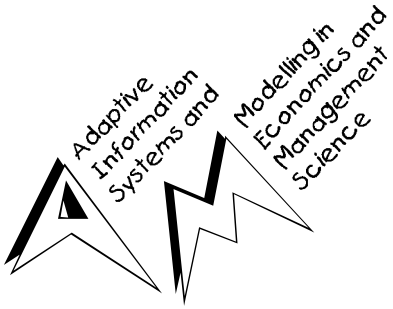
Gaining knowledge and winning with knowledge

Fent, Thomas

Vienna University of Economics and Business Administration

August 2000

Working Paper Series

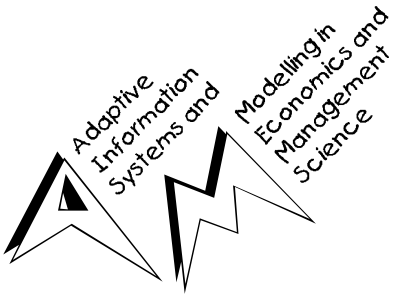


Wissen gewinnen und gewinnen durch Wissen

Thomas Fent

Working Paper No. 72
August 2000

Working Paper Series



August 2000

SFB
'Adaptive Information Systems
and Modelling in Economics and Management Science'

Vienna University of Economics and Business Administration
Augasse 2-6
1090 Vienna
Austria

http://www.wu-wien.ac.at/am

This piece of research was supported by the
Austrian Science Foundation (FWF) under grant SFB # 010
('Adaptive Information Systems and Modelling in Economics and
Management Science').

Thomas Fent

Wissen gewinnen und gewinnen durch Wissen

Abstract

Gemäß Alfred Korzybski (1921) unterscheidet sich der Mensch von Pflanzen und Tieren unter anderem durch seine Eigenschaft als "Zeit-Binder". Diese befähigt ihn, Erfahrung durch die Zeit zu transportieren. Menschen können Wissen aus der Vergangenheit ansammeln und das, was sie wissen, der Zukunft mitteilen. In der vorliegenden Arbeit werden die Möglichkeiten und Grenzen untersucht, diese Fähigkeit durch Algorithmen zu beschreiben, und in künstlichen lernenden Systemen zu implementieren. Zur Illustration wird abschließend aufgezeigt, wie ein künstlicher Agent und seine Umgebung beschaffen sein können, um ihm das Erlernen einer erfolgreichen Strategie in einem einfachen Nimm-Spiel zu ermöglichen.

Wissen gewinnen

Neugeborene verfügen zunächst über nahezu kein Wissen - außer ihren angeborenen Instinkten. Angetrieben durch ihre kindliche Neugierde beobachten sie Ihre Umwelt, und versuchen diese durch eigene Handlungen zu beeinflussen. Die Reaktionen der Umwelt werden ebenfalls beobachtet. Viele dieser Beobachtungen werden im Gehirn abgespeichert, und es bilden sich allmählich Zusammenhänge heraus. Das Kind lernt aus einer gegebenen Situation vorherzusagen, welche Resultate die unterschiedlichen möglichen Handlungsweisen nach sich ziehen. Wenn eine völlig neue Situation auftritt, dann wird (möglicherweise) versucht, Vergleiche mit einer zumindest ansatzweise ähnlichen Situation anzustellen. Wenn auch das nicht weiterhilft, dann wird das Kind eine beliebige Handlung ausprobieren (möglicherweise besteht die Handlung darin, nichts zu tun), und aufgrund dieser einen Erfahrung eine komplett neue Regel bilden. Natürlich können sich diese Regeln im Laufe der Zeit auch verändern, etwa durch Vergessen, Ausprobieren neuer Handlungen (Neugierde) oder wenn sich eine Vorhersage nachträglich als unrichtig herausgestellt hat.

Maschinenlernen

Algorithmen aus dem Bereich des Maschinenlernens (ML) versuchen diese Vorgangsweise zu imitieren. *Maschinenlernen befasst sich mit Computer-Algorithmen, die sich durch Erfahrung automatisch verbessern.* Eine Regelbasis wird zunächst zufällig initialisiert, und anschließend durch Lernalgorithmen verbessert. Bei Bedarf können auch einige fertige Regeln bereits bei der Initialisierung vorgegeben werden. Wenn alle Regeln vorgegeben sind, so spricht man von künstlicher Intelligenz, wenn der Computer die Regeln erst erlernen soll, von Maschinenlernen. Ersteres ist z.B. in Expertensystemen verwirklicht. Dabei wird grundsätzlich menschliches Wissen in einem Computersystem kodiert. ML-systeme hingegen lernen aus der Erfahrung und machen (sinnvolle) Vorhersagen über die Umwelt.

Eine der Schwierigkeiten besteht darin, alle relevanten Umweltzustände, alle möglichen Handlungen und alle möglichen Resultate in einer für den Computer lesbaren Form (durch

Zahlen) zu beschreiben. "Wenn es jemals möglich sein sollte, eine Maschine zu konstruieren, die in der Lage ist, menschliche Sprache zu sprechen, zu verstehen oder zu übersetzen, mathematische Probleme durch Vorstellungskraft zu lösen, einen Beruf auszuüben oder eine Organisation zu leiten, dann müssen wir diese Aktivitäten entweder auf einen Bereich reduzieren, der so exakt ist, daß wir der Maschine genau mitteilen können, wie in welcher Situation verfahren werden soll, oder wir müssen eine Maschine entwickeln, die Dinge tun kann, ohne daß ihr genau vorgegeben wurde, wie ein Problem zu lösen ist. Stattdessen muß der Maschine genau mitgeteilt werden, wie sie zu lernen hat." (Friedberg 1958)

Der Prozess des ML beginnt mit der Identifikation des Lernbereiches und endet mit dem Testen und Anwenden der Ergebnisse des Lernprozesses. Ein Lernbereich ist irgend ein Problem oder eine Menge von Fakten, wo es möglich ist, die zu messenden Merkmale und die vorherzusagenden Resultate zu identifizieren. Natürlich muß dabei zwischen den Merkmalen und den Resultaten ein Zusammenhang bestehen. Ein ML-system arbeitet die Lernmenge (eine Teilmenge des Lernbereiches) ab, und versucht aus diesen Beispielen zu lernen. Die Testmenge (Validierungsmenge) stammt aus dem selben Lernbereich wie die Lernmenge. Die Validierungsdaten werden benutzt, um zu testen, ob das ML-system tatsächlich einen sinnvollen Zusammenhang zwischen Merkmalen und Ergebnissen gelernt hat, oder lediglich die Lernmenge abgespeichert hat. Die Fähigkeit, gelernte Zusammenhänge auf neue Datenmengen anzuwenden nennt man Generalisierung. Die verschiedenen bekannten ML-systeme unterscheiden sich vor allem durch die verwendeten Lernalgorithmen.

Darstellung des Problems

Die Darstellung eines ML-Systems umfasst die Definition, wie mögliche Lösungen des Problems aussehen, welche Inputs die Lösungen akzeptieren, wie die Inputs verarbeitet werden und wie die Outputs produziert werden. Einige mögliche Darstellungsformen sind

- Boole'sche Ausdrücke (boolean representations),
- Schwellwerte (threshold representations),
- Fallunterscheidungen (case-based representations),
- Baumdiagramme (tree representations) und
- Genetische Darstellungen (genetic representations).

Boole'sche Ausdrücke

Nehmen wir an, ein Wissenschafter möchte bestimmen, ob eine bestimmte Person in einem Dick Tracy-Cartoon ein "bad guy" oder ein "good guy" ist. Der Wissenschafter untersucht etliche Jahrgänge von alten Dick Tracy Comics, und erkennt die folgenden Merkmale als hilfreich:

- gerissener Blick
- narbiges Gesicht
- Kopftätowierung
- lässiger Gang
- krumme Nase
- Zweibandfunkgerät am Handgelenk

Alle diese Eigenschaften können mit Hilfe von Boole'schen Variablen (richtig - falsch) kodiert werden. Im folgenden Ausschnitt weist die Figur im ersten Bild ganz links eine krumme Nase und einen gerissenen Blick auf. Handelt es sich dabei um einen "bösen Buben"?

Abbildung 1

Es gibt zwei Grundtypen von Boole'schen Systemen.

Verbindende Boole'sche Systeme - conjunctive Boolean systems (cbs)

In einem cbs werden die Merkmale durch den logischen Operator **UND** verknüpft. Ein Lernalgorithmus könnte möglicherweise folgende Konzepte zur Unterscheidung zwischen "bad guys" und "good guys" gefunden haben:

Konzept 1	gerissener Blick **UND** narbiges Gesicht **UND** Kopftätowierung
Konzept 2	krumme Nase **UND** Zweibandfunkgerät am Handgelenk

Die Konzepte allein reichen nicht aus. Die Konzepte müssen auch interpretiert bzw. klassifiziert werden, und die Klassifikation selbst kann wiederum auf unterschiedliche Weise dargestellt werden. Dick Tracy könnte mit den beiden Konzepten folgendermaßen verfahren. Konzept 1 stimmt mit seinem persönlichen "crime watchers guide" überein. Konzept 2 hingegen passt auf ihn selbst. Dick Tracy würde daher zur Unterscheidung zwischen Verdächtigen und tatsächlich Bösen die Konzepte wie folgt anwenden:

Konzept	Wert	bad guy?
1	richtig	richtig
2	richtig	falsch

Keines dieser beiden Konzepte würde die erwähnte Figur in Abb. 1 erfassen. In einem cbs müssen stets alle Merkmale aus einem Konzept erfüllt sein, um das Konzept anwenden zu können.

In der Notation eines *classifier systems* (cs) könnten die beiden Konzepte wie folgt aussehen:

1 1 1 # # # | 1
1 1 | 0

Um so ein cs verwenden zu können, muß die zu untersuchende Person zunächst bezüglich aller in der vorangegangenen Liste erwähnten Eigenschaften beurteilt werden. Die Resultate werden in einem Zeilenvektor der Länge 6 erfaßt. Ein Wert 1 bedeutet, daß die Eigenschaft erfüllt ist, 0 bedeutet, daß sie nicht erfüllt ist. Dieser Vektor (Input) wird nun mit dem Bedingungsteil des cs – der Teil links vom | - verglichen. Stimmt der Inputvektor an allen, bis auf den mit # gekennzeichneten Stellen, mit dem Bedingungsteil überein, so gilt die Regel als erfüllt. Die # heißen don't care Symbole. Diese Bezeichnung bringt zum Ausdruck, daß dem Wert, den der Input-Vektor an dieser Stelle annimmt, keine Bedeutung beigemessen wird.

Aus allen Regeln, die durch den Input erfüllt werden, wird eine zufällig ausgewählt. Bei diesem Zufallsprozeß werden i.a. Gewichte aufgrund der Erfolge in der Vergangenheit berücksichtigt. Diese zufällig ausgewählte Regel darf nun ihren Aktionsteil als Output an die Umwelt verschicken. Dieser Aktionsteil kann natürlich wieder ein mehrdimensionaler Vektor sein. Dieser Outputvektor kann auf unterschiedliche Weise interpretiert werden. Es kann sich dabei z.B. um die Vorhersage eines bestimmten Systemzustandes (Wetterprognose) handeln, oder die Zahlenwerte im Outputvektor dienen der Kodierung einer bestimmten Handlung (z.B. Springer von G8 auf F6).

Im obigen Beispiel hat der Outputvektor nur eine Stelle. Eine 1 bedeutet „bad guy“ und 0 symbolisiert „good guy“. In komplexen Entscheidungssystemen liegen auch die Entscheidungen (Outputs) in höherdimensionalen Räumen. Folglich müssen dann auch die Outputvektoren entsprechend viele Komponenten aufweisen.

Trennende Boole'sche Systeme - disjunctive Boolean systems(dbs)

In einem dbs werden die Merkmale durch den logischen Operator **ODER** verknüpft. Wenn in einem dbs eines der simplen erlernten Konzepte erfüllt ist, dann wird der Output ebenfalls als **richtig** definiert. Betrachten wir dazu die folgenden drei Konzepte:

Konzept	Merkmal	Wert
Konzept 1	gerissener Blick	**richtig** oder **falsch**
Konzept 2	narbiges Gesicht	**richtig** oder **falsch**
Konzept 3	Kopftätowierung	**richtig** oder **falsch**

Vergleichen wir die Konzepte wieder mit dem bereits bekannten zwielichtigen Herrn aus der Abbildung, so sehen wir, daß Konzept 1 erfüllt ist, der output wird daher als **richtig** gesetzt. Wie das interpretiert werden soll (good vs. bad guy) bleibt der persönlichen Erfahrung des Lesers überlassen.

Wie das folgende Beispiel zeigt, lassen sich auch dbs durch cs implementieren.

```
1 # # # # # | 1
# 1 # # # # | 1
# # 1 # # # | 1
```

Durch die Kombination von cbs und dbs lassen sich noch weit komplexere Zusammenhänge beschreiben.

Schwellwerte

Repräsentationen durch numerische Schwellwerte sind weit mächtiger – weil flexibler – als binäre Repräsentationen. Eine Schwellwert-Einheit produziert immer dann einen Output, wenn der Input den Schwellwert übersteigt. Z.B. wird eine durch einen Thermostat gesteuerte Heizungsanlage immer dann abgeschaltet, wenn der Temperatursensor eine über dem vorgegebenen Schwellwert liegende Temperatur meldet. In einem ML-system kann eine Schwellwert-Einheit dazu dienen, einen Input vorzubereiten, bevor er in das System eingeleitet wird. Z.B. kann verlangt werden, daß von den in der Aufgabenstellung des vorangegangenen Abschnitts erwähnten sechs Eigenschaften mindestens vier erfüllt sein müssen. So einen Zusammenhang durch ein *classifier system* zu beschreiben ist zwar möglich, aber äußerst ineffizient.

1 1 1 1 # # | 1
1 1 1 # 1 # | 1
1 1 # 1 1 # | 1
1 # 1 1 1 # | 1
1 1 1 1 # | 1
1 1 1 # # 1 | 1
1 1 # 1 # 1 | 1
1 # 1 1 # 1 | 1
1 1 1 # 1 | 1
1 1 # # 1 1 | 1
...

In mehrschichtigen Neuronalen Netzwerken (z.B. multilayer perceptron) werden die Daten grundsätzlich in allen Abschnitten durch Schwellwert-Einheiten verarbeitet.

Fallunterscheidungen

Weitere gängige Methoden des Maschinlernens speichern die Daten als Repräsentanten von Klassen, oder sie speichern überhaupt nur allgemeine Beschreibungen der Klassen, indem die Lerndaten in irgendeiner Form gemittelt werden. Beim Ansatz der K-nächsten Nachbarn werden Z.B. die Daten selbst als Teil der Problemrepräsentation verstanden. Ein neuer Input wird klassifiziert, indem er der Klasse zugewiesen wird, in der sich die meisten der K nächsten Elemente befinden. Eine weitere Möglichkeit besteht darin, den Lernbereich durch Hyperebenen in Klassen zu unterteilen. Liegen diskrete Daten zugrunde, so läßt sich diese Form der Klasseneinteilung durch ein cs implementieren.

Beispiel: Nehmen wir an der Lernbereich sei die Menge {0,1,2,3,4,5,6,7}^2, weiters seien im Lernbereich 2 Klassen **A** und **B** definiert durch (siehe Abb.2)

A: $y > 2\frac{1}{4} + \frac{1}{2}x$,

B: $y < 2\frac{1}{4} + \frac{1}{2}x$.

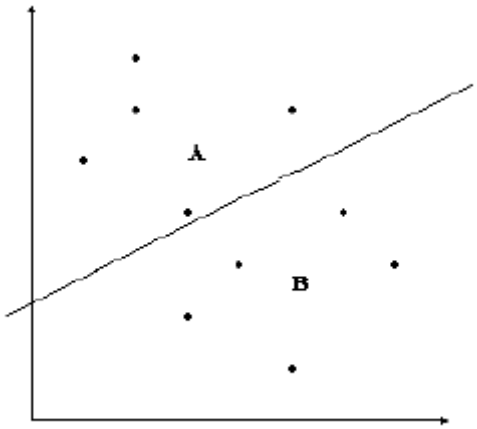


Abbildung 2

Beschreibt man die Koordinaten eines Elementes aus {0,1,2,3,4,5,6,7}^2 durch Vektoren von Binärecodes (e.g. (3,4) $\hat{=}$ (0 1 1 1 0 0)), dann kann obige Klasseneinteilung durch das folgende cs vorgenommen werden:

0 0 # # 1 1 | A
0 1 # 1 # # | A
1 0 # 1 # 1 | A
1 1 # | A
0 0 # | B
0 1 0 | B
1 # 0 1 1 | B
1 # # 1 0 0 | B
1 1 # 1 0 1 | B

Baumdiagramme

Da sich viele Entscheidungssituationen übersichtlich in Form eines Entscheidungsbaumes veranschaulichen lassen, benutzen auch viele ML-systeme Baumdarstellungen. Jeder Knoten des Baumes stellt ein Merkmal dar. Jede Kante steht für einen möglichen Wert der durch den darüberliegenden Knoten gekennzeichneten Eigenschaft. Jeder Endknoten (in einem richtigen Baum wären das die Blätter) stellt eine Klasse dar.

Wenn wir wieder an das Klassifizierungsproblem aus Dick Tracy denken, so steht jeder Knoten für eine der Eigenschaften wie z.B. gerissener Blick, narbiges Gesicht usw. Die unmittelbar mit dem Knoten verbundenen Zweige nehmen die Werte **richtig** oder **falsch** an. Die beiden Konzepte aus dem Abschnitt über cbs könnten in einem Baumdiagramm wie in Abb. 3 aussehen.

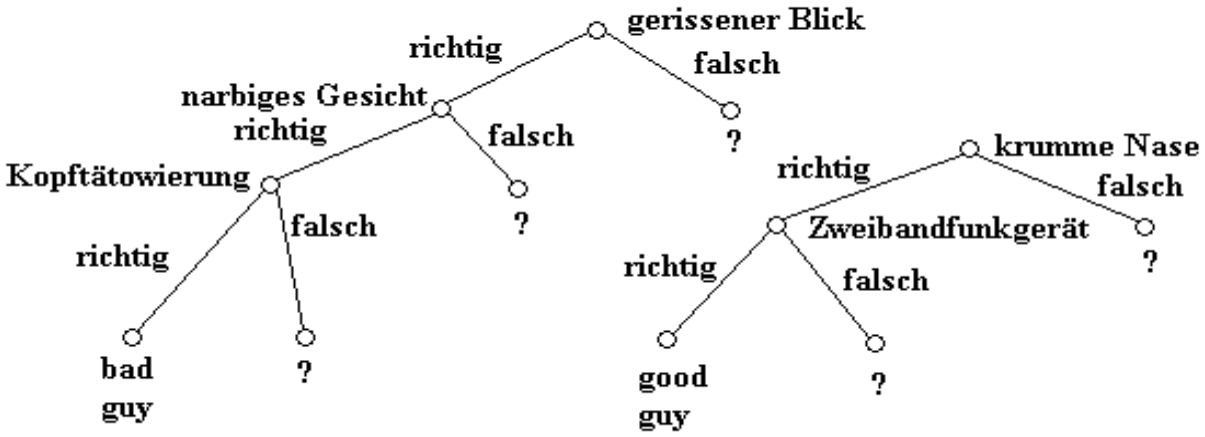


Abbildung 3

Genetische Darstellungen

Genetische (evolutionäre) Darstellungen werden auf vielfältige Weise angewendet. So verwenden z,B. genetische Algoritmen (GA) zumeist binäre Strings konstanter Länge. Jedem Bit wird eine Bedeutung zugeordnet. Da diese Zuordnung vollkommen frei erfolgen kann, stellt diese Form der Darstellung ein hohes Maß ein Freiheit zur Verfügung. Wie bereits teilweise gezeigt wurde, können alle bisher erwähnten Darstellungen in geeigneter Form in eine äquivalene genetische Darstellung transformiert werden. Die GA selbst benutzen überhaupt keine Information, die mit der Bedeutung der Bits in Zusammenhang steht. Darstellungen, bei denen die Bits mehr als nur die Werte 0 und 1 annehmen können - z.B. ganze Zahlen, oder überhaupt alle reellen Zahlen - führen zumeist zu einer Steigerung der Effizienz im Vergleich zur rein binären Darstellung.

Suchverfahren

Die Wahl einer geeigneten Darstellung (Kodierung) des Problems allein ist meist völlig wertlos. Im nächsten Schritt benötigt man ein geeignetes Suchverfahren, um sich im Lösungsraum fortzubewegen. Eine Auswertung des gesamten abzusuchenden Raumes ist normalerweise aus Zeitmangel nicht möglich. Die Suchoperatoren legen fest wie und in welcher Reihenfolge ein ML-system die möglichen Lösungen auswählt. Es ist einleuchtend, daß ein gutes ML-system Suchoperatoren benutzt, die einen Pfad durch den Lösungsraum wählen, der gute Lösungen auffindet und schlechte umgeht. Im wesentlichen gliedern sich die zur Verwendung kommenden Suchstrategien in drei Gruppen.

Blinde Suche (blind search)

Bei der blinden Suche werden die Elemente aus dem Lösungsraum ausgewählt, ohne irgend welche Informationen über die Struktur des Problems oder Resultate aus vorangegangenen Lernschritten zu benutzen. In einem Baum begrenzter Größe kann mittels blinder Suche oft in kurzer Zeit eine sinnvolle Lösung gefunden werden.

Hügel erklimmen (hill climbing)

Beim hill-climbing wird jeweils die beste bisher gefundene Lösung abgespeichert. Neue Lösungen, die schlechter sind als die bisher beste Variante werden zumeist gleich wieder verworfen und nicht abgespeichert. Diese Strategie kommt z.B. im Simulated Annealing und in vielen für Neuronale Netze gängigen Trainingsalgorithmen (z.B. Backpropagation) zur Anwendung. Zu jedem Zeitpunkt wird nur eine Lösung berücksichtigt, und es existieren keine Aufzeichnungen über frühere Lösungen.

Suche entlang von Strahlen (beam search)

Beim beam search wird immer eine beschränkte Population von Lösungen abgespeichert. Selbst weniger erfolgreiche Varianten haben die Chance über mehrere Generationen erhalten zu bleiben. Ein Evaluierungsverfahren trifft die Auswahl, welche Lösungen beibehalten, und welche verworfen werden. Das bedeutet, der abzusuchende Raum wird auf die in der Population vertretenen Repräsentanten eingeschränkt. Der Bedarf an Speicherplatz steigt dadurch zwar drastisch an, andererseits sind aber weniger Auswertungen notwendig - das wirkt sich insbesondere bei sehr komplexen Lösungsräumen vorteilhaft aus.

Gewinnen durch Wissen - Implementierung einer optimalen Nimm-Strategie

Spielregeln

In diesem Abschnitt wird ein gängiges Nimm-Spiel untersucht. Am Beginn des Spieles werden mehrere Reihen von Streichhölzern aufgelegt (siehe z.B. Abb. 4). Die beiden Spieler entfernen abwechselnd von einer der Reihen beliebig viele Streichhölzer. Es muß immer mindestens ein Streichholz weggenommen werden, und es können maximal so viele Streichhölzer entnommen werden, als sich noch in der entsprechenden Reihe befinden. Der Spieler, der zuletzt Streichhölzer entnimmt, hat gewonnen.

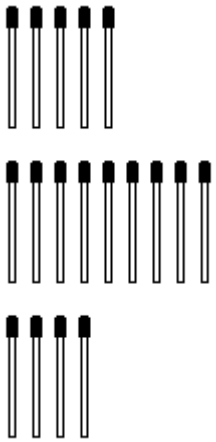


Abbildung 4

Die optimale Strategie

Zunächst werden die Anzahlen der in den Reihen liegenden Streichhölzer in binärer Notation niedergeschrieben (siehe Abb.5, rechte Tabelle). Anschließend werden die Einser in jeder Spalte addiert. Befindet sich in jeder Spalte eine gerade Anzahl von Einsern, so ist das Spiel *in Balance*. Die in Abb. 4 dargestellte Situation ist nicht in Balance, da sich in der Spalte ganz links (2^3) nur ein Einser befindet.

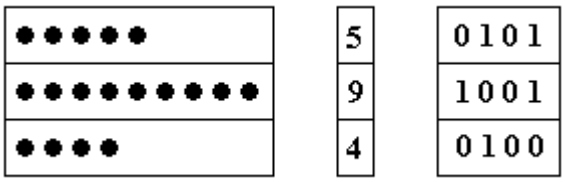


Abbildung 5

Am Ende des Spieles - alle Reihen sind leer - ist das Spiel in Balance. Der Spieler, der zum Zug kommt, wenn sich nur mehr in einer Reihe Streichhölzer befinden, hat die Möglichkeit zu gewinnen. Man kann also nur gewinnen, wenn man das Spiel nicht in Balance vorfindet. Aus einer unbalancierten Situation ist es stets möglich mit einem Zug die Balance wieder herzustellen (ohne Beweis). Weiters ist sofort ersichtlich, daß ein Spiel, das in Balance ist, nach einem beliebigen zulässigen Zug nicht mehr in Balance ist.

Wie sich mittels der von Zermelo entwickelten Rückwärtsrekursion zeigen läßt (siehe Binmore, 1992), besteht die optimal Strategie darin, das Spiel immer wieder in einen balancierten Zustand überzuführen. Kennen beide Spieler die optimale Strategie, und halten diese auch ein, so hängt das Ergebnis davon ab, wer beginnt, und ob sich das Spiel am Beginn in Balance befindet. In Abbildung 6 ist ein Spielverlauf dargestellt, bei dem Spieler I die optimale Strategie anwendet.

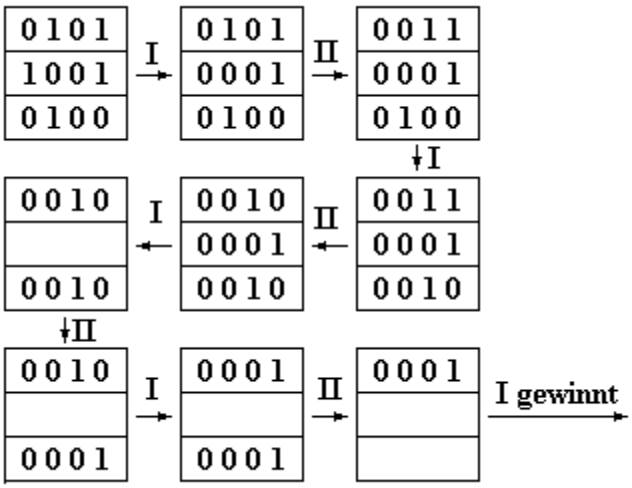


Abbildung 6

Implementierung der optimalen Strategie

Wenn sich das Spiel in Balance befindet, so gibt es offensichtlich keine Möglichkeit eine Gewinn-strategie anzuwenden. In diesem Fall muß man irgend einen Überbrückungszug machen. Kennt der Gegner die optimale Strategie, so hat man in diesem Fall keine Chance, das Spiel im weiteren Verlauf in einer unbalancierten Situation vorzufinden. Ist das Spiel nicht in Balance, so ist ein Zug zu wählen, der das Spiel in Balance bringt. Dazu ist es notwendig, die Anzahl von Streichhölzern in einer der Reihen so anzupassen, daß sie die übrigen Reihen so ergänzt, daß sich in jeder Spalte der binären Darstellung eine gerade Anzahl von Einsern befindet. Nehmen wir an, es liegt ein Spiel mit n Reihen vor. Bezeichnen wir mit x_{ij} die j-te Komponente der Binärdarstellung in Reihe i, und sei k die Reihe, deren Anzahl anzupassen ist. Der optimalen Strategie folgend muß gelten

$$x_{kj} = \left(\sum_{\substack{i=1 \\ i \neq k}}^{n} x_{ij} \right) \bmod 2 \quad \forall j \,. \tag{1}$$

Das ist aber nur dann möglich, wenn dieses neu zu wählende x_{kj} kleiner (echt kleiner!) als die ursprüngliche Anzahl in Zeile k ist. In den meisten Fällen wird es daher sinnvoll sein, die Anpassung in der Reihe vorzunehmen, in der sich noch die meisten Streichhölzer befinden.

Beschränken wir uns auf Nimm-Spiele mit 3 Zeilen und maximal 7 Streichhölzern pro Zeile, so gibt es insgesamt 10 Fälle, in denen es nicht möglich ist, die Reihe mit den meisten Streichhölzern anzupassen (Fälle, in denen die Zeilen bloß vertauscht sind, werden als äquivalent angesehen). Diese 10 Fälle sind in Tabelle 1 aufgelistet.

Fall	**1**		**2**		**3**		**4**	
	1	001	1	001	1	001	2	010
	2	010	4	100	6	110	4	100
	2	010	4	100	6	110	4	100
Fall	**5**		**6**		**7**		**8**	
	2	010	2	010	2	010	3	011
	4	100	5	101	5	101	4	100
	5	101	5	101	6	110	4	100

Fall	9		10					
	3	011	3	011				
	4	100	4	100				
	5	101	6	110				

Tabelle 1

Mit Ausnahme von Fall 7 ist es in allen in der Tabelle angeführten Spielsituationen möglich, das Spiel in Balance zu führen, indem man die Zeile mit der geringsten Anzahl von Streichhölzern gemäß Formel 1 anpaßt. Im Fall 7 bleibt schließlich nur mehr die eine Möglichkeit, die Zeile mit der mittleren Anzahl anzupassen. Sortieren wir die Zeilen nach absteigender Anzahl von Streichhölzern (d.h. Zeile 1 enthält die meisten, Zeile 3 die wenigsten), dann kann mit folgendem Algorithmus der Rechenaufwand minimiert werden:

1. Berechne x_1^{neu} gemäß Formel (1).
2. Gilt $x_1^{neu} < x_1^{alt}$, wähle x_1^{neu} $\Rightarrow$ Fertig!
3. Berechne x_3^{neu} gemäß Formel (1).
4. Gilt $x_3^{neu} < x_3^{alt}$, wähle x_3^{neu} $\Rightarrow$ Fertig!
5. Berechne x_2^{neu} gemäß Formel (1).
6. Gilt $x_2^{neu} < x_2^{alt}$, wähle x_2^{neu} $\Rightarrow$ Fertig!

Abbildung 7 zeigt einen Snapshot der entsprechenden MATLAB Implementierung.

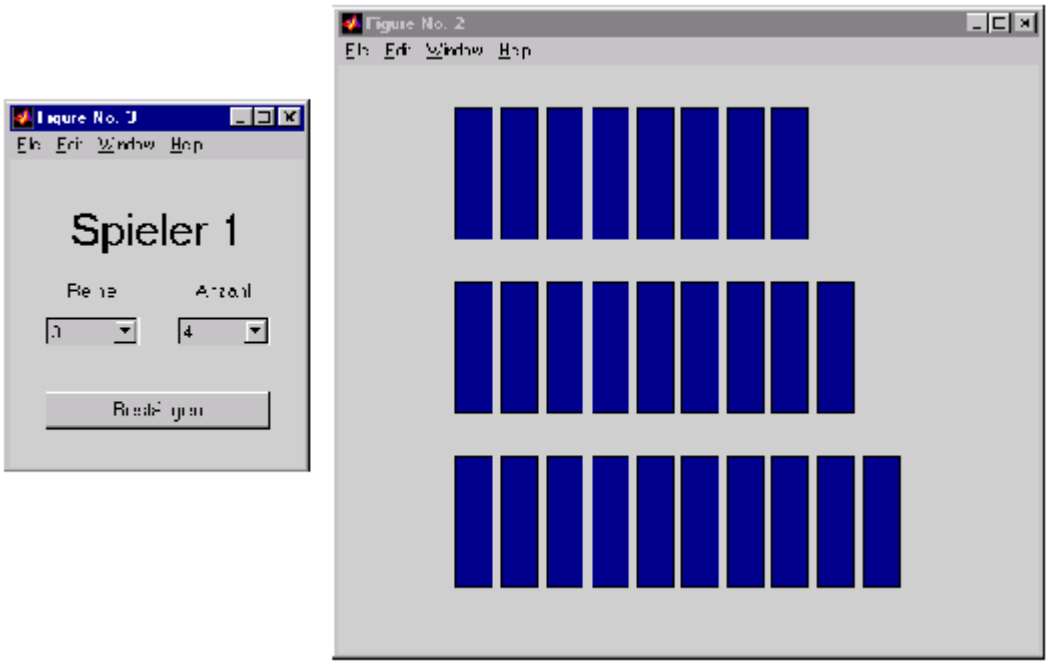


Abbildung 7

Lernumgebung

Ein lernender Agent wird am Beginn mit zufälligen Regeln initialisiert, und versucht diese aufgrund seiner Beobachtungen zu verbessern. Wenn man dabei den lernenden Agenten, der zunächst noch überhaupt keine vernünftigen Strategien kennt, nur gegen optimal spielende Agenten antreten ließe, dann hätte dieser keine Chance dazuzulernen, da er in jedem Fall

verlieren würde. Um diesen Effekt zu vermeiden, benutzt man Agenten, die relativ simple Strategien spielen. Einige Beispiel für solche simplen Strategien sind:

- Der Agent räumt immer die erste Zeile aus, in der sich noch Streichhölzer befinden.
- Der Agent räumt immer die letzte Zeile aus, in der sich noch Streichhölzer befinden.
- Der Agent räumt die Zeile aus, in der sich die meisten Streichhölzer befinden.
- Der Agent räumt die Zeile aus, in der sich die wenigsten (aber > 0) Streichhölzer befinden.
- Alle vier bisher erwähnten Strategien können leicht modifiziert werden, indem aus der jeweiligen Zeile nur ein Streichholz entnommen wird.

Interessant sind vor allem Agenten, die gemischte Strategien spielen. Dabei kann sowohl zwischen den obigen simplen Strategien, als auch der optimalen Strategie abgewechselt werden. In so einem Umfeld trifft ein lernender Agent auf verschiedenste Gegenspieler und sammelt Erfahrungen. Genetische oder evolutionäre Algorithmen dienen in weiterer Folge dazu, aufgrund der in den einzelnen Partien erzielten Ergebnisse (Sieg oder Niederlage), die guten von den schlechten Regeln zu trennen. Solche Algorithmen umfassen im allgemeinen die Komponenten

- Selektion,
- Replikation,
- Rekombination und
- Mutation.

Interessant sind dabei einerseits das Tuning des Lernalgorithmus, andererseits aber auch die Fragestellung in welcher Umgebung (Gegenspieler) es dem lernenden Agenten am besten gelingt, erfolgreiche Strategien zu entwickeln.

Literatur

Banzhaf, W.; Nordin, P.; Keller, R. E.; Francone, F. D. (1998), *Genetic Programming, An Introduction, On the Automatic Evolution of Computer Programs and Ist Applications.* Morgan Kaufmann Publishers.

Binmore, K. (1992), *Fun and Games, A Text on Game Theory*. D.C. Heath and Company.

Friedberg, R. (1958), *A learning machine, part I.* IBM J. Research and Development.

Godlberg, D.E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company.

Korzybski, A.H.S. (1921), *Manhood of Humanity: The Science and Art of Human Engineering*.

Osborne, M.J. (1994), *A Course in Game Theory*. The MIT Press.