Vorlage und Lösung in der Natur existiert, wird diese auf das technische Problem angewendet. Hier liegt der Schwerpunkt der Arbeit bei den technischen Ingenieuren, die sich häufig erst durch ihr Problem an natürlichen Vorbildern orientieren.

Zwei Beispiele, um den Top-Down-Prozess näher zu beschreiben:

- Um z.B. Autoreifen zu verbessern, nahmen Ingenieure Katzen- und Hundepfoten als Vorlage, um Profile mit besserer Fahrbahnhaftung zu entwickeln. Die Forscher erkannten, dass sich die Haut der Pfoten bei einem schnellen Richtungswechsel verbreitert und damit mehr Kontaktfläche zum Boden hat.
- Der hohe Treibstoffverbrauch von Flugzeugen inspirierte Forscher dazu, die Flügel segelnder Vögel im Flug zu beobachten. Man erkannte, dass die Federn an den Flügelspitzen beim Gleitflug nach oben gerichtet waren, dadurch weniger große Luftwirbel erzeugten und die Tiere insgesamt mit weniger Energieverbrauch in der Luft bleiben konnten. Das Resultat wurde schließlich durch Winglets an den Flügelspitzen von Flugzeugen realisiert.

Der Top-Down-Prozess lässt sich wie folgt in vier einzelne Schritte zerlegen:

1. Erfassung und Definition eines technischen Problems (Niedrige Bodenhaftung/Hoher Treibstoffverbrauch)
2. Ähnliche Muster in der Natur suchen (Pfoten von Katzen/Flügelspitzen von Vögeln)
3. Analyse und Verständnis natürlicher Vorbilder (Verbreitung der Haut bei Richtungswechsel/Kleinere Luftwirbel)
4. Aus den gewonnenen Informationen Lösungen für das bestehende Problem suchen und umsetzen (neuartige Profilstrukturen/Winglets)

Der Bottom-Up-Prozess

Der Bottom-Up-Prozess wird häufig auch als Abstraktions-Bionik bezeichnet und hat im Gegensatz zum Top-Bottom-Prozess seine Wurzeln in der biologischen Grundlagenforschung. Hier forschen Bioniker an einer biologischen Struktur, analysieren und beschreiben ihr Prinzip. Der wichtigste Schritt hierbei ist die Abstraktion der Erkenntnisse, um einerseits von der biologischen Sicht auf eine nicht-fachspezifische Übersetzung zu kommen, und andererseits eine geeignete technische Anwendung zu suchen. Diese wird bei entsprechendem Nutzen gemeinsam mit Ingenieuren, Technikern, Statikern, Designern und in vielen anderen Disziplinen umgesetzt.

Zwei bekannte Beispiele aus der Bionik, die über den Bottom-Up-Prozess entstanden sind:

- Biologen entdeckten, dass auf der Oberfläche von Blättern der *Lotus-Pflanze* flüssige und wasserlösliche Substanzen abperlen. Sie untersuchten die Blätter und fanden heraus, dass mikroskopisch kleine Noppen auf den Blättern verhindern, dass sich Schmutz und Wasser auf Oberfläche festhalten können. Die *Adhäsionskräfte* (Anhaftungskräfte einer Oberfläche) sind so gering, dass selbst Flüssigkeiten mit einer geringen Oberflächenspannung ihre Kugelform nicht verlieren und es zu keiner *Benetzung* (Festhaften einer Flüssigkeit an einen festen Gegenstand) kommt. Dieses Prinzip führte dazu, dass Oberflächen entworfen wurden, die einen selbstreinigenden Effekt aufweisen. (z.B. Dachziegel, Fassadenfarbe, Autolack, Reinigungssprays).
- Unter seinem Mikroskop entdeckte ein schweizerischer Ingenieur an den Spitzen der *Großen Klette* kleine elastische Widerhaken, die die Früchte der Pflanze an seiner Kleidung und in dem Fell seiner Hunde festhaften ließen. Die Haken brachen beim Entfernen nicht ab und George de Mestral entwickelte daraus die Idee zwei Materialien einfach und umkehrbar miteinander zu verbinden. Daraus entstanden schließlich die *Klettverschlüsse*, die aus dem heutigen Alltag nicht mehr wegzudenken sind.

Der Bottom-Up-Prozess lässt sich in fünf Schritte einteilen:

1. Biologische (oft auch physikalische) Grundlagenforschung: Erfassung und Definition einer biologischen Auffälligkeit oder Eigenschaft, seiner Funktion und evtl. Umgebung. (Abperlendes Wasser auf Blättern/Selbsthaftende Früchte)
2. Analyse und Beschreibung des Prinzips (Noppen auf der Blattoberfläche/Widerhaken an den Spitzen)
3. Abstraktion und Verallgemeinerung (mikroskopische Unregelmäßigkeiten verhindern Benetzung/zwei Materialien für reversible, aber rel. stabile Verbindungen)
4. Suche technischer Anwendungen (Oberflächen für Fenster, Wände, etc./einfacher Verschluss für Kleidung & Schuhe)
5. Entwicklung einer technischen Anwendung, Suche nach einer Möglichkeit der Umsetzung (hydrophobe Beschichtungen mit noppenförmigen Nano-Strukturen/Haken-, Flausch-, Pilz- und Veloursbänder)

Neben diesen zwei Prozessen kommt es vor, dass Ingenieure ohne direkte Kenntnisse einer biologischen Analogie eine entsprechende Technik finden. In diesem Fall gingen sie und Biologie getrennte Wege. Einige Beispiele:

- Sonar & Echolot Orientierung von Fledermäusen/Delphinen
- Düsenantrieb Rückstoßprinzip bei Quallen/Tintenfischen
- Spritzen Stachel/Rüssel von Bienen/Stechfliegen
- Propeller Ahornblätter/Libellen

III. Evolutionäre Algorithmen

Das Leben auf der Erde hat zu einer unvorstellbar großen Artenvielfalt geführt. Die *Evolution* hat dabei Organismen und Lebensformen hervorgebracht, die sich mit der Zeit optimal an ihre Umwelt und Umgebung angepasst haben. Durch die Manipulation des Erbguts löst die Natur dabei ein schwieriges Optimierungsproblem: Die *Mutation* und *Selektion*. Dieses grundlegende Evolutionsprinzip nachzuahmen und dadurch neue *heuristische* (wiederholbare) *Optimierungsverfahren* zu entwickeln, ist das Ziel *evolutionärer Algorithmen*, bzw. der *evolutionären Programmierung*. Evolutionäre Algorithmen werden meist durch den Top-Down-Prozess nachgebildet und an spezielle Probleme angepasst.

Individuen (zusammengefasst in der *Population*) verändern sich durch unterschiedliche Faktoren (sogenannte *genetische Operatoren*): Mutation, Rekombination und Selektion. Erstaunlich sind hierbei die einfachen Funktionsweisen der Operatoren und das Zusammenspiel der verschiedenen Mechanismen.

1. Mutation

   In der Natur kann es bei der Zellteilung zu „kleinen Fehlern“ bei der Replikation des genetischen Codes vorkommen. Die *Mutation* ist ein stochastischer (ungerichteter) Operator, der die genetischen Informationen eines Individuums *zufällig* verändert. Sinn dabei ist die Erzeugung *alternativer Varianten* im genetischen Material, die bei der Programmierung verwendet werden, um ein bestimmtes Optimum zu finden.

2. Rekombination

   In der Evolution werden die Genome verschiedene Individuen (Eltern) durch die *Rekombination* (d.h. die *Vermehrung* oder *Kreuzung*) vermischt. Dabei können durch unterschiedliche Verfahren neue Generationen von Kindern entstehen. Wie in der Natur kann (durch asexuelle Fortpflanzung) ein evolutionärer Algorithmus ohne Rekombination auskommen (ein Elternteil pro Kind).

3. Selektion

   Die *Selektion* bestimmt welche Individuen in der nächsten Generation zur Fortpflanzung zugelassen werden. Selektionen ohne Störungen sind *deterministisch*, werden jedoch meist durch zufällige Ereignisse (*Umweltselektion*) gestört. Selbst Individuen mit den besten Voraussetzungen (Optimum) für eine Rekombination, können durch einen Unfall sterben, bevor sie ihren Partner wählen (*Paarungselektion*) und Nachkommen zeugen.

Die Möglichkeit Rekombination und Mutation in ihrer Reihenfolge zu vertauschen, hängt von der Frequenz ab, mit der neue Generationen entstehen. So ist es bei langen Lebenszyklen (z.B. beim Menschen) wahrscheinlich, dass eine Mutation einer Zellstruktur *vor* der Rekombination statt-

findet, während bei kurzen Intervallen (z.B. bei *Eintagsfliegen*) die Mutation erst während der Zellteilung möglich ist.

Programmiertechnisch muss bei evolutionären Algorithmen eine Bewertungsfunktion hinzugefügt werden, um festzustellen, ob ein Optimum ermittelt wurde und um den Erfolg einer Generation zu bewerten. Dadurch lassen sich später auch zusätzliche Selektions- und Terminierungsbedingungen errechnen, die auch für deutliche Schwankungen einer Population verantwortlich sein können. Die Selektion soll dabei zwischen sinnvollen und unnützen Mutationen unterscheiden. Ziel hierbei ist es, dass sich die sinnvollen Neuerungen durchsetzen und die unnützen fallengelassen werden, um den Organismus optimal an seine Umweltbedingungen anzupassen. Diese Wechselwirkung wird in der Programmiertechnik auch als *Fitness* (auch *Lösungsgüte*) bezeichnet. Vermehren sich besser angepasste Individuen stark genug, kann die Fitness indirekt über die Anzahl der Nachkommen ermittelt werden. Die Rekombination ist kein zwingend erforderlicher Evolutionsfaktor, da keine neuen Informationen in den „*Genpool*" eingeführt werden. Eine Optimierungsstrategie, die nur durch Rekombination (und ohne Mutation) durchgeführt wird, funktioniert nur, bis die beste Kombination existierender Informationen ermittelt wurde.

Schematische Darstellung ohne Anpassung an ein konkretes Optimierungsproblem:

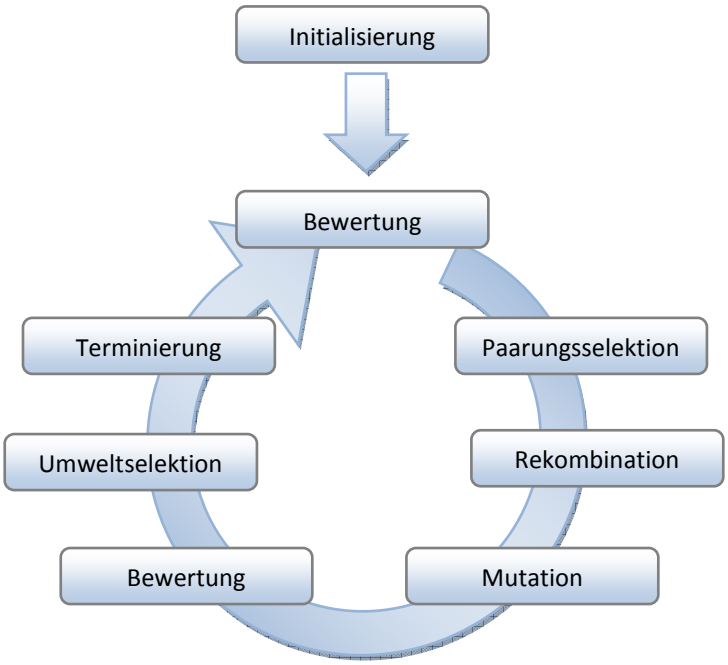

Abbildung 1: Optimierung mit evolutionären Algorithmen

Pseudocode eines einfachen evolutionären Algorithmus‘ mit der Population P:

```
Strategieparameter (z.B. Nachkommen pro Generation)
Initialisiere Ausgangspopulation P(0)
t = 0
Wiederhole (Generationszyklus) wenn nicht Abbruchbedingung:
    Bewertung P(t)
    Selektion von P_neu(t) aus P(t) (welche Individuen werden für die Fortpflanzung ausgewählt)
    Rekombination von P_neu(t) (Bildung der Nachkommen)
    Mutation von P_neu (t) (Erzeugung von Störungen)
    Bewertung der Nachkommen P_neu(t)
    Zufällige Selektion der Nachkommen der von P_neu(t) (Umwelteinflüsse)
    Bilden der Population mit P(t + 1) = P_neu(t) (Reinkarnation)
    t = t + 1
Bis t_Max erreicht oder andere Abbruchbedingung erfüllt ist
Stopp
```

Abbildung 2: Pseudocode eines evolutionären Algorithmus

Evolutionsstrategien

Evolutionsstrategien wurden unabhängig von den evolutionären Algorithmen entwickelt, gehören aber zu den evolutionären Algorithmen, da sie dasselbe Prinzip (Rekombination, Mutation, Selektion) verwenden. Wesentliches Merkmal der Evolutionsstrategien ist, dass die Parameter der Strategie und einzelner Individuen kombiniert und mutiert werden. Die Selbstanpassung (*Adaption*) wird durch einen großen Geburtenüberschuss und durch die Umweltselektion erzielt (vergleichbar mit Lebensformen, die einer gewaltigen natürlichen Selektion unterliegen und eine gewaltige Population zum Überleben benötigen, wie z.B. Meeresschildkröten). Wegen des enormen Rechenaufwands bei großen Populationen wird das natürliche Vorbild der biologischen Evolution nur soweit nachgebildet, wie es das gegebene Problem erforderlich macht.

Zwei bekannte Anwendungen von Evolutionsstrategien:

- Um den geringsten Wiederstand von aneinandergereihten Gelenkplatten in einem Windkanal zu ermitteln, veränderte der deutsche Ingenieur Ingo Rechenberg mit Hilfe von zufälligen Veränderungen die Lage der einzelnen Platten. Durch den Algorithmus konnte er den Luftwiderstand auf ein Minimum reduzieren.
- Die Zweiphasen-Überschalldüse ist ein Kleinantrieb für Raumfahrzeuge. Zur Berechnung einer optimalen Düsenform und der komplexen Strömungsvorgänge bediente man sich einer Evolutionsstrategie, die von einer konischen Ausgangsform mit 50% Wirkungsgrad zu einer völlig neuen Düsenform mit fast 80% Wirkungsgrad geführt hat.

Evolutionäres Programmieren

Zur Simulation endlicher Automaten (anfang nur Analyse und Vorhersage großer Datenströme) benötigte man eine abgewandelte Form evolutionärer Algorithmen, die als *evolutionäres Programmieren* bekannt wurde und Probleme auf innovative Art und Weise lösen sollten (in Intelligenz dem Menschen gleich oder überlegen). Charakteristisch hierfür ist das Fehlen der Rekombination. Die evolutionäre Programmierung konzentriert sich auf eine spezielle *Turnier-Selektion* zur Separation der neuen Generation. Wie bei Evolutionsstrategien gibt es hier eine *selbstadaptive* Steuerung der Strategie- und Mutationsparameter eines Individuums.

- Bekannte Anwendungen hierfür sind *künstliche Intelligenzen* in Spiele-Software mit hohen KI-Anforderungen (Dame, Schach), die Optimierung neuronaler Netze (z.B. die *Backpropagation* (Fehlerrückführung) und das Einlernen von künstlichen, neuronalen Netzen (siehe Kapitel: V. *Neuronale Netze*), sowie Topologieoptimierungen).

Genetische Algorithmen

Während die Evolutionsstrategien und die evolutionäre Programmierung mit reellen Zahlenräumen arbeiten, konzentrieren sich die *genetische Algorithmen* auf die *binäre Darstellung* (z.B. die Mutation einer 8-Bit Zahl: 10101001 zu 10101101 mit einer Mutation an 3. Stelle). Wenn eine Population aus einer bestimmten Anzahl Individuen besteht, werden zwei dieser Individuen anhand ihrer Fitness für die Paarung und einem zufälligen Wert ausgewählt. Der Nachkomme erhält die ersten Bits dieses zufälligen Werts von einem Eltern- und den Rest vom anderen Elternteil. Danach werden mit einer geringen Wahrscheinlichkeit Bits verändert (mutiert) und die Nachkommen ersetzen die Eltern, bis ein Terminierungskriterium erreicht ist. Es werden damit nicht die natürlichen Auswirkungen der Evolution imitiert, sondern die genetische Kodierung, also die *Ursache der Evolution*. Der Schwerpunkt liegt daher in der Rekombination als *Kreuzungsmechanismus*. Ein Individuum ist bei genetischen Algorithmen mit einem *Chromosom* gleichzusetzen, kann aber auch aus mehreren Chromosomen bestehen (*Strings*). Da Computersysteme ebenfalls mit Zahlenwerten in Binärform arbeiten, sind genetische Algorithmen optimal auf derzeitige Rechner zugeschnitten und sind in der Praxis auch die am häufigsten verwendeten evolutionären Algorithmen.

Eine abgewandelte Form von genetischen Algorithmen ist der *Memetische Algorithmus*, der auch als hybrider genetischer Algorithmus (*HyGLEAM, Hybrid General Evolutionary Algorithm And Method*) oder parallel genetischer Algorithmus bezeichnet wird. Der Memetische Algorithmus ergänzt genetische Algorithmen durch ein lokales (Such-)Verfahren, das den besten oder gar alle Nachkommen einer Generation künstlich verbessert und die Eltern gemäß der Akzeptanzregeln ersetzt.

Genetisches Programmieren

Im Unterschied zu genetischen Algorithmen oder der Evolutionsstrategie, wird ein Individuum als ein eigenes Programm (oft auch ein ganzes Computersystem) angesehen. Hierbei wird ein Pool von Programmen (oder Computern) für die Lösung von Problemen unbekannter Form eingesetzt. Die genetische Programmierung beschäftigt sich mit der Lernfähigkeit von Computern (und deren Software), ohne dass sie für einen bestimmten Zweck programmiert wurden. D.h. es wird untersucht, ob Programme Aufgaben lösen können, die sich nicht automatisch generieren lassen. Ziel ist es eine künstliche Intelligenz zu schaffen, der gesagt wird, *was* sie tun soll, ohne die Information *wie* dies zu erledigen ist.

Es gibt derzeit 36 bekannte Fälle von genetisch programmierten Lösungen, die mit einer menschlichen Erfindung vergleichbar sind. 15 dieser Fälle erschufen dabei Erfindungen, die aus dem 20. Jahrhundert, 6 aus dem 21. Jahrhundert stammen und 2 (PID- und Non-PID-Controller), die zu einer patentierbaren, völlig neuen Erfindung geführt haben.

Siehe auch unter: www.genetic-programming.com

IV. Neuronale Netze

Das menschliche Gehirn völlig zu verstehen und seine unvorstellbare Leistungsfähigkeit nachzubilden oder gar zu übertreffen, ist einer der ältesten Träume in der Informationstechnologie. Das komplexe Neuronennetz des Gehirns bildet den Schlüssel dazu und dient als Vorlage und Inspiration *künstlicher, neuronaler Netze*. Im Folgenden beschäftigen wir uns nicht mit der Modellierung eines künstlichen, neuronalen Netzes, das verwendet wird, um das menschliche Gehirn nachzuahmen und besser zu verstehen (Bottom-Up-Prozess), sondern hauptsächlich um Anwendungsprobleme aus der Informationstechnik zu lösen (Top-Down-Prozess) und Informationen zu verarbeiten.

Künstliche, neuronale Netze setzen sich, wie ihre natürlichen Vorbilder, aus *Neuronen* zusammen. Sie bilden die Basis für ein Modell, dessen Ziel es ist, zu lernen und sich mit komplexen Problemen zu beschäftigen, die sich auf üblichem (Rechen-)Weg nicht lösen lassen. Dieses Modell nennt man auch *Konnektionismus*, es setzt sich (unabhängig von seiner Anwendung) aus folgenden Schritten zusammen:

1. Sammlung von Daten
2. Entwicklung eines Modells aufgrund der erhobenen Daten
3. Bildung von Vorhersagen/Hypothesen aufgrund „der Erfahrung“
4. Kontrolle der Lösung

Die wohl größte Schwierigkeit ist die Entwicklung eines Modells (in Schritt 2), dass nur mit viel Rechenleistung erstellt werden kann. Es muss festgehalten werden, dass neuronale Netze in der der Software- und Hardwaretechnik immer noch über bestehende Plattformen realisiert werden müssen. Es stellt sich in der Praxis zusätzlich die Aufgabe (und auch das Problem), dass die Funktionsweise des Konnektionismus nicht algorithmisch dargestellt werden kann (d.h. auch durch übliche mathematische Formeln und Berechnungen nicht lösbar ist). Zudem ist nicht ersichtlich, wie das Netzwerk intern funktioniert. Ein Ergebnis entsteht erst im Zusammenspiel aller Elemente und ist erst nach einiger Zeit und am Ende der Verarbeitungskette sichtbar. Oft liegt auch kein konkretes Resultat einer Verarbeitung vor, sondern eher eine Richtlinie oder Tendenz, die quasi das erlangte *Wissen* des Netzes repräsentiert.

Neuronen

Wie schon erwähnt ist das wichtigste Element in einem neuronalen Netz das *Neuron*. Neuronen sind Zellen, die für die Verarbeitung von Signalen zuständig sind. Neuronen werden in der *Neuroinformatik* auch als *Units*, *Elemente* oder *Knoten* bezeichnet. Sie bestehen aus meist vier Elementen, deren Vorbilder sich ebenfalls in der Biologie wiederzufinden sind.

1. Gewichtungen
   Die Gewichtung zwischen zwei Neuronen ist vergleichbar mit einer *Synapse* in der Biologie. Sie bestimmt wie stark ein Signal Einfluss auf die Aktivierung eines Neurons nehmen kann. In der Regel markiert eine Gewichtung mit dem Wert 0 eine nicht vorhandene Verbindung, ein negativer Wert wirkt hemmend und ein positiver erregend.

2. Übertragungsfunktion
   Die eigentliche Aufgabe (auch: *Netzeingabe*) einer Zelle wird von ihrer *Übertragungsfunktion* bestimmt. Hier laufen unterschiedliche Eingangssignale zusammen und bestimmen anhand ihrer Funktionsweise, wie das Ausgangssignal des Neurons aussehen soll. Vergleichbar ist dies am besten mit der Funktionsweise einer elektrischen Schaltung oder mit boolesche Funktionen (logisches AND, OR, XOR, die auch sehr häufig in neuronalen Netzen zum Einsatz kommen).

3. Aktivierungsfunktionen
   Trifft das Signal der Netzeingabe auf die Aktivierungsfunktion, entscheidet diese anhand eines Schwellenwertes, ob und wann es zu einer Aktivierung kommt. Die verbreitetsten Aktivierungsfunktionen sind:
   - Lineare Funktionen (proportional, ohne Schwellwert)
   - Lineare Funktion mit Schwellwert (z.B. um Rauschen zu unterdrücken)
   - Binäre Funktion (Aktivitätslevel immer 0 oder 1)
   - Sigmoide Funktionen (das Aktivitätslevel steigt langsam an, die Funktion neigt sich dann steiler und flacht bei einem hohen Input wieder ab.)

4. Schwellenwert
   Analog zum Schwellenpotential in Nervenzellen, bestimmt der Schwellenwert, ob ein Neuron aktiviert wird.

Über Synapsen erhalten Neuronen Signale und geben sie an andere weiter. Dies können einfache Signale oder sehr komplexe Informationen sein. Entscheidend für die Errichtung eines neuronalen Netzes ist die Anordnung bzw. *Topologie* der Neuronen, die auch die nächste Phase, das *Training*, beeinflusst.

Man unterscheidet zwischen drei Arten von Neuronen:

- Input-Units (Neuronen, die sogenannte *Reize* von außen empfangen)
- Hidden-Units (Befinden sich zwischen Eingangs- und Ausgangsneuronen und bilden intern die Außenwelt ab)
- Output-Units (Neuronen, die Signale an die Außenwelt weiterleiten)

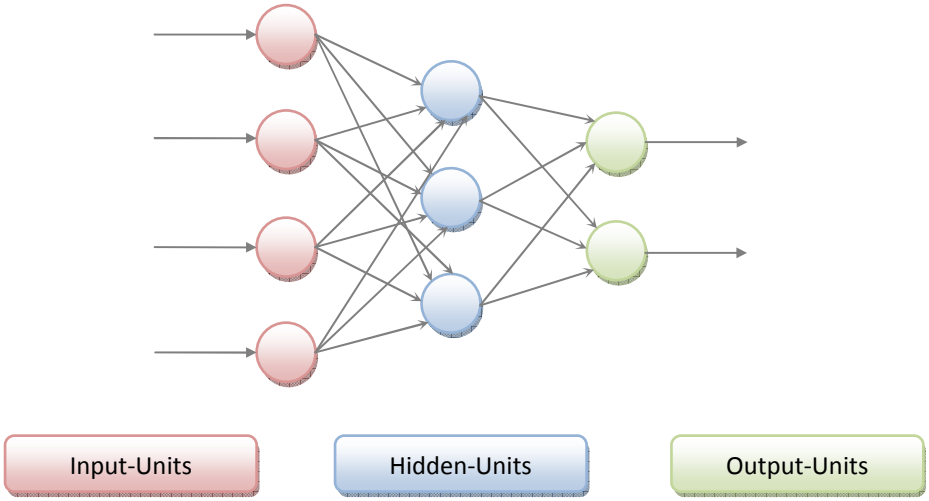

Abbildung 3: Darstellung eines einfachen neuronalen Netzes

Zunächst fassen wir noch einmal zusammen und werfen einen Blick auf das Vorbild in der Biologie. Die Neuronen, mit ihrer einfachen Funktionsweise (sozusagen: Aufnahme, Verarbeitung und Ausgabe von Signalen), bilden das neuronale Netz, das erst in einer bestimmten, gut durchdachten Anordnung funktionieren und komplexe Probleme lösen kann. Doch damit ein neuronales Netz funktionieren kann, muss ihm beigebracht werden wie.

Trainingsphase

Es gibt mehrere Methoden, durch die ein neuronales Netz „lernen“ kann. Man kann Neuronen und Verbindungen hinzufügen oder entfernen, man kann die Gewichtungen zwischen den Neuronen verändern oder die Schwellwerte anpassen, bei denen die Neuronen aktiviert werden. Die Veränderung der Gewichtung zwischen den Neuronen bildet dabei die wichtigste Regel und kommt in der Praxis am häufigsten zum Einsatz, da sie die Topologie eines Netzes nachträglich nicht verändert. Die *Lernregeln* geben dabei die Art und Weise an, wie das neuronale Netz Manipulationen durchführt. Sie geben an, wie man für bestimmte Eingangssignale zugehörige Ausgabesignale schafft. Dies geschieht auf zwei unterschiedlichen Wegen (*Überwachtes und nicht überwachtes Lernen*). Eine dritte Form (*Bestärkendes Lernen*) führt nicht immer zu einer korrekten Konfigurationsänderung und findet in der Praxis daher selten Anwendung:

1. Nicht überwachtes bzw. unbeaufsichtigtes Lernen

   Das neuronale Netz verändert sich nur durch die Eingabe, das Muster der Ausgabe wird nicht vorgegeben. Bekannte Lernverfahren:

   - Hebb‘sche Lernregel
   - Kohonennetze

2. Überwachtes bzw. beaufsichtigtes Lernen
   Durch die Vorgabe des Ausgabemusters werden die Gewichtungen zwischen den Neuronen optimiert.
   - Delta-/Perzeptron-Lernregel
   - Backpropagation

Die Natur zeigt uns, dass die Lernregeln erheblichen Einfluss auf ein neuronales Netz haben können. Es ergeben sich aus ihnen Eigenschaften, die für natürliche sowie auch für künstliche Systeme gelten. Zu den bemerkenswerten Eigenschaften eines neuronalen Netzes gehört, dass es komplexe Muster lernt, ohne die darüber liegenden Regeln (Abstraktion) zu kennen. Dies bedeutet, dass ohne konkrete (Lern-)Erfahrung, keine genaue Vorhersage über die Richtigkeit eines spezifischen Musters getroffen werden kann.

Ausbreitungsphase/“Kann-Phase“

Zur Kontrolle der Modifikationen aus der Trainingsphase wird in der Ausbreitungsphase überprüft, ob das Netz gelernt hat und was es „kann“. Es werden dabei Reize an die Eingangsneuronen gesendet und verglichen, wie das Ausgangssignal des neuronalen Netzes aussieht und was es berechnet hat. Um die Lernfähigkeit eines Netzes zu testen, unterscheidet man zwischen zwei Reizen.

1. Ausgangsreize
   Das neuronale Netz wird überprüft, ob es das zu lernende Material vollständig erfasst und richtig verarbeitet hat. Je nach Ausgangsreiz, wird das neuronale Netz daraufhin angepasst.

2. Neue Reize
   Ob das neuronale Netz in der Lage ist neue Reize zu verarbeiten, erkennt man, wenn es gelernt hat diese zu *generalisieren* und neue Aufgaben zu lösen. Findet es keine Verwendung für diese neuen Reize, hat das neuronale Netz keinen Wert für zukünftige Problemlösungen.

Klassen & Netztypen

Die Klassen der neuronalen Netze unterscheiden sich durch die verschiedenen Topologien, Lernregeln und Verbindungsarten. Es ist jedoch nicht immer eine klare Zuordnung zwischen Lernregel und Netztyp möglich, da manche Netztypen auf einer bestimmten Lernregel basieren, während sich andere mit verschiedenen Lernregeln modellieren lassen. Bei der Klassifikation ist

ebenfalls wichtig, ob es sich um überwachtes oder nicht überwachtes Lernen handelt, es Rückkopplungen (Feedback-Netze) zwischen den einzelnen Neuronen gibt und ob Hidden-Units vorhanden sind.

Wir haben einige bekannte Netztypen neuronaler Netze aufgelistet und geben einen kurzen Überblick über ihre Eigenschaften und Anwendung:

1. McCulloch-Pitts-Netze
   Zu den ältesten und einfachsten neuronalen Netzen gehört das Netz von *McCulloch-Pitts* (1943). Es verwendet ausschließlich binäre Signale (mit dem Fokus auf boolesche Algebra), ist aber viel mächtiger, als normale elektrische Schaltungen, da sie mit Signalstärken und Hemmschwellen funktionieren. Damit lassen sich Neuronen deaktivieren, was zu Schaltungen mit einem sehr viel geringeren Verbrauch an Bauteilen, Leitungen und Strom führt.

2. Perzeptron
   Ein ebenfalls frühes und einfaches, neuronales Netz bildet das *Perzeptron*. Seine Neuronen besitzen angepasste Gewichtungen und Schwellenwerte. Es wurde entwickelt, um einen Eingabevektor in einen Ausgabevektor zu verwandeln. Dabei unterscheidet man zwischen einlagigen (Input- und Output-Units) und mehrlagigen (Input-, Output- und Hidden-Units) Perzeptrons.

3. Lernmatrix
   Eine Matrix aus Eigenschaftsleitungen (n) und Bedeutungsleitungen (m) die untereinander verbunden sind, lassen sich software- als auch hardwaretechnisch realisieren. Durch eine Lernmatrix können sich komplexe Verbindungen zwischen Eigenschaftsmengen und den zugehörigen Bedeutungen herstellen lassen. Dadurch lässt sich eine Schaltung aufbauen, die nach ihrem Training in der Lage ist, Merkmale an die (ihr bekannte und) wahrscheinlichste Bedeutung weiterzuleiten.

4. Pattern Associator
   Neuronales Netz zur *Mustererkennung*, das ähnlich wie die Lernmatrix funktioniert, allerdings zusätzlich noch über eine Reihe praktischer Eigenschaften verfügt. Es gibt eine Toleranz gegenüber internen (fehlerhaften Neuronen) und externen Schäden (fehlerhafter Input), bei der Eingabe können Reize in Gruppen und zentrale Ausgaben in Tendenzen zusammengefasst werden.

5. Rekurrente Netze
   Neuronen unterschiedlicher Schichten werden durch sogenannte *Rückkopplungen* (*Backpropagations*) miteinander verbunden. Dadurch ist es beispielsweise möglich, zeit-

lich codierte Informationen in einem neuronalen Netz zu entdecken. Es wird zwischen direkten, indirekten und seitlichen Rückkopplungen unterschieden. Zudem gibt es vollständige Verbindungen, in der sämtliche Neuronen eines neuronalen Netzes miteinander verbunden sind. Weitere bekannte Rückkopplungsnetze sind die Hopfield-, Elman- und Jordan-Netze.

6. Kohonennetze/Self-Organizing-Maps (SOM)
   Als eines der wichtigsten neuronalen Netze, mit einem nicht überwachten Lernverfahren, gelten die *Kohonennetze*, deren Topologien auf dem menschlichen Gehirn basieren und unterschiedliche Eingangsreize verarbeiten können. Dadurch lassen sich komplexe, multidimensionale Eindrücke verarbeiten, die von einem Kohonennetz so verarbeitet werden, dass sie (räumlich abgebildet) nahe beieinander liegen. Der Raum des Netzes wird also *kartiert*. Laufen ähnliche Signale in die Karte (*map*), werden sie den Mustern zugewiesen, die ihnen auf der Karte am ähnlichsten sind (*clustering*). Die Kohonennetze gibt es in zahllosen Varianten und mit den unterschiedlichsten Erweiterungen.

Die Wahl der Netztypen hängt natürlich von der jeweiligen Anwendung ab. Wichtige Kriterien dabei sind die Hidden-Units, die verwendeten Lernregeln (überwacht, nicht überwacht), evtl. Rückkopplungen und natürlich das Kernkonzept einer Topologie. Oft können die biologische Plausibilität und die Fähigkeit nicht überwacht zu lernen, entscheidende Vorteile bei der Modellierung eines neuronalen Netzes sein.

Anwendungen

Als in den 80er Jahren die Rückkopplung künstlicher, neuronaler Netze wieder neu entdeckt wurde, tauchten auch entsprechende Anwendungen in der Informatik auf, die aus der heutigen Soft- und Hardwaretechnik nicht mehr wegzudenken sind. Neuronale Netze finden sich in unzähligen Bereichen wieder und bilden den zentralen Bestandteil folgender Software- und Anwendungsgebiete:

- Data-Mining-Software (hauptsächlich: Self-Organizing-Maps)
- Text-, Handschrift-, Bild-, Sprach- und Mustererkennung (Pattern Associator)
- Bildverarbeitung, (adaptive) Filter
- Routingstrategien (Internet), Routenplanung (Navigation), Hinderniserkennung
- Kompressionstechnologien (Codierung, Datenkompression)
- Marketing-Analysen (Konsumverhalten, Zielgruppenbestimmung)
- Organisation (Dienst-/Fahrplanoptimierung, Raumbelegungen, …)
- Allgemeine Anwendungen: Kontrolle, Optimierung, Sortierung, Auswahlstrategien, Datei-/Datenmanagement, Strukturierung, Management, Wissensgewinnung, Automatisations-Verfahren, Vorhersagen, Wertpapierbewertungen, Kursprognosen, medizinische Diagnosen

Eigenschaften

Von zentraler Bedeutung bei den neuronalen Netzen ist die *parallele Verarbeitung* von Reizen. Obwohl sie in der heutigen Softwaretechnik nur theoretisch machbar ist, da Computer Befehle pro Prozessor nur seriell verarbeiten, können Programme, die auf neuronalen Netzen basieren, enorme Leistungen vollbringen. Software, in der neuronale Netze zum Einsatz kommen, profitiert davon, dass sich Multikernsysteme zunehmend verbreiten. Zusätzlich zu der parallelen Verarbeitung, ist die *verteilte Speicherung* des Wissens eines neuronalen Netzes eine entscheidende Eigenschaft. Während auf üblichen Datenträgern Informationen seriell und an zentraler Stelle gespeichert werden (z.B. CDs oder Festplatten), lagert ein neuronales Netz eine Information (Wissen) quasi „überall“ oder zumindest in einem bestimmten Teilbereich seiner Topologie aus.

Ein weiterer Vorteil neuronaler Netze ist die (beschränkte) Resistenz gegenüber Schäden und Fehlern, sowie die Kategorisierung ähnlicher Reize. Da neuronale Netze auf dem menschlichen Gehirn basieren, bietet dieses stets eine ideale, biologische Vorlage, auf die man bei der Modellierung eines künstlichen, neuronalen Netzes stets zurückgreifen kann, um dessen Freiheitsgrade und Leistungsfähigkeit zu erhöhen.

Mehr Informationen zu neuronalen Netzen unter www.neuronalesnetz.de.

V. Schwarmintelligenz

Die *Schwarmintelligenz* ist ein emergentes Phänomen, das auf der Kommunikation innerhalb einer sozialen Gemeinschaft beruht und somit intelligente Verhaltensweisen des „Super-Organismus", d.h. aller Individuen der Gemeinschaft, ermöglicht.

Das klassische Beispiel aus der Tierwelt ist der *Ameisenstaat*. Intelligenz ist bei keinem der beteiligten Individuen festzustellen. Einzelne Ameisen haben nur ein sehr begrenztes Repertoire an Verhaltens- und Reaktionsmustern. Erst in einem sich selbst organisierenden *Kollektiv* zeigen sich plötzlich neue, intelligent erscheinende Verhaltensmuster. Nur durch diese kollektive Intelligenz sind komplexe Aufgaben wie Nestbau oder Nahrungsbeschaffung möglich. Zwar haben die Biologen lange Zeit gedacht, dass beispielsweise Ameisen- oder Bienenstaaten zentral durch ihre Königin gesteuert werden, allerdings hat sich diese Vermutung als Irrtum herausgestellt.

In gewisser Weise ist auch ein Gehirn ein solcher *Super-Organismus*, der für sich genommen aus „dummen" Individuen, den schon (in den neuronalen Netzen erwähnten) *Neuronen*, besteht. Ein einzelnes Neuron ist nichts weiter als ein Integrator mit Reaktionsschwelle. Erst das komplexe Zusammenspiel von Milliarden von Neuronen ergibt, was wir unter Intelligenz verstehen.

Wir wollen hier jedoch den Begriff „Schwarmintelligenz" auf die Domäne der Informatik und deren Nutzung innerhalb dieser Domäne beschränken, um etwas genauer ins Detail gehen zu können. In der Informatik versteht man unter Schwarmintelligenz das Forschungsumfeld der agentenbasierten „*Künstlichen Intelligenz*" (*KI*), auch „*Verteilte Künstliche Intelligenz*" (*VKI*) genannt. Dieses Arbeitsgebiet versucht komplexe, vernetzte Softwareagenten-Systeme nach dem Vorbild von staatenbildenden Insekten wie Ameisen, Bienen und Termiten als auch teilweise von Vogelschwärmen zu modellieren. Der eigentliche Begriff „*swarm intelligence*" wurde erst Ende der 80-er Jahre im Kontext der Robotik-Forschung eingeführt.

Die Individuen staatenbildender Insekten agieren in eingeschränkter Unabhängigkeit, sind jedoch bei der Erfüllung ihrer Aufgaben sehr zielgerichtet. Die Gesamtheit solcher Insektengesellschaften ist überaus leistungsfähig, was Forscher auf eine hochgradig entwickelte Form der Selbstorganisation zurückführen. Zur Kommunikation untereinander nutzen Ameisen beispielsweise Pheromone und Bienen den Schwänzeltanz. Trotz des Fehlens einer zentralisierten Steuerung bzw. Aufsicht, ist ein Kollektiv eindeutig mehr als nur die Summe aller beteiligten Individuen. In den letzten Jahren hat man verstärkt festgestellt, dass diese Eigenschaft gerade bei komplexen und verteilten Computersystemen ein überaus wichtiges Designziel darstellt, da diese tierischen Organisationsformen ziemlich flexibel, robust und anpassungsfähig sind. Dies wird unter anderem durch die dezentrale Steuerung erreicht, wodurch ein System keinen „Single-Point-Of-Failure" besitzt, da es fehlertolerant gegenüber dem Versagen einzelner Schwarmmitglieder ist. Die Entwicklung solcher Systeme wird zudem dadurch begünstigt, dass die einzelnen Individuen eines Schwarms nur sehr eingeschränkte kognitive Fähigkeiten besitzen. Das

simple Verhalten primitiver Individuen kann daher recht einfach durch Programme und Algorithmen nachgebildet werden.

Die Forschung im Bereich der *Verteilten Künstlichen Intelligenz* geht davon aus, dass durch die Kooperation künstlicher Agenten höhere kognitive Leistungen simuliert werden können. Dieses Phänomen wird auch „*Society of Mind*“ bezeichnet. Zur Implementierung der Kommunikation zwischen den Agenten wird häufig die „*Knowledge Query and Manipulation Language*“ (*KQNL*) eingesetzt.

Neben dem Forschungsumfeld der Verteilten Künstlichen Intelligenz ist Schwarmintelligenz auch ein nicht klar abgegrenztes Mode-Wort. In diesem Zusammenhang wurde beispielsweise auch die Ablösung der herkömmlichen, hardwarebasierten Netzwerke durch Schwarmintelligenz Systeme diskutiert, was sich aber schnell als utopisch und nicht wissenschaftlich fundiert herausgestellt hat. Diesem Leitbild der Schwarmintelligenz wird das Potential unterstellt, die Gesellschaft und ganze Märkte zu reformieren. Als Grundlage für diese Aussage dienen die sogenannten „*Smart Mobs*“, wie zum Beispiel die „*Critical Mass*“-Bewegung, bei der sich scheinbar zufällig und unorganisiert, Fußgänger und Fahrradfahrer treffen, um auf ihre Rechte gegenüber motorisierten Verkehrsteilnehmern aufmerksam zu machen.

Besonders erfolgreich war man bisher bei der Übertragung der Verhaltensweisen von Ameisen, weshalb wir hier hauptsächlich auf Anwendungen der Schwarmintelligenz nach ihrem Vorbild eingehen.

Die Ameisen

Über Millionen von Jahren wurden Ameisen von der Evolution darauf getrimmt, sich auch noch im größten Chaos zurechtzufinden und zur Sicherung der Existenz immer eine Lösung zu finden. Komplexe Handlungen werden dabei nicht durch die Leistungen einzelner Koloniemitglieder vollbracht, sondern können nur durch Gemeinschaftsaktionen entstehen, an denen viele Koloniemitglieder beteiligt sind. Man spricht bei solchen Super-Organismen auch von einem „*emergenten Verhalten*“: Das Wirken einzelner Individuen auf der *Mikroebene* zeigt seine Resultate erst auf der *Makroebene*. Dies fasst den Grundgedanken zusammen, der Schwarmintelligenz ausmacht

Die Futtersuche gehört für jede Lebensform zu einer existenziellen Aufgabe. In der Natur hat man beobachtet, dass Ameisen einen verblüffend guten und kurzen Weg zu ihrer Nahrungsquelle und zurück finden und das obwohl sie fast blind sind und kaum Informationen über ihre weiter entfernte Umwelt haben. Die Erklärung für dieses Phänomen ist, dass jede Ameise eine Pheromonspur legt, der weitere Ameisen zu einer gewissen Wahrscheinlichkeit flogen werden. Die Wahrscheinlichkeit hierfür wird durch die Qualität des Weges bestimmt, die hauptsächlich von der Schnelligkeit und der Gefährlichkeit des Weges abhängt. Optimal ist die Futterversorgung, wenn möglichst viel Futter in kurzer Zeit zum Nest gelangt.

Vereinfacht kann die Futterbeschaffung von Ameisen wie folgt beschrieben werden: Beim Ausschwärmen aus dem Nest markieren die Ameisen ihren Pfad mit Pheromonen, die die Artgenossen als Wegweiser wahrnehmen. Hat eine Ameise Beute gefunden, dann kehrt sie damit ins Nest zurück. Sie wählt den Pfad als Rückweg, der mit der größten Intensität an Pheromonen gekennzeichnet ist. War sie als Erste am Futter, dann kehrt sie auf ihrem eigenen Pfad zurück. Ist sie am Nest angekommen, so ist die Konzentration der Pheromone auf ihrem Pfad schon doppelt so hoch. Da Ameisen im Durchschnitt dieselbe Geschwindigkeit haben, hat automatisch der kürzeste bzw. schnellste Pfad die höchste Konzentration an Pheromonen, da die anderen Ameisen noch nicht wieder am Nest angekommen sind. Der Pfad mit der höchsten Konzentration an Pheromonen dient allen Ameisen als Wegweiser zum am nächstgelegenen Futter. Untersuchungen haben gezeigt, dass sich durch diesen natürlichen „Algorithmus“ nach einer gewissen Zeit automatisch die kürzeste Verbindung zwischen Futter und Nest ergibt. Ist die Futterquelle einmal aufgebraucht, dann kommt den Ameisen die Zeit zu gute, denn die Pheromone evaporieren (d.h. lösen sich auf) nach einer gewissen Zeit, sodass neue Pfade gebildet werden können.

Dieses Verhalten resultiert schließlich in den bekannten Ameisenstraßen und dient als Vorlage zur Lösung des in der Informatik existierend „*Travelling Salesman*“-Problems. Einzelne Ameisen werden dabei durch Agenten repräsentiert, während die gesamte Kolonie den Schwarm darstellt.

Die Futtersuche im Detail

In der Abbildung 4 (unten) ist eine Ameisenstraße schematisch dargestellt. Bei der ersten Darstellung a) wandern die Ameisen noch ungestört von A nach E. Dann wird ihnen plötzlich ein Hindernis (Obstacle), zum Beispiel ein Stein, in den Weg gelegt wie in b) und c) zu sehen ist.

Die Ameisen, die sich unmittelbar vor dem Hindernis befinden, können also nicht mehr weiter gehen und sehen auch keine Pheromonspur, an der sie sich orientieren könnten. Also wählt die eine Hälfte der Ameisen den Weg A – B – H – D – E (links um das Hindernis) und die andere den Weg A – B – C – D – E (rechts um das Hindernis).

Dies geschieht solange, bis die ersten Ameisen das Hindernis hinter sich gelassen haben. In diesem konkreten Fall also wenn die ersten Ameisen, die nach E wandern, den Punkt D und die Ameisen, die nach A wandern, den Punkt B erreicht haben. Sobald dies passiert, nehmen alle folgenden Ameisen eine höhere Konzentration an Pheromonen auf dem kürzesten bzw. schnellsten Pfad wahr und bevorzugen diesen fortan.

Die Konsequenz daraus ist, dass immer mehr Ameisen den, bis dahin kürzesten bzw. schnellsten Weg, A – B – C – D – E wählen und die Konzentration an Pheromonen auf diesem Pfad stetig zunimmt, bis schließlich fast alle Ameisen den schnellen Weg gehen. Dies wird solange fortgeführt, bis das ganze Futter von einer Futterquelle abtransportiert wurde und somit eine neue Futterquelle gefunden werden muss.

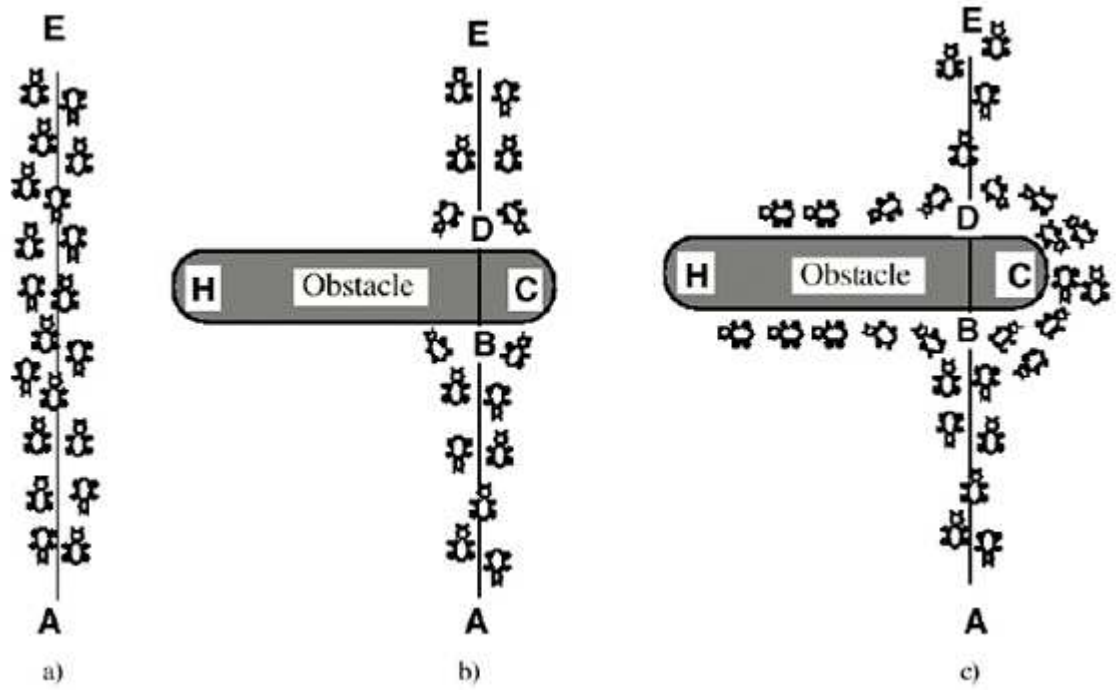

Abbildung 4: Schematische Darstellung einer Ameisenstraße

Von den Ameisen zu den Agenten

Wie schon erwähnt, ist das Funktionsprinzip der Futtersuche von Ameisen ein sehr vielversprechender Ansatz zur Lösung des „Travelling Salesman"-Problems. Um dieses Problem jedoch effizient lösen zu können, benötigen Software-Agenten noch ein paar Optimierungen, die ihre Kollegen in der Natur nicht haben. Diese haben natürlich zur Folge, dass zugleich die Fähigkeiten des ganzen Schwarms bzw. der Kolonie erweitert werden. Dies wurde berücksichtigt indem man mehrere Varianten des „Ameisen Kolonie"-Algorithmus implementiert hat. Die simpelste Variante des Algorithmus richtet sich dabei weitest gehend nach dem Vorbild in der Natur und wird einfach nur „*Ant Colony*"- (*AC*) oder „*Ant System*" (*AS*)-Algorithmus genannt. Mit jeder Variante kamen intelligente Optimierungen der Entwickler hinzu, wodurch der Algorithmus effizienter eingesetzt werden kann, sich jedoch auch immer mehr vom natürlichen Vorbild entfernt. Hier wollen wir nur auf die bedeutendsten Varianten des „*Ameisen-Algorithmus*" eingehen.

Das „Travelling Salesman"-Problem

Das „Travelling Salesman"-Problem, auch TSP genannt, besteht aus folgender Aufgabenstellung: Ein Vertreter soll eine möglichst kurze Rundreise durch mehrere Städte machen und dabei keine

Stadt zweimal besuchen. Am Ende der Reise soll er wieder in die Stadt zurückkehren, von der er gestartet ist.

Kombinatorisch gesehen gibt es (N-1)! Möglichkeiten für verschiedene Wege, wobei N die Anzahl der Städte ist (*asymmetrisches TSP*). Für den Fall, dass die Entfernung von „A nach B“ die gleiche ist wie von „B nach A“ und dies für alle Städte gilt, dann bleiben noch (N-1)!/2 Möglichkeiten übrig (*symmetrisches TSP*). Mit wachsender Städteanzahl wird es offensichtlich schwieriger, den optimalen Weg nur durch bloßes Durchprobieren aller Wege zu ermitteln, da es ganz einfach viel zu viele mögliche Wege gibt. Ein sehr naiver Lösungsansatz dieses Problems besteht darin, einfach immer in die Stadt zu gehen, die der aktuellen Stadt am nächsten liegt. Ein anderer Lösungsansatz besteht darin, das Verhalten der Ameisen bei der Futtersuche nachzubilden, indem die Wege mit künstlichen Pheromonspuren markiert werden.

Doch um das TSP effizient Lösen zu können, müssen wir unsere Ameisen-Agenten optimieren. Zum einen sind die Agenten im Vergleich zu den Ameisen nicht komplett blind, da sie die Entfernungen der einzelnen Städte kennen und sich anhand von diesem Kriterium orientieren. Zum anderen geben wir ihnen ein „Gedächtnis“, denn sie müssen sich merken welche Städte sie schon besucht haben. Ein vernachlässigbarer Unterschied ist, dass unsere Agenten in einer diskreten Zeit leben, da ein Computer nur diese kennt.

Die Ameisen-Agenten halten sich an folgendes Verhaltensmuster beim Ausführen des Programms: Sie wählen die Stadt, zu der sie als nächstes gehen, anhand der Intensität der Pheromonspur aber auch der Entfernung. Da sie sich gemerkt haben, in welchen Städten sie schon waren, schließen sie diese bei der Wahl der nächsten Stadt von vorneherein aus. Sobald die Rundreise beendet ist, also alle Städte genau einmal besucht worden sind, verteilen unsere Ameisen-Agenten nachträglich ihre Pheromonspuren auf dem Weg, den sie gegangen sind. Wir werden später noch einige Optimierungsmöglichkeiten für diesen Algorithmus aufzeigen.

„Help Car 54“ – Anwendung des TSP

Das Preisausschreiben mit dem Namen „Help Car 54“ wurde 1963 in den USA veranstaltet. Ziel war es, den kürzesten Weg durch 33 amerikanische Städte zu finden. Dies entspricht einem symmetrischen TSP mit 33 Städten und ergibt in etwa $1.3 * 10^{35}$ mögliche Wege, um alle Städte genau einmal zu durchqueren. Das stellte ein perfektes Problem für künstliche Ameisen Agenten dar. Die rot eingezeichnete Linie in der Abbildung ist die kürzeste Strecke, die unsere Ameisen Agenten gefunden haben.

Abbildung 5: Help Car 54 - Travelling Salesman Problem

Ant System (AS)

Das Ant System (AS) beruht auf der oben beschriebenen Futtersuche von Ameisen. Dabei liefert das AS eine Heuristik für die Lösung des TSP. Das Verhalten der Ameisen wird dabei durch den folgenden Algorithmus beschrieben.

```
Initialization
for every edge (i, j) do
    ti j = t0
end for
for k = 1 to m do
    place ant k on a randomly chosen city
end for
let T+ be the shortest tour found from beginning and L+ its length
for t = 1 to tmax do
    for k = 1 to m do
        build tour Tk(t) by applying n−1 times the following steps:
        choose the next city j with probability pi j where i is the current city
    end for
    for k = 1 to m do
        compute length Lk(t) of tour Tk(t) produced by ant k
    end for
    if an improved tour is found then
        update T+ and L+
    end if
    for every edge (i, j) do
        update pheromone trails
    end for
end for
print the shortest tour T+ and its length L+
```

Abbildung 6: Ant System Pseudocode

Ant Colony System (ACS)

Das Ant Colony Systen stellt eine Erweiterung des zuvor beschriebenen AS dar. Ziel dieser Erweiterung ist es, das Verhalten des Algorithmus zu optimieren.

Zum einen ist die Übergangsregel für die Wahl der nächsten Stadt eine andere. Dabei wird der Algorithmus dahingehend optimiert, dass er eher eine schon gefundene Lösung weiter optimiert anstatt neue Lösungen zu suchen.

Eine zweite Änderung macht ACS an der virtuellen Pheromonspur, die von den Ameisen-Agenten gelegt wird. Im ACS darf nur die Ameise, die den besten Pfad gefunden hat, am Schluss einer Iteration Pheromone ablegen. Das veranlasst die Ameisen in der nächsten Iteration des Algorithmus dazu, in der „Nähe“ dieser Lösung zu suchen. Hinzu kommt jedoch, dass jede Ameise schon auf ihrer Reise die Konzentration der Pheromone auf ihrem Pfad verändert. Entgegen dem natürlichen Vorbild vermindern sie jedoch die Pheromon-Konzentration, anstatt sie zu erhöhen. Dies hat den Effekt, dass diese Pfade für andere Ameisen uninteressanter werden, wodurch sich die Pfade mehr verteilen und die Ameisen quasi nicht hintereinander herlaufen. So

wird die Wahrscheinlichkeit erhöht, dass eine Ameise einen bessern Pfad findet als den bisher bekannten.

Zusätzlich zum Gedächtnis der Ameisen führt das ACS noch eine Kandidatenliste ein. Ausgehend von einem Startpunkt enthält diese Liste eine Anzahl von Städten, die dem Ausgangspunkt am nächsten gelegen sind. Diese „Kandidaten“ werden dann von der Ameise zuerst bereist.

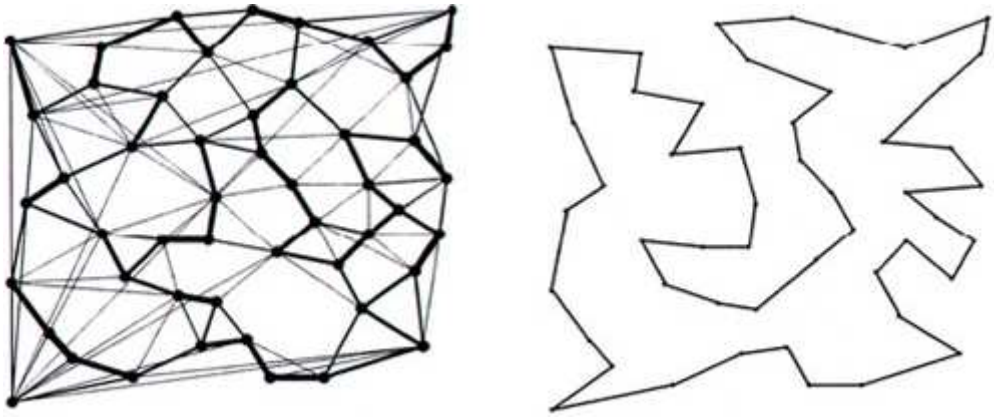

Abbildung 7: Konstruktion der kürzesten Strecke mit einem ACS

In der Abbildung sieht man ein aktives ACS. Die Dicke der Linien zeigt die Konzentration des Pheromons (links). Neben sehr stark markierten Pfaden gibt es solche, die etwas weniger Konzentration an Pheromonen aufweisen und solche, die kaum mit Pheromonen markiert wurden. Die Abbildung zeigt außerdem die im weiteren Verlauf gefundene beste Lösung (rechts). Die Pfade, die eine geringe Konzentration an Pheromonen aufweisen, können als alternative Pfade angesehen werden. Diese werden vor allem dann relevant, wenn man von einem statischen TSP zu einem dynamischen Problem übergeht: Fügt man Knoten in den Graphen ein oder nimmt man welche heraus, so ist das ACS in der Lage, schnell und effizient eine neue optimale Lösung zu finden.

Ant Colony Optimization (ACO)

Die vorher beschriebenen Algorithmen AS und ACS stellen *Metaheuristiken* für das TSP dar. Jedoch kann das Verhalten der Algorithmen weiter optimiert und für andere Einsatzgebiete angepasst werden. Nachfolgend möchten wir nur eine mögliche Optimierungen nennen, sodass man sich ein Bild von der Flexibilität dieser Metaheuristik machen kann.

Beispielsweise kann innerhalb der virtuellen Ameisenkolonie ein *Ranking* eingeführt werden, das von der Länge der gefunden Pfade abhängt. Die Menge des abgesonderten Pheromons ist dann proportional zum Rang der Ameise zu bestimmen.

Die vorgestellten Verfahren werden unter dem Begriff „*Ant Colony Optimization*“ (*ACO*) zusammengefasst. ACO kann immer dann eingesetzt werden, wenn der kürzeste Pfad durch einen Graphen gesucht wird. Auf diese abstrakte Schicht lassen sich neben dem TSP viele andere Probleme reduzieren. Beispielsweise kann das „*Job Scheduling*“-Problem mit Ant Colony Optimization gelöst werden. Hierbei geht es darum, eine bestimmte Menge von Jobs auf eine vorgegebene Menge von Maschinen zu verteilen. Auf weitere Anwendungen gehen wir später genauer ein.

Allgemein wird ACO für Optimierungsprobleme eingesetzt, was der Name schon andeutet. Ein Vorteil besteht vor allem darin, dass das Problem dynamisch sein kann, sich der Graph und die Distanzen der Knoten also ändern dürfen. Wie verschiedene Experimente gezeigt haben, liefert ACO bessere Ergebnisse als andere Verfahren. Die Tabelle vergleicht ACO mit anderen Heuristiken wie „*Simmulated Annealing*“ (*SA*), „*Tabu Search*“ (*TS*) und den *neuronalen Netzen* (*NN*).

Problem	GA	SA	TS	NN
Travelling Salesman	besser	besser	gleich	-
Vehicle Routing	-	besser	schlechter	besser
Single Machine Tardiness	-	besser	-	-

Abbildung 8: Performance-Vergleich verschiedener Heuristiken

Analog zum ACO versucht man auch das Verhalten von Bienen beim Nektar sammeln zu modellieren und damit Optimierungsprobleme zu lösen. Jedoch war man bei keiner Umsetzung der Verhaltensweisen von Insekten so erfolgreich wie bei den Ameisen. Das erklärt auch die immer größer werdende Rolle des Ameisen-Algorithmus in der angewandten Informatik.

Anwendungen

Hier sollen nun auf Anwendungen des Ameisen-Algorithmus bzw. der „Ant Colony Optimization“ im Detail eingegangen werden. Dieses Kapitel zeigt die Bedeutung dieser Metaheuristik sowohl in der Informatik als auch in anderen Arbeitsgebieten auf.

1. AntNet

Das *AntNet* definiert ein *Routing-Verfahren* für paketbasierte Netzwerke, wie beispielsweise IP-Infrastrukturen. Ähnlich wie im ACS „krabbeln“ dabei virtuelle Ameisen durch

das Netz. Im Vergleich zu anderen Routing-Verfahren ist die Idee beim AntNet, dass das Routing der Pakete auf Grund des aktuell bestehenden Netzwerkverkehrs geschehen soll. Bei anderen Verfahren basiert das Routing in der Regel auf angegebenen Kosten, wobei diese Kosten verschiedene Faktoren berücksichtigen können, wie beispielsweise Entfernungen oder zeitliche Aspekte.

```
Initialization
the initialization is the same as the main loop, but without data traffic
for each node /* conrurrently on all nodes */ do
    while t _ tmax
     if t mod Dt = 0 then
          select destination node
          launch forward ant
     end if
     for each forward ant /* concurrently for all forward ants */ do
      while current node 6= destination node do
          select link using routing table and link queues
          put ant on link queue
          wait on data link queue
          cross the link
          push the elapsed time information on the stack
      end while
      launch backward ant
      die
     end for
     for each backward ant /* concurrently for all backward ants */ do
      while current node neq destination node do
          choose next node by popping the stack
          wait on high priority link queue
          cross the link
          update the traffic model
          update the routing table
      end while
      die
     end for
    end while
end for
```

Abbildung 9: Algorithmus - AntNet

Das Netz besteht in diesem Modell aus einer Menge von Knoten, wobei jeder Knoten einen oder mehrere Nachbarknoten hat. Das Netz wird von zwei Arten von Ameisen bereist: Es gibt vorwärts und rückwärts laufende Ameisen.

Eine vorwärts laufende Ameise krabbelt von einem Quellknoten zu einem Zielknoten. Der Quellknoten ist für das Erzeugen der virtuellen Ameise zuständig, das Ziel wird dabei zufällig gewählt. Die vorwärts laufenden Ameisen werden, wie der Algorithmus zeigt, zusammen mit dem normal aufkommenden Verkehr durch das Netz geroutet. Die Ameisen befinden sich also in denselben Warteschlangen und benötigen zum Durchqueren des Netzes dieselbe Zeit, wie ein Paket mit Benutzerdaten. So können die Ameisen Informationen über den aktuellen Verkehr sammeln, der sich im Netz befindet. Diese Daten werden bei der Reise auf einem Stack abgelegt. Wenn die Ameise an einem neuen Knoten ankommt, speichert sie ihn auf dem Stack, damit ihr Pfad rekonstruiert werden kann. Zudem wird noch die Zeit festgehalten, die für die Strecke zwischen zwei Knoten benötigt wurde.

Im Gegensatz zum AS und ACS gibt es beim AntNet keine global verfügbaren Informationen, vergleichbar mit den Variablen für die Pheromone. Alle Daten müssen dezentral in den einzelnen Knoten oder von den Ameisen selbst gespeichert werden. Gerät eine Ameise auf ihrem Pfad in einen Zyklus, so löscht sie die entsprechenden Daten von ihrem Stack und läuft normal weiter. Der Weg der Ameise von der Quelle zum Ziel wird durch die Routingtabellen der Knoten bestimmt. Jeder Router hat eine Routingtabelle, in der verzeichnet wird, über welchen Nachbar-Router ein Paket am günstigsten geroutet werden kann, wenn es zu seinem Ziel soll. Die einzelnen Werte sind stochastisch verteilt, so dass die Summe über alle Nachbarknoten für einen bestimmten Zielknoten 1 ergibt. Zudem besitzt jeder Knoten Ergebnisse von Berechnungen über die Zeitspanne, die ein Paket benötigt, um vom aktuellen Punkt an sein Ziel zu gelangen. Zusätzlich zu den Werten wird deren Streuung in einem *Traffic-Model* gespeichert.

Im Algorithmus ist zu sehen, dass eine vorwärts laufende Ameise stirbt, wenn sie am Zielknoten angekommen ist. Zuvor wird von ihr jedoch eine Ameise erzeugt, die denselben Weg durch das Netz zurück läuft. Diese rückwärts laufende Ameise nutzt den Stack ihrer Vorgängerin, um die Knoten in umgekehrter Richtung zu bereisen. Dabei ist die Aufgabe dieser Ameise, die Routingtabellen der Router zu aktualisieren. Auf jedem Knoten wird dazu die aktuell benötigte Zeit mit den bisher errechneten Durchschnitten verglichen. Abhängig davon werden die entsprechenden Einträge in der Routingtabelle angepasst. AntNet wurde nach diesem Algorithmus in *OMNeT++* implementiert und mit verschiedenen anderen Routing-Verfahren verglichen. Die folgende Abbildung zeigt, dass AntNet bezüglich Durchsatz und Delay dabei besser ist, als beispielsweise *OSPF*.

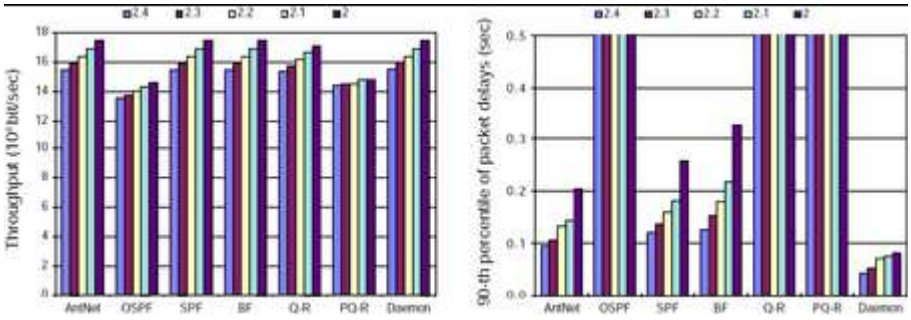

Abbildung 10: Vergleich von AntNet zu anderen Routing-Verfahren

AntNet ist, wie es hier beschrieben wird, in der Praxis nicht im Einsatz. Jedoch gibt es Unternehmen der Telekommunikationsbranche, die Ideen und Ansätze der Schwarmintelligenz und ACO benutzen, um ihre Netze zu optimieren. Beispielsweise haben die France Télécom, British Telecom und MCI-Worldcom solche Verfahren im Einsatz.

Ein weiterer interessanter Ansatz ist das Übertragen des AntNet auf das Verkehrs-Routing in Städten. Die auf AntNet basierenden Navigationsgeräte erlauben es die Autos, gemäß der aktuell vorherrschenden Verkehrslage, auf dem optimalen Weg durch die Stadt zu führen. Auch Bienen besitzen bestimmte Eigenschaften, die für die Optimierung von Routing-Verfahren eingesetzt werden können. Der „*BeeHive*"-Algorithmus arbeitet nach der Idee, wie Bienen ihren Nestkollegen Richtung, Distanz und Qualität einer Futterquelle mitteilen. BeeHive wurde ebenfalls in OMNeT++ simuliert und zeigte sich dort ebenso erfolgreich wie AntNet.

2. Logistik

Logistik ist eine Branche, in der Zeit und Geld eine zentrale Rolle spielen. Ein wichtiger Faktor ist dabei oft die Wegstrecke, die beispielsweise LKWs auf ihren Touren zurücklegen. Dieses Problem präsentiert sich ganz ähnlich zum beschriebenen TSP.

Das „*Ant Colony System for Vehicle Routing Problems with Time Windows*" basiert auf der oben beschriebenen Lösung für das TSP und kann dazu eingesetzt werden, sowohl die Anzahl der benötigten Fahrzeuge, als auch die jeweilige Wegstrecke in einer LKW-Flotte zu optimieren. Im Gegensatz zum ACS bestehen hier also zwei Zielkriterien. Deshalb werden zwei Ameisenkolonien parallel benutzt: Die erste Kolonie versucht einen gültigen Tourenplan zu finden, der mit einem Fahrzeug weniger auskommt, als bisher benötigt wurde. Die zweite Kolonie versucht dann ihrerseits mit dieser Anzahl von Fahrzeugen, die zu fahrende Wegstrecke zu minimieren. Die Knoten, die im Graphen zu bereisen sind, stehen bei dieser Problemstellung für die verschiedenen Kunden. In die Attraktivität einer Route gehen neben dem Abfahrtszeitpunkt vom Kunden auch die Fahrzeit vom aktuellen Kunden zum nächsten Kunden, sowie das verfügbare Zeitfenster, indem Waren

an den Kunden geliefert werden können, ein. Hat einer der beiden Algorithmen eine bessere Lösung gefunden, so initialisiert er den anderen jeweils neu. Dieser übernimmt die Optimierung und versucht dann seinerseits, das Problem neu zu lösen. Wurde eine minimierte Fahrzeugzahl gefunden, so beginnen beide Algorithmen von vorne.

Verfahren die auf ACO basieren werden unter anderem beim führenden italienischen Lebensmittel-Logistiker „Number1 Logistics Group“ und bei der „Siemens Corporate Technology“ eingesetzt.

Es gibt noch eine Vielzahl anderer logistischer Probleme, die auf das TSP zurückzuführen sind. Es geht beispielsweise um den Weg durch ein Warenlager, der es ermöglicht, auf der kürzesten Strecke angeforderte Waren aus ihren Lagerplätzen zu einem Sammelpunkt zu schaffen, von dem sie weiterverarbeitet werden können. Die Minimierung der Wegstrecke führt dabei zur Optimierung der Zeit, die für solche Vorgänge benötigt wird.

Ein ACS kann ebenfalls genutzt werden, um die optimale Reihenfolge von Aufträgen für das „*Scheduling*“ von Querverteilwagen („*Traversing Car*“) in einem Warenlager, wie es in der unteren Abbildung zu sehen ist, zu bestimmen. Solche Querverteilwagen transportieren Waren zwischen mehreren Lagerregalen, verschiedenen Kommissionierplätzen und dem Warenausgang bzw. Wareneingang. Zusätzlich müssen Puffer an bestimmten Stellen berücksichtigt werden. Eine einfache Art zur Organisation der Fahraufträge läuft nach dem FIFO-Prinzip. Ein Verfahren nach ACO ermöglicht eine Optimierung der Bearbeitungs- und Blockadezeiten. Nach jeder Fahrt wird die nächste optimale Fahrt neu errechnet. Dabei stehen die Knoten für die Fahraufträge, die bearbeitet werden müssen.

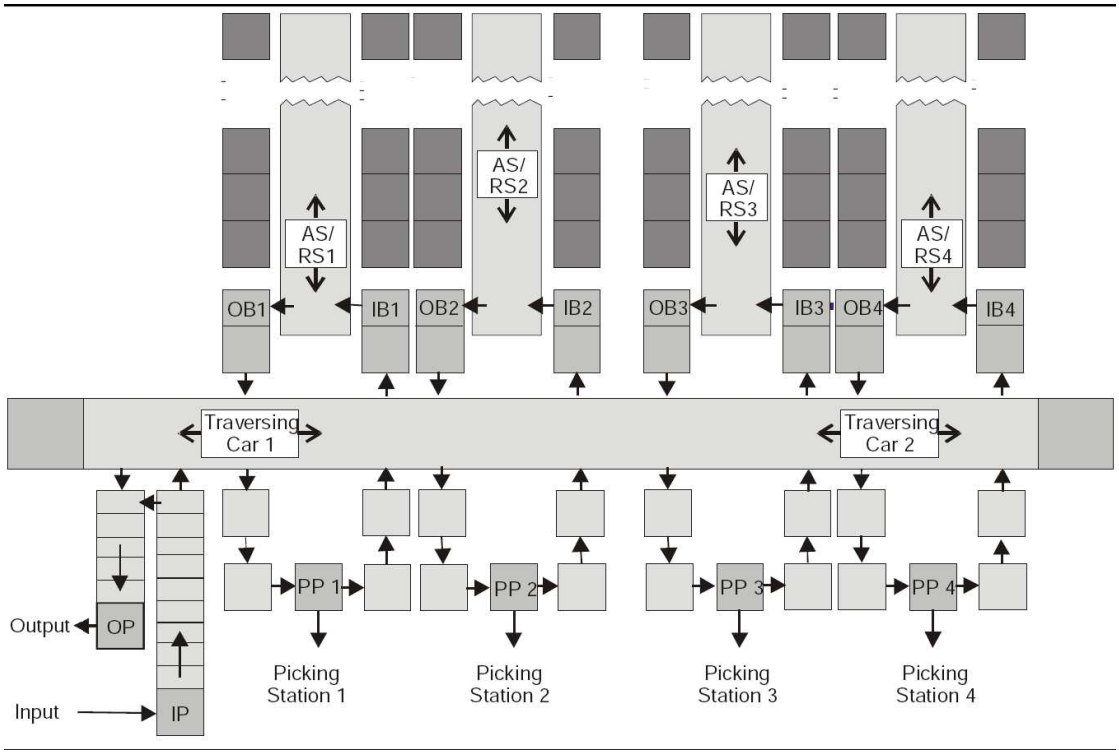

Abbildung 11: Schematische Darstellung eines Lagers

3. Robotik

Anders als bei den zuvor beschriebenen, auf der ACO Metaheuristik basierenden Anwendungen, wird die Schwarmintelligenz in der Robotik mit einem ganz anderen Fokus eingesetzt. Hier interessieren weniger Optimierungsprobleme sondern mehr die *Interaktion* und die gemeinsame Problemlösung im Schwarm.

Inspirieren ließ man sich hierfür beispielsweise vom gemeinsamen Abtransport von Beute, wie es unter anderem bei Ameisen beobachtet wurde. Weitere Vorbilder sind zum Beispiel das Verhalten von Ameisen, die sich gemeinsam zu einer Brücke oder Leiter arrangieren, um andere Schwarmmitglieder das Überqueren von Hindernissen zu ermöglichen oder auch die Organisation von Wespen beim Nestbau.

Die grundlegende Idee ist, dass kleine Roboter, die alleine wegen ihrer begrenzten Fähigkeiten ineffizient wären, sich zu größeren Gruppen zusammenschließen und so im Stande sind gemeinsam komplexe Aufgaben zu lösen, zu denen sie eigenständig nicht in der Lage gewesen wären. Beispiele sind unter anderem das Überwinden von Stufen, die höher sind als die Roboter, das Überschreiten von Spalten, die breiter sind als die Roboter oder das Bewegen von Objekten, die sehr viel schwerer sind als einzelne Roboter. Die untere Abbildung zeigt, wie eine Zusammenarbeit schematisch aussehen könnte (links)

und wie eine konkrete Realisierung durch Robotern mit genannten Fähigkeiten aussieht (rechts).

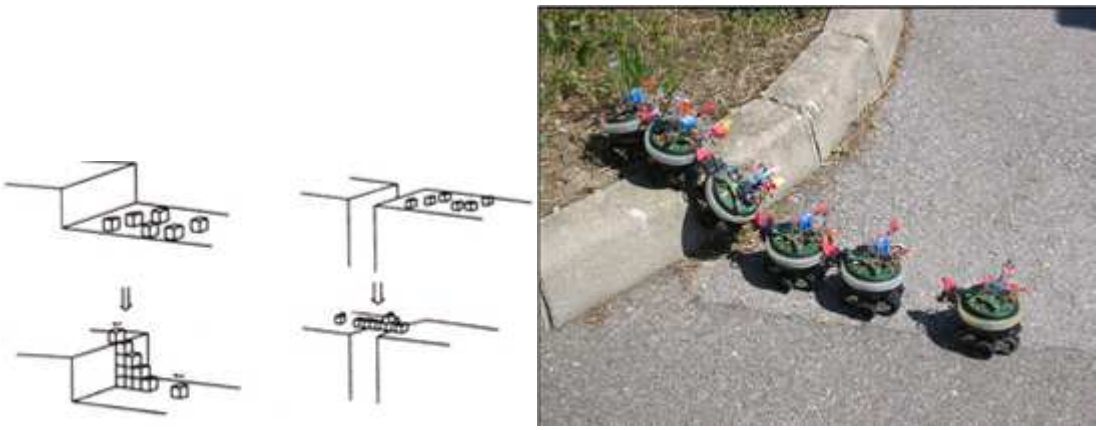

Abbildung 12: Sich selbst organisierende Roboter

Insbesondere werden die Mechanismen zur Kommunikation der einzelnen Schwarmmitglieder von der Natur abgeschaut. Solche Roboterschwärme arbeiten wie natürliche Schwärme dezentralisiert und müssen sich selbst organisieren. Insekten nutzen die Umwelt als Kommunikationsmedium, wie Ameisen die ihre Umwelt durch Pheromone verändern, oder werden direkt von Artgenossen zu Aktionen stimuliert. Diese Stimuli seitens der Umwelt oder der Artgenossen führen dann zum Abspulen bestimmter Verhaltensmuster. Das Aussenden einfacher Signale und das Erfassen der direkten Umwelt sind für simple Roboter, beispielsweise durch Licht und entsprechender Sensorikn möglich.

Die folgenden, recht simplen Verhaltensregeln ermöglichen, dass Roboter die in der Abbildung 12 (links) dargestellte Treppenform errichten, um ein Hindernis zu überwinden.

1. Die Aufwärtsbewegung
   Wenn Roboter „B“ links neben Roboter „A“ steht, sich kein Roboter auf und rechts neben „A“ befindet, dann fordert „A“ „B“ dazu auf, dass „B“ in anheben soll. Falls sich kein Roboter auf „B“ befindet und sich ein anderer Roboter unter oder links neben „B“ befindet, dann hebt „B“ den Roboter „A“ auf sich drauf. Andernfalls verneint „B“ die Anfrage von „A“.

2. Die Linksbewegung
   Wenn ein Roboter „B“ unter Roboter „A“ ist und sich kein Roboter auf oder links neben „A“ befindet, dann bittet „A“ Roboter „B“, dass er ihn nach links bewegt. Falls sich links neben „B“ ein weiterer Roboter befindet, dann flogt „B“ der Aufforderung. Andernfalls verneint „B“ die Anfrage von „A“.

Weitere Anwendungen

Anwendungsmöglichkeiten schwarmintelligenter Verfahren beschränken sich nicht auf die hier vorgestellten Anwendungsgebiete, sondern sind fast so vielfältig wie die Arten der Vorbilder selbst. Optimierungsprobleme beispielsweise existieren in sehr vielen Branchen.

Das Verhalten geschickter Sortiervorgänge hat man ebenfalls schon von Insekten abgeschaut und in künstliche Systeme übertragen. Bestimmte Gattungen von Ameisen sortieren ihre Larven nach Größe oder sammeln tote Schwarmmitglieder an einem zentralen Punkt. Dies inspirierte Forscher zu einem Modell, mit diesem Banken ihre Finanzdaten analysieren können.

Der „*AntTree*“ ist ein Algorithmus, der für das Bilden von hierarchischen Clustern entwickelt wurde. Dieses Verfahren beruht ebenfalls auf dem Verhalten von Ameisen, die sich zur Lösung von komplexen Aufgaben zusammenschließen. AntTree gruppiert (*clustert*) Dokumente um eine Suche in ihnen zu ermöglichen. Dabei werden diese Dokumente selbst wie Ameisen modelliert, die wiederum die Verbindung zu anderen „Ameisen Dokumenten“ herstellen. Es wurde schon ein webbasiertes Suchportal entwickelt, dass diese Verbindungen zwischen den „Ameisen Dokumenten“ verwendet.

Das Problem der optimalen Lastverteilung bei Clustern von Webservern wurde durch ein Modell gelöst, das auf dem Verhalten der Bienen beim Sammeln von Nektar beruht. Eine weitere realisierte Anwendung ist die Arbeitsteilung in einer Fertigungsfabrik, die nach der Arbeitsteilung eines Bienenstaats gestaltet wurde und so ihre Effizienz deutlich steigern konnte.

Fazit und Ausblick

Als vor etwa 15 Jahren die ersten Forscher damit begonnen haben das Verhalten von Ameisen in Algorithmen nachzubilden um damit komplexe Probleme zu lösen, sprach man von einer „verrückten“ Idee. Die erfolgreiche Anwendung auf verschiedenste Probleme zeigte jedoch, dass ACO weit mehr als nur eine verrückte Idee war.

Die Zukunft des ACO liegt nach Meinungen der Forscher unter anderem darin, auch die Klasse der dynamischen Probleme optimal lösen zu können. Eine der größten Stärken der ACO Algorithmen ist ihre Fähigkeit zur *Selbstadaption*. So ist es beispielweise möglich im laufenden Betrieb, ohne Neustart der Algorithmen, die Problemstellung zu verändern und dabei immer noch effizient die optimale Lösung zu ermitteln. Hinzu kommt der Ansatz, dass bestimmte Variablen auch stochastisch verteilt werden können, wie beispielsweise im „*Probalistic Travelling Salesman Problem*“ (PTSP), bei dem Städte mit einer bestimmten Wahrscheinlichkeit bereist werden müssen. Weitere Forschungsfelder sind ACO-Algorithmen, die parallel (d.h. auf mehreren Prozessoren gleichzeitig) ablaufen können, Optimierungsprobleme in mehreren Dimensionen sowie kontinuierliche Datenräume im Gegensatz zu den diskreten Problemen wie das TSP.

Auch die Übertragung von Schwarmintelligenz in den Entwurf verteilter Systeme verspricht sehr gute Entwicklungsmöglichkeiten. Am Fraunhofer-Institut für Arbeitswirtschaft und Organisation in Stuttgart wurde Ende Januar 2007 beispielsweise das Projekt „*IWARD*“ initiiert. Hier sollen Roboter entwickelt werden, die in Krankenhäusern aktiv sind und beispielsweise Ärzte suchen, Krankenschwestern rufen, die Zimmer sauber halten oder die Besucher führen. Sie sollen auf Basis der Schwarmintelligenz einerseits autonom agieren können und andererseits auch ständig im Kontakt mit den Kollegen stehen, um gemeinsam komplexere Aufgaben lösen zu können.

Die Forschung im Bereich der Schwarmintelligenz bleibt also weiterhin sehr agil. Solange selbst in der Biologie noch nicht jedes Verhaltensmuster innerhalb der Schwärme entschlüsselt wurde, sind sicherlich auch die Übertragungsmöglichkeiten von natürlichen in künstliche Systeme noch nicht erschöpft. Beispielsweise ist noch unbekannt, auf welche Art und Weise das Verhalten von Ameisen durch die Temperatur beeinflusst wird. Allerdings konnte man bereits beobachten, dass diese scheinbar Auswirkungen hat.

Nach Meinungen führender Wissenschaftler erlangen einige Verfahren der Schwarmintelligenz, die nach dem Vorbild natürlicher Schwärme entwickelt wurden, in etwa 5 bis 10 Jahren Marktreife. Damit steht dieses Themengebiet auf einer ähnlichen Entwicklungsstufe wie beispielsweise „biometrisches Bezahlen“ oder der Einsatz von RFID für Einzelprodukte.

Schluss

Wir hoffen mit dieser Ausarbeitung einen Einblick in das Thema der „Nature Patterns“ verschafft und deutlich gemacht zu haben, welches ungeheure Potential die Natur der Informationstechnik bietet. Es ist durchaus denkbar, dass natürliche Vorbilder in geraumer Zeit die Informationstechnik revolutionieren werden. Wir sind gespannt auf diese Entwicklung und stehen für Fragen oder mehr Informationen zu dem Thema gerne zur Verfügung.

Valentin Schwind & Boris Wüst

Quellen

Bionik
http://www.uni-saarland.de/fak8/bi13wn/wabionik.htm#BM1
http://de.wikipedia.org/wiki/Bionik

Klettverschluss
http://de.wikipedia.org/wiki/Klettverschluss

Lotus-Effekt
http://de.wikipedia.org/wiki/Lotuseffekt

Evolutionäre Algorithmen
http://www.fh-meschede.de/public/willms/ea/ea.html
http://de.wikipedia.org/wiki/Evolution%C3%A4rer_Algorithmus

Genetische Programmierung
http://www.genetic-programming.com

Neuronale Netze
http://www.neuronalesnetz.de

PAPINI, Emanuele: Optimieren um zu konkurrieren
http://www.antopt.com/admin/pdfeventi3/Seminar.pdf

Swarm-bots project homepage
http://www.swarm-bots.org

Bee Colony Optimization
http://en.wikipedia.org/wiki/Bee_Colony_Optimization

Emergenz
http://de.wikipedia.org/wiki/Emergenz

Ameisenalgorithmus
http://de.wikipedia.org/wiki/Ameisenalgorithmus

Ameisen
http://de.wikipedia.org/wiki/Ameisen
http://www.morgenwelt.de/wissenschaft/001019-ameisen.htm

Kollektive Intelligenz in der Informatik
http://de.wikipedia.org/wiki/Kollektive_Intelligenz#Kollektive_Intelligenz_in_der_Informatik

Schwarmintelligenz
http://en.wikipedia.org/wiki/Swarm_intelligence