Technische Universität Wien

Vienna University of Technology

DIPLOMARBEIT

Algorithmische Spieltheorie: Effiziente Algorithmen zur Bestimmung spieltheoretischer Lösungen

Ausgeführt am Institut für

Wirtschaftsmathematik

der Technischen Universität Wien

unter der Anleitung von

Prof. Dr. Alexander MEHLMANN

durch

Markus Pecher
1120 Wien
Rotenmühlgasse 14

in Zusammenarbeit mit

Markus Riedl
1160 Wien
Roterdstraße 51

Datum

Unterschrift

Ehrenwörtliche Erklärung

Wir versichern, dass wir die eingereichte Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und uns auch sonst keiner unerlaubten Hilfsmittel bedient haben. Wir erklären weiters, dass wir diese Diplomarbeit bisher weder im In- noch im Ausland in irgendeiner Form als wissenschaftliche Arbeit vorgelegt haben.

Wien, September 2010

Markus Pecher und Markus Riedl

Danksagung

Wir möchten uns an dieser Stelle bei all jenen Personen bedanken, die uns bei der Erarbeitung dieser Diplomarbeit geholfen haben. Unser herzlicher Dank gilt Herrn Prof. Dr. Alexander Mehlmann, dem Betreuer unserer Arbeit. Bereits durch eine abwechslungsreiche Gestaltung der Lehrveranstaltungen hat er unser Interesse an der Spieltheorie geweckt, weshalb wir dieses Gebiet der Mathematik für unsere Diplomarbeit gewählt haben. Vielen Dank für die hilfreichen Anregungen und die angenehme Leitung.

Ein großes Dankeschön gilt unseren Eltern und Geschwistern, die, davon abgesehen, dass sie uns das Studium ermöglichten, auch immer großes Interesse an unserer Arbeit zeigten und uns soweit wie möglich unterstützten.

An dieser Stelle möchten wir auch unsere Korrekturleser Andreas Krail, Ernst Pecher, Mag. Doris Puchner und Dr. Michael Riedl dankend erwähnen, die sich sehr gewissenhaft unsere Arbeit durchgelesen haben.

Des Weiteren möchten wir uns bei Bernhard, Georg, Günther, Julia und Lukas, unseren ständigen Wegbegleitern durch das Studium, bedanken, die uns durch ihre unzähligen Hilfestellungen das Studium erleichtert haben.

Abschließend geht ein Dankeschön an all unsere Freunde und Kollegen, insbesondere an Judith und Julia, die dafür gesorgt haben, dass auch in schwierigen Zeiten des Studiums der Spaß nie zu kurz blieb.

Vielen Dank,
Markus und Markus

Abstract

In recent years computers became more and more important in the area of game theory. As a consequence a new field called algorithmic game theory arose, which deals with both the design of efficient algorithms for finding winning strategies and the analysis of the complexity of games. This diploma thesis gives a short introduction to this topic and presents various algorithms for finding Nash equilibria, evolutionary stable strategies and algorithms in cooperative games. A couple of examples illustrate their functioning and a comparison of the algorithms is carried out.

keywords: *game theory, algorithm, Nash equilibrium, evolutionary stable strategy*

Diese Diplomarbeit wurde von zwei Autoren verfasst, wobei

- **Kapitel 2**, welches das Auffinden von Nash-Gleichgewichten behandelt, von **Markus Pecher**,
- **Kapitel 3** und **Kapitel 4**, die sich mit evolutionär stabilen Strategien bzw. mit kooperativen Lösungen beschäftigen, von **Markus Riedl**
- und die Einleitung, die Konklusion sowie die Implementierung des Algorithmus von Govindan und Wilson mitsamt der numerischen Untersuchung gemeinsam erarbeitet wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis

"Pure mathematics is the world's best game. It is more absorbing than chess, more of a gamble than poker, and lasts longer than Monopoly. It's free. It can be played anywhere - Archimedes did it in a bathtub."

Richard J. Trudeau

Dots and Lines, 1978

1 Einleitung

Wie dieses Zitat auf humoristische Weise beschreibt, ist die Mathematik eine faszinierende Wissenschaft und gleichsam auch auf ihre eigene Weise ein Spiel. Es gibt Kinder, die Spaß daran haben, Zahlen zu kombinieren und Erwachsene die stundenlang versuchen, ein mathematisches Rätsel zu lösen. Der Grund für diese Hingabe ist die fesselnde Spannung, wenn die so heiß ersehnte Lösung zum Greifen nahe ist und die Genugtuung, wenn sie schlussendlich gefunden wurde. Dass das Spiel und die Mathematik eng miteinander verknüpft sind, zeigt wohl kein Teilgebiet der Mathematik besser auf, als die Spieltheorie.

Abbildung 1: Sun Tzu

Die ältesten Überlieferungen spieltheoretischer Überlegungen gehen auf den chinesischen General Sun Tzu zurück, der sich bereits vor 2500 Jahren detaillierte Gedanken zur Kriegsführung gemacht hat. Wie in der Übersetzung [1] von Giles nachzulesen ist, verblüfft Sun Tzu mit taktischen und strategischen Manövern, wo er alle möglichen Varianten "durchspielte" um sich auf jeden Eventualfall vorzubereiten. Napoleon bediente sich dieser Aufzeichnungen und gewann unzählige Schlachten, eher er in Waterloo gestoppt wurde, wo der Stabsoffizier General von Clausewitz diente, dem diese Schriften ebenfalls bekannt waren.

Abbildung 2: Der Computerpionier John von Neumann

Im 18. und 19. Jahrhundert beschäftigten sich mehrere Wissenschaftler, unter ihnen Waldegrave, Bernoulli, Bertrand, Cournot und von Stackelberg, mit diversen spieltheoretischen Fragestellungen. Da damals nur einzelne Probleme analysiert wurden, existierte die Spieltheorie als eigene Disziplin erst 1928, als der Ungar John von Neumann mit der Veröffentlichung einer Sammlung von seinen Arbeiten den Grundstein legte. Zusammen mit Oskar Morgenstern schrieb er 1944 das Buch "Theory of Games and Economic Behavior", welches das erste umfangreiche Buch war, welches sich dieser Thematik widmete. Unter anderem behandelten sie darin das Zwei-Personen-Nullsummenspiel, das zum Beispiel für die Beschreibung des Phänomens des Wettrüstens verwendet wurde. In dieser Zeit lag das Hauptaugenmerk auf der kooperativen Spieltheorie, wo

Spieler durch Bildung von Koalitionen versuchen, sich optimal zu verhalten. Der interessierte Leser findet weitere Informationen über die Entwicklung der Spieltheorie bis 1945 in dem Buch "The History of Game Theory" [2].

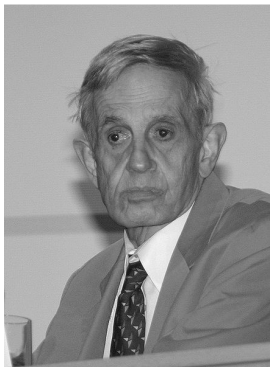

Abbildung 3: John Forbes Nash

Das erste allgemeine Lösungskonzept wurde vom amerikanischen Mathematiker und Nobelpreisträger John F. Nash entwickelt und 1951 in seiner Dissertation "Non-Cooperative Games" veröffentlicht. Das nach ihm benannte Nash-Gleichgewicht existiert im Gegensatz zu den damals bekannten Lösungen auch für mehr als zwei Spieler und für Nicht-Nullsummenspiele. Nash prägte mit seinen Arbeiten dieses Gebiet nachhaltig. Nicht nur deshalb, sondern auch aufgrund seiner paranoiden Schizophrenie sowie seiner extremen politischen Einstellung, wurde sein Leben von Regisseur Ron Howard in dem Oskar-preisgekrönten Meisterwerk "A Beautiful Mind" verfilmt. Abgesehen von seinen Auszeichnungen ist dies auch der bekannteste Film über einen Mathematiker.

Ab den 70er Jahren führte ein erhöhtes Interesse an der Spieltheorie zu einer Entwicklung von verschiedensten Teilgebieten. John Maynard Smith und George R. Price fanden mit der Evolutionären Spieltheorie eine Anwendung der Spieltheorie in der Biologie. Sie entwickelten den Begriff der evolutionär stabilen Strategie, der dazu dient, die natürliche Selektion mathematisch zu beschreiben. Der große Unterschied zur klassischen Spieltheorie liegt darin, dass man seine Strategie nicht immer wählen kann, sondern dass meist davon ausgegangen wird, dass sie erblich veranlagt ist.

Ein weiteres Gebiet, das durch die Weiterentwicklung des Computers und die Erfindung des Internets an Bedeutung gewann, ist die Algorithmische Spieltheorie. Sie stellt eine Verbindung zwischen Spieltheorie, Volkswirtschaftslehre und Informatik dar und befasst sich mit der effizienten Berechnung von verschiedenen Arten von Gleichgewichten und der Analyse der Komplexität von Spielen. 2007 erschien das umfangreiche Buch "Algorithmic Game Theory" [3], woran sich über 40 Wissenschaftler beteiligt haben. Es beinhaltet nicht nur wichtige Algorithmen für diverse Spiele, sondern auch praxisbezogene Anwendungen. Nachdem es sich um eine sehr junge Wissenschaft handelt, gibt es sicher noch einiges zu entdecken.

Da dieser Zweig der Spieltheorie top-aktuell ist, bot es sich an, das Thema "Algorithmische Spieltheorie: Effiziente Algorithmen zur Bestimmung spieltheoretischer Lösungen" zu wählen. Das Ziel der Arbeit ist eine Sammlung von Algorithmen für die verschiedenen Problemstellungen der Spieltheorie zu erstellen. Die Vorgehensweise hierfür besteht jeweils aus der Darlegung der Grundlagen, der Erarbeitung der Funk-

tionsweise des Algorithmus, mitsamt der Angabe des Pseudocodes und eines dem Verständnis dienenden Beispiels. Der wesentliche Inhalt gliedert sich in drei Kapitel.

Das erste Kapitel der Arbeit behandelt das Auffinden von Nash-Gleichgewichten. Nachdem die dafür notwendigen Grundlagen erläutert wurden, werden zunächst vier in der Literatur weniger häufig vertretene Algorithmen untersucht, ehe auf den weit verbreiteten Lemke-Howson-Algorithmus eingegangen und der ebenso bekannte Algorithmus von Govindan-Wilson skizziert wird. Nach einem kurzen Vergleich der Algorithmen wird noch der Spezialfall der degenerierten Spiele beleuchtet und abschließend extensive Spiele behandelt.

Im zweiten Kapitel werden Problemstellung und Lösungsmethoden der evolutionären Spieltheorie dargestellt. Bei der Suche nach evolutionär stabilen Strategien wird auch auf den Zusammenhang mit dem Nash-Gleichgewicht eingegangen. Nach einem einleitenden Beispiel werden der Algorithmus des Wiener Universitätsprofessors Immanuel Bomze und zwei Methoden von Zhi Lin beschrieben. Die evolutionäre Spieltheorie ist wohl für die in der Spieltheorie nicht so bewanderten Leser der interessanteste Teil der Arbeit, wie es auch schon von den Korrekturlesern mehrfach bestätigt wurde.

Der dritte Abschnitt befasst sich mit der kooperativen Spieltheorie. Dabei werden aus den vielen verschiedenen Lösungswegen der Kern und die Kreuz-Monotonie ausgewählt, welche näher erläutert werden. Auf andere Wege, wie zum Beispiel die Nash-Verhandlungslösung, wird hier nicht eingegangen. Die optimale Kostenverteilung wird bei submodularen Spielen und dem Facility-Location-Spiel durch das Primal-Dual-Verfahren berechnet.

Als letztes Kapitel bildet die Konklusion das Schlusswort.

2 Algorithmen zur Bestimmung von Nash-Gleichgewichten

Die algorithmische Bestimmung von Nash-Gleichgewichten ist einer der zentralen Punkte der Algorithmischen Spieltheorie. In den letzten Jahrzehnten wurden hierfür Methoden entwickelt, welche sich Hilfsmitteln aus den verschiedensten Gebieten der Mathematik bedienen. Dieses Kapitel stellt einige dieser Herangehensweisen vor.

2.1 Grundlagen

Zunächst werden Spiele in Normalform betrachtet, für welche Algorithmen zur Bestimmung von Nash-Gleichgewichten (siehe Definition 2.6) vorgestellt werden. In Kapitel 2.9 werden mehrere Algorithmen präsentiert, welche Spiele in extensiver Form behandeln.
Bei Spielen in Normalform agieren die Spieler gleichzeitig. Dies bedeutet, sie können nur Erwartungen bilden, welche Strategien die Kontrahenten verwenden werden. Jeder Spieler besitzt eine bekannte Auszahlungsmatrix, die für jede mögliche Kombination an Strategien einen fixen Nutzenwert beinhaltet. Das Ziel der Spieler ist es, durch rationales Handeln den eigenen Nutzen zu maximieren. Ein in der Literatur häufig vertretenes Beispiel ist jenes des Gefangenendilemmas.

Beispiel 2.1 (Gefangenendilemma)**.**

Zwei Verbrecher werden bei einer Tat gefasst und auf die Polizeistation gebracht. Die Höchststrafe für das Verbrechen beträgt sieben Jahre. Die derzeitige Beweislage würde aber nur ausreichen, um sie für drei Jahre hinter Gitter zu bringen. Aus diesem Grund werden sie getrennt verhört. Die Polizei bietet beiden den folgenden Deal an: Sollte jemand bereit sein, als Zeuge gegen den anderen auszusagen, so verringert sich seine Haftstrafe auf ein Jahr, sein Komplize erhält allerdings die Höchststrafe von sieben Jahren. Wenn jedoch beide Verbrecher als Zeuge gegen den anderen aussagen, werden beide für fünf Jahre eingesperrt. Diese Situation führt zu den Auszahlungsmatrizen A und B, welche hier übersichtlich in Bimatrix-Form dargestellt sind.

$A\backslash B$	aussagen	schweigen
aussagen	-5, -5	-1, -7
schweigen	-7, -1	-3, -3

Tabelle 1: Auszahlungen des Gefangenendilemmas

Der Zeilenspieler wählt eine Zeile i der Matrix A und der Spaltenspieler eine Spalte j

der Matrix B. Dies führt zu den Auszahlungen a_{ij} für den Zeilenspieler und b_{ij} für den Spaltenspieler. Das einzige Gleichgewicht in diesem Beispiel ist das Aussagen beider Spieler. Denn unabhängig davon, welche Strategie der damalige Komplize verwendet, erhält jeder Spieler einen höheren Nutzen, wenn er als Zeuge aussagt.

Der wesentliche Unterschied bei Spielen in extensiver Form liegt in der zeitlichen Abfolge des Agierens der Spieler. Anders als bei Spielen in Normalform beginnt hier ein Spieler mit der Wahl seiner Aktion und der nächste Spieler kann darauf reagieren. Spieler können mehrmals an der Reihe sein und es ist auch möglich, dass man nicht weiß, welche Aktionen von den anderen Spielern bisher gewählt wurden. In einigen Spielen tritt auch ein sogenannter Zufallsspieler auf, der seine Aktionen nach einer allgemein bekannten Wahrscheinlichkeitsverteilung auswählt und keine Auszahlung erhält. Ein Beispiel für einen solchen Zufallsspieler, oft Spieler 0 genannt, ist die Natur.

Eine Strategie in Spielen in extensiver Form bezeichnet einen vollständigen Handelsplan eines Spielers. Sie bestimmt also eine Aktion für jede mögliche Situation, in welcher der Spieler möglicherweise eine Entscheidung treffen muss.

Beispiel 2.2 (Vertrauensspiel)**.**

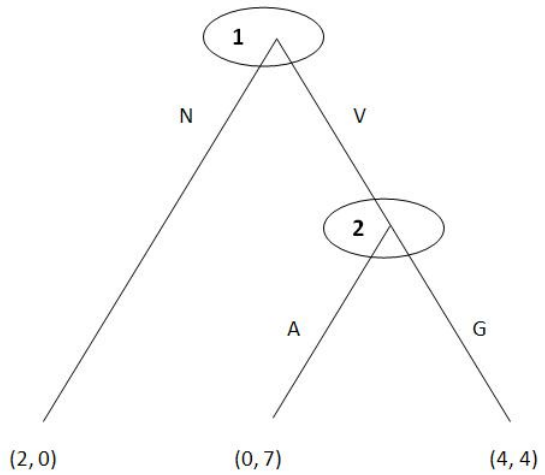

Abbildung 4: Baumdiagramm des Vertrauenspiels

Das Vertrauensspiel besteht aus einem Treugeber (Spieler 1) und einem Treuhänder (Spieler 2). Das Spiel beginnt mit einer Aktion von Spieler 1, der sich zwischen N (Nichtkooperation) und V (Vertrauensvoller Kooperation) entscheiden muss. Spieler

2 kommt nur dann zum Zug, falls ihm der Treugeber vertraut. In diesem Fall hat der Treuhänder die Wahl zwischen A (Ausbeutung von Spieler A) und G (Gerechte Aufteilung). Mithilfe eines Baumdiagramms, wie hier in Abbildung 4, lassen sich die zeitliche Abfolge und die Auszahlungen grafisch darstellen.

Die höchstmögliche Auszahlung des Treugebers wird mit der Strategiekombination (V, G) erreicht. Allerdings hat der Treugeber das Risiko, dass er ausgenutzt wird und am Ende gar nichts erhält. Dieses Risiko kann er vermeiden, indem er dem Treuhänder nicht vertraut und sich stattdessen mit einer geringeren Auszahlung zufrieden gibt.

Wie diese einleitenden Beispiele vermuten lassen, sind Spieltheoretiker seit jeher daran interessiert, Gleichgewichte in derartigen Konfliktsituationen zu bestimmen. Der zentrale Begriff hierbei ist das Nash-Gleichgewicht, welches, wie schon erwähnt, von John F. Nash Jr. in seiner Dissertation 1951 [4] entwickelt wurde. Um dieses zu erarbeiten, wird der Darstellung von Bernhard von Stengel in [5] gefolgt.

Im Folgenden wird ein Bimatrix-Spiel betrachtet. Wie im Beispiel 2.1 erwähnt, besitzen die Spieler die $m \times n$-Auszahlungsmatrizen A und B, welche den Nutzen für den Zeilen- und den Spaltenspieler angeben. Anstatt eine reine Strategie zu wählen, können die Spieler auch einen Zufallsmechanismus bestimmen, welcher dann eine reine Strategie auswählt. Diese sogenannte gemischte Strategie wird nun formal definiert.

Definition 2.3 (Gemischte Strategie)**.** *Eine gemischte Strategie x für einen Spieler ist ein stochastischer Vektor (nichtnegative Komponenten, Summe der Einträge ist 1), der angibt, mit welcher Wahrscheinlichkeit die jeweiligen Strategien gespielt werden.*

Ab jetzt bezeichnet der m-Vektor x die (gemischte) Strategie des Zeilenspielers (Spieler 1) und analog der n-Vektor y die des Spaltenspielers (Spieler 2).

Definition 2.4 (Träger)**.** *Der Träger einer gemischten Strategie x ist die Menge aller reinen Strategien i, für die $x_i > 0$ gilt.*

Weiters bezeichnet $M = \{1, \ldots, m\}$ die Menge aller reinen Strategien von Spieler 1 und $N = \{m + 1, \ldots, m + n\}$ die Menge aller reinen Strategien von Spieler 2.

Definition 2.5 (Beste Antwort)**.** *Die beste Antwort auf eine gemischte Strategie y von Spieler 2 ist eine gemischte Strategie x von Spieler 1, die die erwartete Auszahlung $x^T Ay$ maximiert.*

Definition 2.6 (Nash-Gleichgewicht)**.** *Ein Nash-Gleichgewicht ist ein Tupel (x, y) von gemischten Strategien, die gegenseitig beste Antworten sind.*

Satz 2.7. *Die gemischte Strategie x ist eine beste Antwort auf y genau dann, wenn für alle $i \in M$*

$$x_i > 0 \Rightarrow (Ay)_i = u = \max\{(Ay)_k \mid k \in M\} \tag{1}$$

gilt.

Beweis. $(Ay)_i$ ist die erwartete Auszahlung von Spieler 1 bei der Wahl der i-ten Zeile. Durch Umformen erhält man

$$x^T Ay = \sum_{i \in M} x_i (Ay)_i = \sum_{i \in M} x_i (u - (u - (Ay)_i)) = u - \sum_{i \in M} x_i (u - (Ay)_i).$$

Da $x_i \geq 0$ und $u - (Ay)_i \geq 0$ folgt $x^T Ay \leq u$. Weiters gilt $x^T Ay = 0$ genau dann, wenn $(Ay)_i = u$ aus $x_i > 0$ folgt. □

Dieser Satz ermöglicht eine schnelle Überprüfung eines möglichen Nash-Gleichgewichts, denn anstatt die unendliche Menge an gemischten Strategien zu untersuchen, genügt es nun, nur alle reinen Strategien zu betrachten. Einige der in diesem Kapitel vorgestellten Algorithmen werden sich dieses Satzes bedienen.

In der Spieltheorie unterscheidet man zwischen degenerierten und nichtdegenerierten Spielen.

Definition 2.8 (Degeneriert)**.** *Ein Zwei-Personen-Spiel wird degeneriert genannt, wenn es eine gemischte Strategie mit Trägergröße k gibt, die mehr als k beste Antworten besitzt.*

Andernfalls spricht man von einem nichtdegenerierten Spiel.

Eine andere Herangehensweise an das Auffinden eines Nash-Gleichgewichts ist die Polyedertheorie, weshalb eine kurze Einführung darüber notwendig ist.

Eine Linearkombination $\sum_{i=1}^{k} z_i \lambda_i$ von Punkten $z_1, \ldots, z_k$ heißt *affine Linearkombination*, wenn $\lambda_1, \ldots, \lambda_k$ reele Zahlen sind und $\sum_{i=1}^{k} \lambda_i = 1$. Ist $\lambda_i \geq 0$ für alle i, spricht man von einer *konvexen Linearkombination*. Eine Menge von Punkten ist *konvex*, wenn sie abgeschlossen bezüglich konvexer Linearkombinationen ist. Weiters nennt man sie *affin unabhängig*, wenn keiner der Punkte durch eine affine Linearkombination der anderen darstellbar ist. Eine konvexe Menge hat *Dimension* d genau dann, wenn sie $d + 1$ affin unabhängige Punkte besitzt.

Definition 2.9 (Polyeder)**.** *Ein Polyeder P im $\mathbb{R}^d$ ist die Menge $\{z \in \mathbb{R}^d \mid Cz \leq q\}$ bei gegebener Matrix C und Vektor q.*

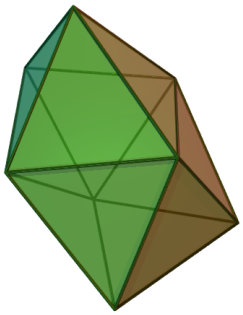

Abbildung 5: Ein Beispiel für ein dreidimensionales Polyeder

Ein Polyeder ist *volldimensional*, wenn es Dimension d hat. Ist es beschränkt, so spricht man von einem *Polytop*. Die Menge $\{z \in P \mid c^T z = q_0\}$ mit $c \in \mathbb{R}^d$ und $q_0 \in \mathbb{R}$ wird als *Face* bezeichnet, wenn $c^T z \leq q_0$ für alle z aus P gilt. Eine *Ecke* von P ist das eindeutige Element eines nulldimensionalen Face von P und eine *Kante* von P ist ein eindimensionales Face von P. Hat ein Face die Dimension $d-1$, so wird es als *Facet* bezeichnet. Ein d-dimensionales Polyeder P nennt man *einfach*, wenn kein Punkt zu mehr als d Facets von P gehört.

Weitere wichtige Begriffe werden in den später folgenden Kapiteln definiert, wie zum Beispiel die *Markierung* in Kapitel 2.3.

2.2 Der Träger-Test-Algorithmus

Wie schon weiter oben erwähnt, kann Satz 2.7 dazu verwendet werden, Nash-Gleichgewichte zu finden, indem alle möglichen Träger von gemischten Strategien überprüft werden. Alle reinen Strategien im Träger eines Spielers erzielen die gleiche, maximale Auszahlung für ihn. Nach dem Lösen der dadurch erhaltenen Gleichungen erhält man die Wahrscheinlichkeiten der gemischten Strategien des Gegners. Diese Idee findet man in [5].

Zur Erklärung der Vorgehensweise des Algorithmus dient folgende Aussage.

Satz 2.10. *Im Nash-Gleichgewicht (x, y) eines nichtdegenerierten Bimatrix-Spiels besitzen die gemischten Strategien x und y Träger derselben Größe.*

Beweis. Folgt direkt aus Satz 2.7. □

Algorithmus 2.11. *(Träger-Test-Algorithmus)*
Input: *Ein nichtdegeneriertes Bimatrix-Spiel.*
Output: *Alle Nash-Gleichgewichte.*
for $k = 1, \ldots, \min\{M, N\}$ ***do***
  for *jedes Tupel* (I, J) *an Teilmengen der Größe* k *von* M *bzw.* N ***do***
    Löse das Gleichungssystem $\sum_{i \in I} x_i b_{ij} = v \quad \forall j \in J, \quad \sum_{i \in I} x_i = 1$
    Löse das Gleichungssystem $\sum_{j \in J} a_{ij} y_j = u \quad \forall i \in I, \quad \sum_{j \in J} y_j = 1$
      if $x \geq \mathbf{0}$, $y \geq \mathbf{0}$ *und (1) gilt für* x *und* y ***then***
        (x, y) *ist ein Nash-Gleichgewicht.*

Das folgende Beispiel illustriert die Arbeitsweise des Träger-Test-Algorithmus.

Beispiel 2.12.

Gegeben sei das nichtdegenerierte 2-Personen-Spiel

$$A = \begin{pmatrix} 3 & 3 \\ 2 & 5 \\ 0 & 6 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 2 \\ 2 & 6 \\ 3 & 1 \end{pmatrix}$$

- $k = 1$
  - $(\{1\}, \{4\})$

$$\begin{aligned} 3 \cdot x_1 &= v \\ x_1 &= 1 \\ 3 \cdot y_1 &= u \\ y_1 &= 1 \end{aligned}$$

Man erhält $v = 3$ und $u = 3$.

Da $x \geq \mathbf{0}$, $y \geq \mathbf{0}$, $3 = u = \max\{3, 2, 0\}$ und $3 = v = \max\{3, 2\}$ gilt, ist $(x, y) = ((1, 0, 0), (1, 0))$ ein Nash-Gleichgewicht.

- $(\{2\}, \{4\})$

$$\begin{aligned} 2 \cdot x_2 &= v \\ x_2 &= 1 \\ 2 \cdot y_1 &= u \\ y_1 &= 1 \end{aligned}$$

Man erhält $v = 2$ und $u = 2$.

$x \geq \mathbf{0}$ und $y \geq \mathbf{0}$ sind zwar erfüllt, jedoch erhält man einen Widerspruch aus $2 = u = \max\{3, 2, 0\}$.

- $(\{3\}, \{4\})$

$$\begin{aligned} 3 \cdot x_3 &= v \\ x_3 &= 1 \\ 0 \cdot y_1 &= u \\ y_1 &= 1 \end{aligned}$$

Man erhält $v = 3$ und $u = 0$.

$x \geq \mathbf{0}$ und $y \geq \mathbf{0}$ sind zwar erfüllt, jedoch erhält man einen Widerspruch aus $0 = u = \max\{3, 2, 0\}$.

- $(\{1\}, \{5\})$

$$\begin{aligned} 2 \cdot x_1 &= v \\ x_1 &= 1 \\ 3 \cdot y_2 &= u \\ y_2 &= 1 \end{aligned}$$

Man erhält $v = 2$ und $u = 3$.

$x \geq \mathbf{0}$ und $y \geq \mathbf{0}$ sind zwar erfüllt, jedoch erhält man einen Widerspruch aus $3 = u = \max\{3, 5, 6\}$.

- $(\{2\}, \{5\})$

$$\begin{aligned} 6 \cdot x_2 &= v \\ x_2 &= 1 \\ 5 \cdot y_2 &= u \\ y_2 &= 1 \end{aligned}$$

Man erhält $v = 6$ und $u = 5$.

$x \geq \mathbf{0}$ und $y \geq \mathbf{0}$ sind zwar erfüllt, jedoch erhält man einen Widerspruch aus $5 = u = \max\{3, 5, 6\}$.

- $(\{3\}, \{5\})$

$$\begin{aligned} 1 \cdot x_3 &= v \\ x_3 &= 1 \\ 6 \cdot y_2 &= u \\ y_2 &= 1 \end{aligned}$$

Man erhält $v = 1$ und $u = 6$.

$x \geq \mathbf{0}$, $y \geq \mathbf{0}$ und $6 = u = \max\{3, 5, 6\}$ sind zwar erfüllt, jedoch erhält man einen Widerspruch aus $1 = v = \max\{3, 1\}$.

- $k = 2$
  - $(\{1, 2\}, \{4, 5\})$

$$\begin{aligned} 3 \cdot x_1 + 2 \cdot x_2 &= v \\ 2 \cdot x_1 + 6 \cdot x_2 &= v \\ x_1 + x_2 &= 1 \\ 3 \cdot y_1 + 3 \cdot y_2 &= u \\ 2 \cdot y_1 + 5 \cdot y_2 &= u \\ y_1 + y_2 &= 1 \end{aligned}$$

Man erhält $x_1 = \frac{4}{5}$, $x_2 = \frac{1}{5}$, $v = \frac{14}{5}$ und $y_1 = \frac{2}{3}$, $y_2 = \frac{1}{3}$, $u = 3$.

Da $x \geq \mathbf{0}$, $y \geq \mathbf{0}$ und die Bedingungen

$$\begin{aligned} (Ay)_1 &= 3 = u = \max\{3, 3, 2\} \\ (Ay)_2 &= 3 = u = \max\{3, 3, 2\} \\ (x^T B)_1 &= \tfrac{14}{5} = v = \max\{\tfrac{14}{5}, \tfrac{14}{5}\} \\ (x^T B)_2 &= \tfrac{14}{5} = v = \max\{\tfrac{14}{5}, \tfrac{14}{5}\} \end{aligned}$$

erfüllt sind, ist $(x, y) = ((\frac{4}{5}, \frac{1}{5}, 0), (\frac{2}{3}, \frac{1}{3}))$ ein Nash-Gleichgewicht.

  - $(\{1, 3\}, \{4, 5\})$

$$\begin{aligned} 3 \cdot x_1 + 3 \cdot x_3 &= v \\ 2 \cdot x_1 + 1 \cdot x_3 &= v \\ x_1 + x_3 &= 1 \\ 3 \cdot y_1 + 3 \cdot y_2 &= u \\ 0 \cdot y_1 + 6 \cdot y_2 &= u \\ y_1 + y_2 &= 1 \end{aligned}$$

Man erhält $x_1 = 2$, $x_3 = -1$, $v = 3$ und $y_1 = \frac{1}{2}$, $y_2 = \frac{1}{2}$, $u = 3$.

Die Bedingung $x \geq \mathbf{0}$ ist hier nicht erfüllt.

- $(\{2,3\},\{4,5\})$

$$\begin{array}{rcrcl} 2 \cdot x_2 & + & 3 \cdot x_3 & = & v \\ 6 \cdot x_2 & + & 1 \cdot x_3 & = & v \\ x_2 & + & x_3 & = & 1 \\ 2 \cdot y_1 & + & 5 \cdot y_2 & = & u \\ 0 \cdot y_1 & + & 6 \cdot y_2 & = & u \\ y_1 & + & y_2 & = & 1 \end{array}$$

Man erhält $x_2 = \frac{1}{3}$, $x_3 = \frac{2}{3}$, $v = \frac{8}{3}$ und $y_1 = \frac{1}{3}$, $y_2 = \frac{2}{3}$, $u = 4$.

Da $x \geq \mathbf{0}$, $y \geq \mathbf{0}$ und die Bedingungen

$$\begin{array}{rcccccl} (Ay)_2 & = & 4 & = & u & = & \max\{3,4,4\} \\ (Ay)_3 & = & 4 & = & u & = & \max\{3,4,4\} \\ (x^T B)_1 & = & \frac{8}{3} & = & v & = & \max\{\frac{8}{3},\frac{8}{3}\} \\ (x^T B)_2 & = & \frac{8}{3} & = & v & = & \max\{\frac{8}{3},\frac{8}{3}\} \end{array}$$

erfüllt sind, ist $(x,y) = ((0,\frac{1}{3},\frac{2}{3}),(\frac{1}{3},\frac{2}{3}))$ ein Nash-Gleichgewicht.

Der Algorithmus findet also die drei Nash-Gleichgewichte $(x,y) = ((1,0,0),(1,0))$, $(x,y) = ((\frac{4}{5},\frac{1}{5},0),(\frac{2}{3},\frac{1}{3}))$ und $(x,y) = ((0,\frac{1}{3},\frac{2}{3}),(\frac{1}{3},\frac{2}{3}))$. Da es ein nichtdegeneriertes Spiel ist und alle Träger-Profile behandelt wurden, gibt es keine weiteren Nash-Gleichgewichte.

2.3 Der Ecken-Test-Algorithmus

Wie auch beim Träger-Test-Algorithmus findet man eine Formulierung von Bernhard von Stengel in [5]. Um mögliche Träger von Gleichgewichtsstrategien zu finden, kann man auch *Beste-Antwort-Polytope* verwenden. Dazu muss zuerst ein *Beste-Antwort-Polyeder* definiert werden.

Definition 2.13 (Beste-Antwort-Polyeder)**.** *Das Beste-Antwort-Polyeder eines Spielers ist die Menge seiner gemischten Strategien zusammen mit der oberen Einhüllenden der erwarteten Auszahlungen für den anderen Spieler.*

Betrachtet man ein Bimatrix-Spiel mit Auszahlungsmatrizen $A \in \mathbb{R}^{M\times N}$ und $B \in \mathbb{R}^{M\times N}$ und gemischten Strategien x und y, so ergeben sich die folgenden Beste-Antwort-Polyeder für beide Spieler:

$$\begin{aligned} \bar{P} &= \left\{ (x,v) \in \mathbb{R}^M \times \mathbb{R} \mid x \geq \mathbf{0},\ B^T x \leq \mathbf{1}v,\ \sum_{i\in M} x_1 = 1 \right\}, \\ \bar{Q} &= \left\{ (y,u) \in \mathbb{R}^N \times \mathbb{R} \mid y \geq \mathbf{0},\ Ay \leq \mathbf{1}u,\ \sum_{j\in N} y_j = 1 \right\} \end{aligned}$$

Die Bedingungen, welche $\bar{P}$ und $\bar{Q}$ definieren, können vereinfacht werden, indem u und v eliminiert werden. Dafür dividiert man in $\bar{P}$ die Ungleichung $B^T x \leq \mathbf{1}v$ durch v. Daraus erhält man eine neue Ungleichung und die neue Variable $\frac{x_i}{v}$ wird wieder als x_i bezeichnet. Analog wird bei $\bar{Q}$ vorgegangen. Es wird also der erwartete Gewinn auf 1 normalisiert und die Bedingungen $\sum_{i\in M} x_1 = 1$ und $\sum_{j\in N} y_j = 1$ fallen weg. Folglich lassen sich die Beste-Antwort-Polytope als

$$\begin{aligned} P &= \left\{ x \in \mathbb{R}^M \mid x \geq \mathbf{0},\ B^T x \leq \mathbf{1} \right\}, \\ Q &= \left\{ y \in \mathbb{R}^N \mid Ay \leq \mathbf{1},\ y \geq \mathbf{0} \right\} \end{aligned}$$

anschreiben. Die Vektoren x und y müssen hier nicht zwingend stochastisch sein, jedoch kann man jeden dieser Vektoren ungleich dem Nullvektor zu einem stochastischen Vektor normalisieren:

$$nrml(x) := \frac{x}{\sum_i x_i}.$$

Wie sich für das Beispiel 2.12 auf Seite 16 das Beste-Antwort-Polyeder und das Beste-Antwort-Polytop für Spieler 2 grafisch veranschaulichen lassen zeigt Abbildung 6.

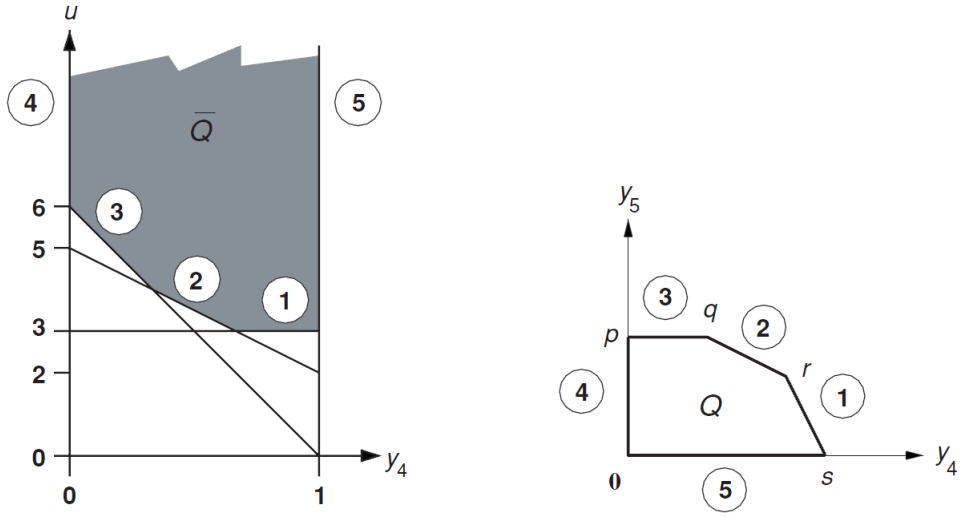

Abbildung 6: Beste-Antwort-Polyeder $\bar{Q}$ und Beste-Antwort-Polytop Q

Das linke Bild zeigt $\bar{Q}$ für $y_4 \in [0,1]$, wodurch $y_5 = 1 - y_4$ eindeutig bestimmt ist. Die Facets sind mit den eingekreisten Zahlen gekennzeichnet, wobei die Facets ①, ② und ③ die besten Antworten von Spieler 1 abhängig von y_4 sind. Die übrigen Facets ④ und ⑤ geben an, wenn die eigenen Strategien mit Wahrscheinlichkeit 0 oder 1 gespielt werden. Die rechte Grafik zeigt das dazugehörige Polytop Q nach Elimination von u.

Definition 2.14 (Markierung)**.** *Ein Punkt (y, u) von $\bar{Q}$ besitzt die Markierung $[k] \in M \cup N$, wenn die k-te Ungleichung in der Definition von $\bar{Q}$ bindend ist.*

Für $k = i \in M$ ist $\sum_{j \in N} a_{ij} y_j = u$ die i-te bindende Ungleichung und für $k = j \in N$ ist $y_j = 0$ die j-te bindende Ungleichung. Analog dazu hat ein Punkt y (x) eines Polytops Q (P) die Markierung $[k]$.

Definition 2.15 (vollständig markiert)**.** *Ein Tupel $(x, y) \in P \times Q$ ist vollständig markiert, wenn jede Markierung $[k] \in M \cup N$ entweder als Markierung von x oder y auftritt.*

Satz 2.16. *Ein Tupel (x, y) ist ein Nash-Gleichgewicht genau dann, wenn es vollständig markiert ist.*

Beweis. Angenommen (x, y) ist ein Nash-Gleichgewicht. Eine fehlende Markierung o.B.d.A. $[i] \in M$ würde bedeuten, dass eine reine Strategie existiert, für die weder $x_i \geq 0$ noch $(Ay)_i \leq 1$ bindend sind. Daher kann x_i keine beste Antwort sein (vgl. 1), was im Widerspruch zur Definition des Nash-Gleichgewichts steht.

Nun wird angenommen, dass (x, y) vollständig markiert ist. Die beste-Antwort-Bedingung 1 ist erfüllt, da $(Ay)_i = 1$ aus $x_i > 0$ und $(B^T x)_i = 1$ aus $y_i > 0$ folgt. Daher liegt ein Nash-Gleichgewicht vor. □

Algorithmus 2.17. *(Ecken-Test-Algorithmus)*
Input: *Ein nichtdegeneriertes Bimatrix-Spiel.*
Output: *Alle Nash-Gleichgewichte.*
for *jede Ecke* x *von* $P \backslash \mathbf{0}$ ***do***
  for *jede Ecke* y *von* $Q \backslash \mathbf{0}$ ***do***
    if (x, y) *ist vollständig markiert* ***then***
      $(nrml(x), nrml(y))$ *ist ein Nash-Gleichgewicht.*

Die Arbeitsweise des Ecken-Test-Algorithmus wird anhand von Beispiel 2.18 vorgeführt.

Beispiel 2.18.

Gegeben sei das nichtdegenerierte 2-Personen-Spiel

$$A = \begin{pmatrix} 3 & 3 \\ 2 & 5 \\ 0 & 6 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 2 \\ 2 & 6 \\ 3 & 1 \end{pmatrix}$$

Die Ecken wurden hierbei händisch berechnet. Bei der Implementierung kann man diese Aufgabe von einem Hilfsalgorithmus (z.B. LRS=lexicographic revers search) erledigen lassen.

- Ecke$(P) = a = (\frac{1}{3}, 0, 0)$
  - Ecke$(Q) = p = (0, \frac{1}{6})$
    Da $x_1 > 0$, muss $(Ay)_1 = 1$ erfüllt sein, damit (x, y) die Markierung [1] besitzt. $(Ay)_1 = \frac{1}{2} \neq 1 \Rightarrow$ kein Nash-Gleichgewicht.
  - Ecke$(Q) = q = (\frac{1}{12}, \frac{1}{6})$
    Da die Ecken unterschiedliche Trägergrößen besitzen, kann nach Satz 2.10 kein Nash-Gleichgewicht vorliegen.
  - Ecke$(Q) = r = (\frac{2}{9}, \frac{1}{9})$
    Da die Ecken unterschiedliche Trägergrößen besitzen, kann nach Satz 2.10 kein

Nash-Gleichgewicht vorliegen.

- Ecke$(Q) = s = (\frac{1}{3}, 0)$
  $x_1 > 0,\ y_4 > 0 \Rightarrow$ da $(Ay)_1 = 1$ und $(B^T x)_1 = 1$ erfüllt sind, besitzt (x, y) alle Markierungen. Somit ist $(nrml(x), nrml(y)) = ((1, 0, 0), (1, 0))$ ein Nash-Gleichgewicht.

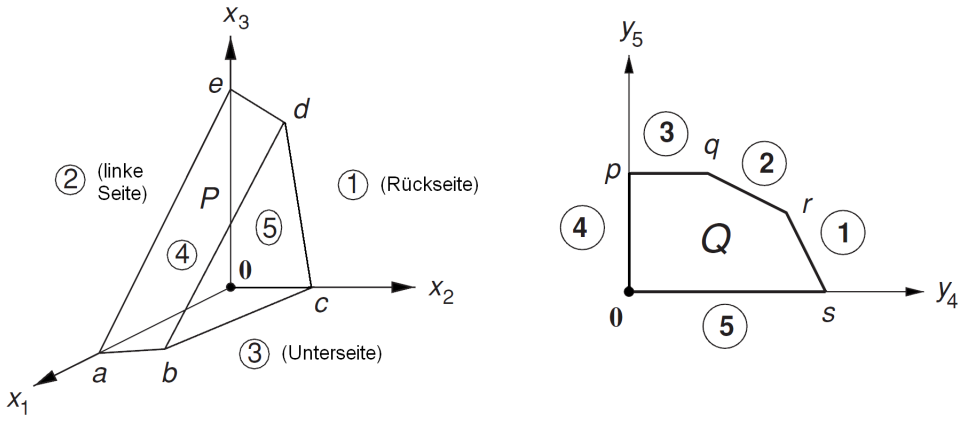

Abbildung 7: Die Beste-Antwort-Polytope P und Q.

- Ecke$(P) = b = (\frac{2}{7}, \frac{1}{14}, 0)$
  - Ecke$(Q) = p = (0, \frac{1}{6})$
    Da die Ecken unterschiedliche Trägergrößen besitzen, kann nach Satz 2.10 kein Nash-Gleichgewicht vorliegen.
  - Ecke$(Q) = q = (\frac{1}{12}, \frac{1}{6})$
    Da $x_1 > 0$, muss $(Ay)_1 = 1$ erfüllt sein, damit (x, y) die Markierung [1] besitzt. $(Ay)_1 = \frac{3}{4} \neq 1 \Rightarrow$ kein Nash-Gleichgewicht.
  - Ecke$(Q) = r = (\frac{2}{9}, \frac{1}{9})$
    $x_1 > 0,\ x_2 > 0,\ y_4 > 0,\ y_5 > 0 \Rightarrow$ da $(Ay)_1 = 1,\ (Ay)_2 = 1,\ (B^T x)_4 = 1$ und $(B^T x)_1 = 1$ erfüllt sind, besitzt (x, y) alle Markierungen. Somit ist der Punkt $(nrml(x), nrml(y)) = ((\frac{4}{5}, \frac{1}{5}, 0), (\frac{2}{3}, \frac{1}{3}))$ ein Nash-Gleichgewicht.
  - Ecke$(Q) = s = (\frac{1}{3}, 0)$
    Da die Ecken unterschiedliche Trägergrößen besitzen, kann nach Satz 2.10 kein Nash-Gleichgewicht vorliegen.

- Ecke$(P) = c = (0, \frac{1}{6}, 0)$
  - Ecke$(Q) = p = (0, \frac{1}{6})$
    Da $x_2 > 0$, muss $(Ay)_2 = 1$ erfüllt sein, damit (x, y) die Markierung [2] besitzt. $(Ay)_2 = \frac{5}{6} \neq 1 \Rightarrow$ kein Nash-Gleichgewicht.
  - Ecke$(Q) = q = (\frac{1}{12}, \frac{1}{6})$

Da die Ecken unterschiedliche Trägergrößen besitzen, kann nach Satz 2.10 kein Nash-Gleichgewicht vorliegen.

- ∘ Ecke$(Q) = r = (\frac{2}{9}, \frac{1}{9})$
  Da die Ecken unterschiedliche Trägergrößen besitzen, kann nach Satz 2.10 kein Nash-Gleichgewicht vorliegen.
- ∘ Ecke$(Q) = s = (\frac{1}{3}, 0)$
  Da $x_2 > 0$, muss $(Ay)_2 = 1$ erfüllt sein, damit (x, y) die Markierung [2] besitzt. $(Ay)_2 = \frac{2}{3} \neq 1 \Rightarrow$ kein Nash-Gleichgewicht.

• Ecke$(P) = d = (0, \frac{1}{8}, \frac{1}{4})$

- ∘ Ecke$(Q) = p = (0, \frac{1}{6})$
  Da die Ecken unterschiedliche Trägergrößen besitzen, kann nach Satz 2.10 kein Nash-Gleichgewicht vorliegen.
- ∘ Ecke$(Q) = q = (\frac{1}{12}, \frac{1}{6})$
  $x_2 > 0,\ x_3 > 0,\ y_4 > 0,\ y_5 > 0 \Rightarrow$ da $(Ay)_2 = 1,\ (Ay)_3 = 1,\ (B^T x)_4 = 1$ und $(B^T x)_1 = 1$ erfüllt sind, besitzt (x, y) alle Markierungen. Somit ist der Punkt $(nrml(x), nrml(y)) = ((0, \frac{1}{3}, \frac{2}{3}), (\frac{1}{3}, \frac{2}{3}))$ ein Nash-Gleichgewicht.
- ∘ Ecke$(Q) = r = (\frac{2}{9}, \frac{1}{9})$
  Da $x_3 > 0$, muss $(Ay)_3 = 1$ erfüllt sein, damit (x, y) die Markierung [3] besitzt. $(Ay)_3 = \frac{2}{3} \neq 1 \Rightarrow$ kein Nash-Gleichgewicht.
- ∘ Ecke$(Q) = s = (\frac{1}{3}, 0)$
  Da die Ecken unterschiedliche Trägergrößen besitzen, kann nach Satz 2.10 kein Nash-Gleichgewicht vorliegen.

• Ecke$(P) = e = (0, 0, \frac{1}{3})$

- ∘ Ecke$(Q) = p = (0, \frac{1}{6})$
  Da $y_5 > 0$, muss $(B^T x)_2 = 1$ erfüllt sein, damit (x, y) die Markierung [5] besitzt. $(B^T x)_5 = \frac{1}{3} \neq 1 \Rightarrow$ kein Nash-Gleichgewicht.
- ∘ Ecke$(Q) = q = (\frac{1}{12}, \frac{1}{6})$
  Da die Ecken unterschiedliche Trägergrößen besitzen, kann nach Satz 2.10 kein Nash-Gleichgewicht vorliegen.
- ∘ Ecke$(Q) = r = (\frac{2}{9}, \frac{1}{9})$
  Da die Ecken unterschiedliche Trägergrößen besitzen, kann nach Satz 2.10 kein Nash-Gleichgewicht vorliegen.
- ∘ Ecke$(Q) = s = (\frac{1}{3}, 0)$
  Da $x_3 > 0$, muss $(Ay)_3 = 1$ erfüllt sein, damit (x, y) die Markierung [3] besitzt. $(Ay)_3 = 0 \neq 1 \Rightarrow$ kein Nash-Gleichgewicht.

Wie schon der Träger-Test-Algorithmus findet der Ecken-Test-Algorithmus alle drei Nash-Gleichgewichte $(x, y) = ((1, 0, 0), (1, 0))$, $(x, y) = ((\frac{4}{5}, \frac{1}{5}, 0), (\frac{2}{3}, \frac{1}{3}))$ und $(x, y) = ((0, \frac{1}{3}, \frac{2}{3}), (\frac{1}{3}, \frac{2}{3}))$.

2.4 Zwei einfache Methoden zur Auffindung eines Nash-Gleichgewichts

Der nachfolgende Abschnitt folgt der Darstellung von Porter, Nudelman und Shoham. In [6] wird gezeigt, dass einfache Methoden existieren, um ein Nash-Gleichgewicht im 2-Personen- und n-Personen-Spiel zu finden. Diese zwei Verfahren werden im Folgenden als Algorithmus 1 und Algorithmus 2 bezeichnet. Wie der Leser im Kapitel 2.7 erfahren wird, funktioniert im 2-Personen-Spiel Algorithmus 1 deutlich besser als der Lemke-Howson-Algorithmus und im n-Personen-Spiel ist Algorithmus 2 dem Algorithmus von Govindan-Wilson überlegen.

Die zwei hier vorgestellten Algorithmen gehen den gesamten Raum der möglichen Träger durch und schränken diesen ein, indem nach strikt dominierten Strategien gesucht wird. Die grundlegende Idee hinter den beiden Methoden ist die abschließende Verwendung des Feasibility Program aus [7]. Falls zu gegebenen Trägern ein Nash-Gleichgewicht existiert, wird dieses ausgegeben. Die möglichen Träger werden aufsteigend der Größe nach sortiert. Da ein Großteil der in der Praxis auftretenden Spiele mindestens ein Nash-Gleichgewicht mit kleinen Trägern besitzt, erzielen die Algorithmen oft sehr schnell ein Ergebnis.

2.4.1 Notation

Betrachte ein endliches n-Personen-Spiel in Normalform.

- $N = \{1, \ldots, n\}$ ist die Menge der Spieler.
- $A_i = \{a_{i1}, \ldots, a_{im_i}\}$ ist die Menge der Strategien von Spieler i, wobei m_i die Anzahl an Strategien von Spieler i ist. Im Folgenden bezeichne a_i die Nummer einer gewissen Strategie a_{ij} von Spieler i und $a = (a_1, \ldots, a_n)$ ein Profil von Strategien. Weiters sei $a_{-i} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$ dieses Profil ohne der Strategie von Spieler i, sodass (a_i, a_{-i}) wieder ein komplettes Profil bildet.
- Aus der Menge $P_i = \{p_i : A_i \to [0,1] | \sum_{a_i \in A_i} p_i(a_i) = 1\}$ wählt jeder Spieler i eine gemischte Strategie. Der Träger einer gemischten Strategie ist laut Definition 2.4 der Vektor von allen Strategien $a_i \in A_i$, für die $p_i(a_i) > 0$ gilt.
- Die Funktion $u_i : A_1 \times \ldots \times A_n \to \mathbb{R}$ gibt den Nutzen für Spieler i bei gegebenem Strategieprofil $a = (a_1, \ldots, a_n)$ an. Da die Spieler gemischte Strategien verwenden können, berechnet $u_i(p) = \sum_{a \in A} p(a) u_i(a)$ den erwarteten Nutzen von $p = (p_1, \ldots, p_n)$, wobei $p(a) = \prod_{i \in N} p_i(a_i)$.
- Die Größe des Trägers jedes Spielers wird durch den ganzzahligen Vektor $x = (x_1, \ldots, x_n)$ beschrieben.

2.4.2 Das Feasibility Program

Wie oben erwähnt, berechnet das unten angeführte FEASIBILITY PROGRAM bei gegebenem Träger-Profil $S = (S_1, \ldots, S_n)$, falls möglich, ein Nash-Gleichgewicht, wobei $S_i \subseteq A_i$ für jeden Spieler i gilt. Der erwartete Nutzen von Spieler i in einem Gleichgewicht wird nun mit v_i bezeichnet.

Hilfsprogramm 2.19. FEASIBILITY PROGRAM

Input: *Ein Träger-Profil* $S = (S_1, \ldots, S_n)$

Output: *Ein Nash-Gleichgewicht* p*, falls ein Profil an Strategien* $p = (p_1, \ldots, p_n)$ *und ein Vektor* $v = (v_1, \ldots, v_n)$ *mit Werten* v_i *existiert, sodass die folgenden Bedingungen erfüllt sind:*

(i)	$\forall i \in N, a_i \in S_i:$	$\sum_{a_{-i} \in S_{-i}} p(a_{-i}) u_i(a_i, a_{-i}) = v_i$
(ii)	$\forall i \in N, a_i \notin S_i:$	$\sum_{a_{-i} \in S_{-i}} p(a_{-i}) u_i(a_i, a_{-i}) \leq v_i$
(iii)	$\forall i \in N:$	$\sum_{a_i \in S_i} p_i(a_i) = 1$
(iv)	$\forall i \in N, a_i \in S_i:$	$p_i(a_i) \geq 0$
(v)	$\forall i \in N, a_i \notin S_i:$	$p_i(a_i) = 0$

Die Bedingungen (i) und (ii) sorgen dafür, dass jeder Spieler indifferent zwischen allen Strategien in seinem Träger ist und keine Strategie außerhalb des Trägers bevorzugt. Daraus folgt, dass kein Spieler seinen erwarteten Nutzen erhöhen kann, indem er zu einer reinen Strategie wechselt. Diese Tatsache ist äquivalent dazu, dass p tatsächlich ein Nash-Gleichgewicht ist.

2.4.3 Funktionsweise von Algorithmus 1

Die Effizienz von Algorithmus 1 beruht auf drei wesentlichen Merkmalen. Zuerst werden jene Träger-Profile behandelt, bei welchen die Größe der Träger der zwei Spieler möglichst wenig voneinander abweicht. Ist diese Differenz bei mehreren Träger-Profilen gleich groß, so tritt das zweite Sortierkriterium in Kraft. Dieses bestimmt, dass jene Profile zuerst abgehandelt werden, deren Summe der Größe der Träger kleiner ist. Weshalb diese Reihenfolge zweckmäßig ist, geht auf die Arbeit von McLennen und Berg zurück, auf welche hier nicht näher eingegangen wird. Interessierte Leser werden auf [8] verwiesen.

Der dritte Grundgedanke ist die stufenweise Realisierung zweier Träger aus einem verkleinerten Suchraum. Diese Einschränkung wird mithilfe des Begriffs der bedingten Dominanz von Strategien erzielt.

Definition 2.20 (bedingte Dominanz)**.** *Sei ein Profil $R_{-i} \subseteq A_{-i}$ von Mengen von möglichen Strategien gegeben. Gilt*

$$\exists a_i' \in A_i \ \forall a_{-i} \in R_{-i} : \ u_i(a_i, a_{-i}) < u_i(a_i', a_{-i}),$$

so nennt man die Strategie $a_i \in A_i$ bedingt dominiert.

Nun kann der Pseudocode von Algorithmus 1 angegeben werden.

Algorithmus 2.21. *(Algorithmus 1)*
for all *Profile von Trägergrößen $x = (x_1, x_2)$, in aufsteigender Reihenfolge nach den Sortierkriterien $|x_1 - x_2|$ und $(x_1 + x_2)$ geordnet* ***do***
  for all *$S_1 \subseteq A_1$ mit $S_1 = |x_1|$* ***do***
  $A_2' := \{a_2 \in A_2 | a_2$ ist nicht dominiert, S_2 gegeben$\}$
    if *$\nexists a_1 \in S_1$ bedingt dominiert, A_2' gegeben* ***then***
      for all *$S_2 \subseteq A_2'$ mit $S_2 = |x_2|$* ***do***
        if *$\nexists a_1 \in S_1$ bedingt dominiert, S_2 gegeben* ***then***
          if *alle Bedingungen des* FEASIBILITY PROGRAM *für $S = (S_1, S_2)$ erfüllt sind* ***then***
            Gib das gefundene Nash-Gleichgewicht p aus!

Im Folgenden Abschnitt wird die Arbeitsweise des Algorithmus anhand zweier Beispiele präsentiert.

Beispiel 2.22.

Gegeben sei das 2-Personen-Spiel aus Tabelle 2.

$p1 \backslash p2$	4	5	6
1	0,4	2,5	3,2
2	4,0	3,4	2,3
3	3,3	1,1	5,2

Tabelle 2: Ein Bimatrix-Spiel mit jeweils drei Strategien

Die beiden Sortierkriterien ergeben folgende Reihenfolge der zu durchlaufenden Trägergrößen:

$$(1,1),\ (2,2),\ (3,3),\ (1,2),\ (2,1),\ (2,3),\ (3,2),\ (1,3),\ (3,1)$$

- $(1,1)$
  $S_1 = \{1\}$

$A'_2 = \{5\}$

Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

$S_2 = \{5\}$

Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

Das FEASIBILITY PROGRAM startet mit $S = (S_1, S_2) = (\{1\}, \{5\})$
Die zu erfüllenden Bedingungen lauten

$$\begin{array}{lcl} p(5)u_1(1,5) & = & v_1 \\ p(1)u_2(1,5) & = & v_2 \\ p(5)u_1(2,5) & \leq & v_1 \\ p(5)u_1(3,5) & \leq & v_1 \\ p(1)u_2(1,4) & \leq & v_2 \\ p(1)u_2(1,6) & \leq & v_2 \\ p(1) & = & 1 \\ p(5) & = & 1 \end{array}$$

Aus den ersten beiden Zeilen erhält man $v_1 = 2$ und $v_2 = 5$. Setzt man alle Werte in die dritte Zeile ein, führt dies zu dem Widerspruch $1 \cdot 3 \leq 2$.

$S_1 = \{2\}$

$A'_2 = \{5\}$

Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

$S_2 = \{5\}$

Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

Das FEASIBILITY PROGRAM startet mit $S = (S_1, S_2) = (\{2\}, \{5\})$
Die zu erfüllenden Bedingungen lauten

$$\begin{array}{lcl} p(5)u_1(2,5) & = & v_1 \\ p(2)u_2(2,5) & = & v_2 \\ p(5)u_1(1,5) & \leq & v_1 \\ p(5)u_1(3,5) & \leq & v_1 \\ p(2)u_2(2,4) & \leq & v_2 \\ p(2)u_2(2,6) & \leq & v_2 \\ p(2) & = & 1 \\ p(5) & = & 1 \end{array}$$

Aus den ersten beiden Zeilen erhält man $v_1 = 3$ und $v_2 = 4$. Da alle Ungleichungen erfüllt sind, liefert Algorithmus 1 das reine Nash-Gleichgewicht $p = (0, 1, 0, 0, 1, 0)$.

Beispiel 2.23.

Gegeben sei das 2-Personen-Spiel aus Tabelle 3.

$p1\backslash p2$	4	5	6
1	1,2	3,1	0,0
2	0,1	0,3	2,1
3	2,0	1,0	1,3

Tabelle 3: Bimatrix-Spiel mit drei Strategien

Die beiden Sortierkriterien ergeben folgende Reihenfolge der zu durchlaufenden Trägergrößen:

$$(1,1),\ (2,2),\ (3,3),\ (1,2),\ (2,1),\ (2,3),\ (3,2),\ (1,3),\ (3,1)$$

- $(1,1)$

  $S_1 = \{1\}$

  $A'_2 = \{4\}$

  Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

  $S_2 = \{4\}$

  Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

  Das FEASIBILITY PROGRAM startet mit $S = (S_1, S_2) = (\{1\}, \{4\})$

  Die zu erfüllenden Bedingungen lauten

$$\begin{array}{lcl} p(4)u_1(1,4) & = & v_1 \\ p(1)u_2(1,4) & = & v_2 \\ p(4)u_1(2,4) & \leq & v_1 \\ p(4)u_1(3,4) & \leq & v_1 \\ p(1)u_2(1,5) & \leq & v_2 \\ p(1)u_2(1,6) & \leq & v_2 \\ p(1) & = & 1 \\ p(4) & = & 1 \end{array}$$

  Aus den ersten beiden Zeilen erhält man $v_1 = 1$ und $v_2 = 1$. Setzt man alle Werte in die vierte Zeile ein, führt dies zu dem Widerspruch $1 \cdot 2 \leq 1$.

  $S_1 = \{2\}$

  $A'_2 = \{5\}$

  Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

  $S_2 = \{5\}$

  Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

Das FEASIBILITY PROGRAM startet mit $S = (S_1, S_2) = (\{2\}, \{5\})$
Die zu erfüllenden Bedingungen lauten

$$\begin{aligned} p(5)u_1(2,5) &= v_1 \\ p(2)u_2(2,5) &= v_2 \\ p(5)u_1(1,5) &\leq v_1 \\ p(5)u_1(3,5) &\leq v_1 \\ p(2)u_2(2,4) &\leq v_2 \\ p(2)u_2(2,6) &\leq v_2 \\ p(2) &= 1 \\ p(5) &= 1 \end{aligned}$$

Aus den ersten beiden Zeilen erhält man $v_1 = 0$ und $v_2 = 3$. Setzt man alle Werte in die dritte Zeile ein, führt dies zu dem Widerspruch $1 \cdot 3 \leq 0$.

$S_1 = \{3\}$
$A'_2 = \{6\}$
Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.
$S_2 = \{6\}$
Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.
Das FEASIBILITY PROGRAM startet mit $S = (S_1, S_2) = (\{3\}, \{6\})$
Die zu erfüllenden Bedingungen lauten

$$\begin{aligned} p(6)u_1(3,6) &= v_1 \\ p(6)u_2(3,6) &= v_2 \\ p(6)u_1(1,6) &\leq v_1 \\ p(6)u_1(2,6) &\leq v_1 \\ p(3)u_2(3,4) &\leq v_2 \\ p(3)u_2(3,5) &\leq v_2 \\ p(3) &= 1 \\ p(6) &= 1 \end{aligned}$$

Aus den ersten beiden Zeilen erhält man $v_1 = 1$ und $v_2 = 3$. Setzt man alle Werte in die vierte Zeile ein, führt dies zu dem Widerspruch $1 \cdot 2 \leq 1$.

- $(2, 2)$
  $S_1 = \{1, 2\}$
  $A'_2 = \{4, 5\}$
  Da die Strategie 2 bedingt dominiert ist, wird dieses Träger-Profil nicht weiter untersucht.
  $S_1 = \{1, 3\}$

$A_2' = \{4, 6\}$

Da die Strategie 1 bedingt dominiert ist, wird dieses Träger-Profil nicht weiter untersucht.

$S_1 = \{2, 3\}$

$A_2' = \{5, 6\}$

Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

$S_2 = \{5, 6\}$

Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

Das FEASIBILITY PROGRAM startet mit $S = (\{2, 3\}, \{5, 6\})$.

Die zu erfüllenden Bedingungen lauten

$$\begin{array}{lcll} p(5)u_1(2,5) & + & p(6)u_1(2,6) & = & v_1 \\ p(5)u_1(3,5) & + & p(6)u_1(3,6) & = & v_1 \\ p(2)u_2(2,5) & + & p(3)u_2(3,5) & = & v_2 \\ p(2)u_2(2,6) & + & p(3)u_2(3,6) & = & v_2 \\ p(2) & + & p(3) & = & 1 \\ p(5) & + & p(6) & = & 1 \\ p(5)u_1(1,5) & + & p(6)u_1(1,6) & \leq & v_1 \\ p(2)u_2(2,4) & + & p(3)u_2(3,4) & \leq & v_2 \end{array}$$

Die ersten sechs Zeilen bilden zwei Gleichungssysteme mit jeweils drei Gleichungen und drei Unbekannten. Man erhält $v_1 = 1$, $v_2 = \frac{9}{5}, p(2) = \frac{3}{5}, p(3) = \frac{2}{5}, p(5) = p(6) = \frac{1}{2}$.

Wenn man bei der Ungleichung $p(5)u_1(1,5) + p(6)u_1(1,6) \leq v_1$ die Werte einsetzt, ergibt sich mit $\frac{1}{2} \cdot 3 + \frac{1}{2} \cdot 0 \leq 1$ ein Widerspruch.

- $(3, 3)$

$S_1 = \{1, 2, 3\}$

$A_2' = \{4, 5, 6\}$

Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

$S_2 = \{4, 5, 6\}$

Da die **if**-Bedingung erfüllt ist, wird fortgesetzt.

Das FEASIBILITY PROGRAM startet mit $S = (\{1, 2, 3\}, \{4, 5, 6\})$.

Die zu erfüllenden Bedingungen lauten

$$
\begin{array}{lclclcl}
p(4)u_1(1,4) & + & p(5)u_1(1,5) & + & p(6)u_1(1,6) & = & v_1 \\
p(4)u_1(2,4) & + & p(5)u_1(2,5) & + & p(6)u_1(2,6) & = & v_1 \\
p(4)u_1(3,4) & + & p(5)u_1(3,5) & + & p(6)u_1(3,6) & = & v_1 \\
p(1)u_2(1,4) & + & p(2)u_2(2,4) & + & p(3)u_2(3,4) & = & v_2 \\
p(1)u_2(1,5) & + & p(2)u_2(2,5) & + & p(3)u_2(3,5) & = & v_2 \\
p(1)u_2(1,6) & + & p(2)u_2(2,6) & + & p(3)u_2(3,6) & = & v_2 \\
p(1) & + & p(2) & + & p(3) & = & 1 \\
p(4) & + & p(5) & + & p(6) & = & 1
\end{array}
$$

Man erhält zwei Gleichungssysteme mit jeweils vier Gleichungen und vier Unbekannten. Beim Auflösen dieser Systeme ergibt sich kein Widerspruch, sodass Algorithmus 1 das Nash-Gleichgewicht $p = (\frac{6}{13}, \frac{3}{13}, \frac{4}{13}, \frac{1}{9}, \frac{1}{3}, \frac{5}{9},)$ ausgibt.

Bei dem zweiten hier angeführten Beispiel hat Algorithmus 1 leider versagt, da dies kein Nash-Gleichgewicht mit kleineren Träger-Profilen besitzt. Bei dem ersten Beispiel hingegen wurde in wenigen Schritten ein Nash-Gleichgewicht gefunden.

2.4.4 Funktionsweise von Algorithmus 2

Im Gegensatz zu Algorithmus 1 kann Algorithmus 2 ein Nash-Gleichgewicht in einem n-Personen-Spiel für $n > 2$ berechnen. Algorithmus 2 ist keine reine Verallgemeinerung von Algorithmus 1, da die Sortierkriterien der Träger-Profile hier in umgekehrter Reihenfolge auftreten. Auf eine kleine Summe der Trägergrößen wird also mehr Wert gelegt als auf die Balance zwischen diesen Größen. Wie bei der Erläuterung der Funktionsweise von Algorithmus 1 findet sich eine Erklärung für dieses Vorgehen in [8]. Algorithmus 2 bedient sich der beiden Hilfsprogramme RECURSIVE-BACKTRACKING und ITERATED REMOVAL OF STRICTLY DOMINATED STRATEGIES (IRSDS). Bei gegebenem Vektor von Mengen von möglichen Träger-Profilen $D = (D_1, \ldots, D_n)$ streicht die Prozedur IRSDS in einer repeat-Schleife alle bedingt dominierten Strategien, solange entweder eine der Mengen D_i leer ist oder keine Strategien mehr gestrichen werden können.
Nachstehend werden die Pseudocodes von Algorithmus 2 und der beiden Hilfsprogramme angeführt.

Algorithmus 2.24. *(Algorithmus 2)*
for all *Profile von Trägergrößen $x = (x_1, \ldots, x_n)$, in aufsteigender Reihenfolge nach den Sortierkriterien $\sum_i x_i$ und $\max_{i,j} |x_i - x_j|$ geordnet* ***do***
$\forall i : S_i := NULL$
$\forall i : D_i := \{S_i \subseteq A_i : |S_i| = x_i\}$
if RECURSIVE-BACKTRACKING$(S, D, 1)$ *gibt ein Nash-Gleichgewicht p aus* ***then***
Gib das gefundene Nash-Gleichgewicht p aus!

Nun folgt der Pseudocode des Hilfsprogramms RECURSIVE-BACKTRACKING.

Hilfsprogramm 2.25. *(*RECURSIVE-BACKTRACKING)
Input: ein bestimmtes Träger-Profil $S = (S_1, \ldots, S_n)$
ein Profil an möglichen Trägern $D = (D_1, \ldots, D_n)$
ein Index i, der angibt, welcher Träger als nächstes initialisiert wird
Output: ein Nash-Gleichgewicht p oder *FEHLER*
if $i = n + 1$ **then**
if alle Bedingungen des FEASIBILITY PROGRAM *für $S = (S_1, \ldots, S_n)$ erfüllt sind* ***then***
gib das gefundene Nash-Gleichgewicht p aus!
else
gib FEHLER aus!
else
for all $d_i \in D_i$ ***do***

$S_i := d_i$
$D_i := D_i \backslash \{d_i\}$
if $IRSDS((\{S_1\}, \ldots, \{S_i\}, D_{i+1}, \ldots, D_n))$ *nicht abbricht* ***then***
if RECURSIVE-BACKTRACKING$(S, D, i+1)$ *gibt ein Nash-Gleichgewicht zurück* ***then***
gib das Nash-Gleichgewicht p zurück!
gib FEHLER zurück!

Abschließend folgt der Pseudocode des Hilfsprogramms ITERATED REMOVAL OF STRICTLY DOMINATED STRATEGIES.

Hilfsprogramm 2.26. *(*ITERATED REMOVAL OF STRICTLY DOMINATED STRATEGIES *(IRSDS))*

Input: *ein Profil an möglichen Trägern* $D = (D_1, \ldots, D_n)$
Output: *aktualisiertes Profil an möglichen Trägern oder FEHLER*
repeat
geändert:=false
for all $i \in N$ ***do***
for all $a_i \in d_i \in D_i$ ***do***
for all $a_i' \in A_i$ ***do***
if $\forall a_{-i} \in d_{-i} \in D_{-i}, u_i(a_i, a_{-i}) < u_i(a_i', a_{-i})$
then
$D_i := D_i \backslash \{d_i \in D_i : a_i \in d_i\}$
geändert:=true
if $D_i = \emptyset$ ***then***
gib FEHLER zurück!
until *geändert=false*
gib D zurück!

2.5 Der Lemke-Howson-Algorithmus

Der Lemke-Howson-Algorithmus ist einer der bekanntesten kombinatorischen Algorithmen zur Auffindung eines Nash-Gleichgewichts im Zwei-Personen-Spiel. Der Algorithmus wurde von C.E. Lemke und J.J.T. Howson 1964 in ihrer Arbeit "Equilibrium Points of Bimatrix Games" [9] veröffentlicht. Zusätzlich diente [10] als Hilfestellung bei der Ausarbeitung. Ähnlich wie beim Ecken-Test-Algorithmus verwendet man die beste Antwort-Polytope, auf welchen man einen Pfad von Ecke zu Ecke durchwandert, bis man schließlich zu einem vollständig markierten Tupel an Ecken gelangt. Dieser Pfad wird in der Literatur als *LH-Pfad* bezeichnet.

2.5.1 Funktionsweise

Der Algorithmus arbeitet mit den zwei Polytopen P und Q aus Kapitel 2.3

$$\begin{aligned} P &= \{x \in \mathbb{R}^M \mid x \geq \mathbf{0},\ B^T x \leq \mathbf{1}\}, \\ Q &= \{y \in \mathbb{R}^N \mid Ay \leq \mathbf{1},\ y \geq \mathbf{0}\} \end{aligned}$$

Wie oben müssen hier die Vektoren x und y nicht zwingend stochastisch sein, jedoch kann man jeden dieser Vektoren ungleich dem Nullvektor mit $nrml(x)$ und $nrml(y)$ zu einem stochastischen Vektor normalisieren.

Das Polytop P beinhaltet Informationen über die besten Antworten und die Träger wie folgt:

- Wenn $x_i \geq 0$ für ein $x \in P_1$ mit Gleichheit erfüllt ist, dann ist i nicht im Träger von x.
- Wenn $x^T \leq 1$ für ein $x \in P_1$ mit Gleichheit erfüllt ist, dann ist j die beste Antwort zu $nrml(x)$.

Analoges gilt für das Polytop Q.

Für den nächsten Satz ist folgende Definition erforderlich:

Definition 2.27 (bindendes Teilsystem)**.** *Die Menge aller bindenden Ungleichungen an der Stelle x bilden das bindende Teilsystem $\beta(x)$.*

Satz 2.28. *In einem einfachen d-dimensionalen Polytop gilt*

(a) $|\beta(\eta)| = d$ *für jeden Eckpunkt* η

(b) $\beta(\eta) \neq \beta(\eta')$ *für zwei verschiedene Eckpunkte* η *und* η'

(c) *jeder Eckpunkt grenzt an d Kanten. Insbesondere gibt es für jedes* $\vartheta \in \beta(\eta)$ *einen eindeutigen Nachbarn* η' *von* η *mit* $\beta(\eta') \cap \beta(\eta) = \beta(\eta) \backslash \{\vartheta\}$

Der Punkt (c), angewendet auf P oder Q, wird im Folgenden das Entfernen der Markierung k von η und der gleichzeitige Erhalt eines neuen Eckpunkts η' genannt, wobei k die zu der Ungleichung ϑ gehörende Markierung bezeichnet. Der Punkt (c) sagt außerdem aus, dass η' nur eine einzige Markierung enthält, die η nicht besitzt, nämlich die zu der eindeutigen Ungleichung $\beta(\eta') \backslash \beta(\eta)$ gehörende Markierung k'. Man spricht von der Aufnahme der Markierung k'.

Nun wird der Algorithmus angegeben und seine Korrektheit im nicht-degenerierten Fall überprüft. Sei nun immer x ein Eckpunkt von P und y ein Eckpunkt von Q. Weiters sei k_0 eine beliebige Markierung von x.

Algorithmus 2.29. *(Der Lemke-Howson-Algorithmus)*
$x :=$*Nullvektor der Länge* m
$y :=$*Nullvektor der Länge* n
$k := k_0$
loop
In P *entferne Markierung* k *von* x
x' *ist der neue Eckpunkt*
Markierung k' *wird aufgenommen*
$x = x'$
if $k' = k_0$ ***stop looping***
In Q *entferne Markierung* k' *von* y
y' *ist der neue Eckpunkt*
Markierung k'' *wird aufgenommen*
$y = y'$
if $k'' = k_0$ ***stop looping***
end loop
gib das Nash-Gleichgewicht $(nrml(x), nrml(y))$ *aus!*

Es ist leicht zu sehen wie der Algorithmus angepasst werden muss, damit die erste Markierung von y anstelle von x entfernt wird.

Satz 2.30. *Der Output des Lemke-Howson-Algorithmus ist ein Nash-Gleichgewicht.*

Beweis. In Folge bezeichne eine *Anordnung* ein Paar (x, y), wobei x ein Eckpunkt von P und y ein Eckpunkt von Q ist und x und y haben gemeinsam jede Markierung $M \cup N \backslash k_0$. Entscheidend ist nun, dass zu jedem Zeitpunkt des Algorithmus die Punkte x und y eine Anordnung bilden.

Zwei Anordnungen (x, y) und (x', y') heißen benachbart, wenn entweder

a) $x = x'$ und eine Kante von Q verbindet y mit y'

b) $y = y'$ und eine Kante von P verbindet x mit x'

Es sei angemerkt, dass jeder Schritt des Algorithmus von einer Anordnung zu einer benachbarten Anordnung führt. Es gibt zwei verschiedene Arten von Anordnungen:

- x und y besitzen gemeinsam alle Markierungen: Diese Anordnung ist benachbart zu genau einer anderen Anordnung, da entweder x oder y die Markierung k_0 besitzt und diese Markierung entfernt werden muss, obgleich x oder y diese besitzt.
- x und y teilen eine doppelte Markierung: Diese Anordnung ist benachbart zu genau zwei anderen Anordnungen, da man die doppelte Markierung sowohl von x als auch von y entfernen kann.

Betrachte nun einen Graph, dessen Punkte alle Anordnungen sind und dessen Kanten allen Paaren von benachbarten Anordnungen entsprechen. Wie oben beschrieben, besitzt jeder Punkt entweder alle Markierungen oder beinhaltet eine doppelte Markierung, sprich der Punkt hat Grad 1 oder Grad 2. Der Lemke-Howson-Algorithmus startet in der Anordnung $(0, 0)$. Diese Anordnung besitzt alle Markierungen und ist daher der Endpunkt eines Pfades. Der Algorithmus läuft entlang des Pfades bis er einen anderen Grad-1-Punkt findet. Dieser Punkt hat alle Markierungen und ist ungleich der Anfangskonfiguration $(0, 0)$. Ebenso kann der Algorithmus nicht an einer Anordnung der Form $(x, 0)$ oder $(0, y)$ abbrechen. Mithilfe von Satz 2.16 endet der Lemke-Howson-Algorithmus an einer Anordnung, die einem Nash-Gleichgewicht entspricht. □

2.5.2 Implementierung und Beispiele

Um ein Nash-Gleichgewicht mit dem Lemke-Howson-Algorithmus händisch zu berechnen, wird die Vorgehensweise in die folgenden vier Schritte unterteilt:

1. Vorbereitung
2. Initialisierung des Tableaus
3. Pivotisierung
4. Output

Um die Tableau-Schreibweise anwenden zu können, werden Schlupfvariablen eingeführt und die Bezeichnungen Basis- und Nicht-Basis-Variable verwendet. Das Aufnehmen einer Variable in die Basis bedeutet, dass die entsprechende Markierung entfernt wird, ebenso entspricht das Entfernen einer Variable aus der Basis dem Hinzufügen der Markierung.

1. Vorbereitung

Da das Eliminieren strikt dominierter Strategien keine Auswirkung auf das Nash-Gleichgewicht hat, können diese im Sinne einer Arbeitserleichterung beruhigt gestrichen werden. Beachte, dass die Dominanz auch von gemischten Strategien erzeugt werden kann. Anschließend wird eine hinreichend große Konstante zu allen Einträgen der Matrix addiert um zu gewährleisten, dass die geforderten Voraussetzungen erfüllt sind.

2. Initialisierung des Tableaus

Um das Spiel zu lösen, benötigt man zwei Tableaus, eines für jeden Spieler. Sei r_i die Schlupfvariable in der Ungleichung $A_i y \leq 1$ und s_j jene in der Ungleichung $x^T B_j \leq 1$. Somit erhält man das System

$$Ay + r = 1,\ B^T x + s = 1,$$

wobei x, y, r, s nichtnegativ sind.

Das folgende Spiel soll als Beispiel dienen:

$$A = \begin{pmatrix} 1 & 3 & 0 \\ 0 & 0 & 2 \\ 2 & 1 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 0 & 3 \end{pmatrix}$$

Alle Einträge sind nichtnegativ, keine Strategie wird strikt dominiert, es existieren keine reinen Nash-Gleichgewichte und es handelt sich um ein nicht-degeneriertes Spiel. Im Starttableau sind $\{r_i | i \in M\} \cup \{s_j < j \in N\}$ die Basisvariablen und daher formt man das obige System um zu $r = 1 - Ay$,

$$\begin{array}{llllll} r_1 = 1 & -y_4 & -3y_5 & & & [\text{A1}] \\ r_2 = 1 & & & -2y_6 & & [\text{A2}] \\ r_3 = 1 & -2y_4 & -y_5 & -y_6 & & [\text{A3}] \end{array}$$

und $s = 1 - B^T x$,

$$\begin{array}{llllll} s_4 = 1 & -2x_1 & -x_2 & & & [\text{B1}] \\ s_5 = 1 & -x_1 & -3x_2 & & & [\text{B2}] \\ s_6 = 1 & & -x_2 & -3x_3 & & [\text{B3}]. \end{array}$$

3. Pivotisierung

Zu Beginn wird willkürlich eine der x- oder y-Variablen in die Basis aufgenommen, was der beliebigen Wahl der Markierung k_0 entspricht, die entfernt wird. Bei der Wahl von x_1 ergibt die Min-Ratio-Regel (man betrachtet hierzu die Koeffizienten von x_1), dass s_4 die Basis verlassen muss. Nun wird in [B1] nach x_1 aufgelöst und die neue Gleichung in [B2] und [B3] eingesetzt.

$$\begin{array}{lllll} x_1 = & 1/2 & -1/2s_4 & -1/2x_2 & & [\text{B1}] \\ s_5 = & 1/2 & +1/2s_4 & -5/2x_2 & & [\text{B2}] \\ s_6 = & 1 & & -x_2 & -3x_3 & [\text{B3}] \end{array}$$

Beim Lemke-Howson-Algorithmus bestimmt die aus der Basis genommene Variable jene Variable, die als nächste in die Basis aufgenommen wird. Es gibt $m + n$ komplementäre Paare von Variablen, nämlich $\{r_i, x_i\}$ für $i \in M$ und $\{s_j, y_j\}$ für $j \in N$. Diese Paare bilden die $m + n$ komplementären Bedingungen

$$\begin{aligned} r_i x_i &= 0, \quad i \in M \\ s_j y_j &= 0, \quad j \in N, \end{aligned}$$

welche als Abbruchbedingungen dienen. Zu Beginn sind alle Bedingungen erfüllt, nach dem ersten Schritt jedoch ändert sich diese Tatsache. Die Pivotisierung wird solange durchgeführt, bis wieder alle komplementären Bedingungen erfüllt sind. Äquivalent ausgedrückt bricht der Algorithmus ab, wenn von jedem komplementären Paar an Variablen genau eine Basisvariable und die andere Nicht-Basisvariable ist. Bei dem gewählten Beispiel muss y_4 in die Basis aufgenommen werden, da die zu y_4 komplementäre Variable s_4 im letzten Schritt die Basis verlassen hat. Mit Hilfe der Min-Ratio-Regel erkennt man, dass nach der Aufnahme von y_4 die Variable r_3 aus der Basis gestrichen werden muss.

$$\begin{array}{lllll} r_1 = & 1/2 & +1/2r_3 & -5/2y_5 & +1/2y_6 & [\text{A1}] \\ r_2 = & 1 & & & -2y_6 & [\text{A2}] \\ y_4 = & 1/2 & -1/2r_3 & -1/2y_5 & -1/2y_6 & [\text{A3}] \end{array}$$

Wie oben erläutert, wird x_3 aufgrund der komplementären Bedingung in die Basis aufgenommen und die Min-Ratio-Regel besagt, dass s_6 aus der Basis genommen wird.

$$\begin{array}{lllll} x_1 = & 1/2 & -1/2s_4 & -1/2x_2 & & [\text{B1}] \\ s_5 = & 1/2 & +1/2s_4 & -5/2x_2 & & [\text{B2}] \\ x_3 = & 1/3 & & -1/3x_2 & -1/3s_6 & [\text{B3}] \end{array}$$

Im nächsten Schritt muss y_6 aufgenommen werden und die Min-Ratio-Regel lässt r_2 aus der Basis entfernen.

$$\begin{array}{lllll} r_1 = & 3/4 & +1/2r_3 & -5/2y_5 & -1/4r_2 \quad [A1] \\ y_6 = & 1/2 & & & -1/2r_2 \quad [A2] \\ y_4 = & 1/4 & -1/2r_3 & -1/2y_5 & +1/4r_2 \quad [A3] \end{array}$$

Nun wird x_2 aufgenommen und s_5 verlässt die Basis.

$$\begin{array}{lllll} x_1 = & 2/5 & -3/5s_4 & +1/5s_5 & \quad [B1] \\ x_2 = & 1/5 & +1/5s_4 & -2/5s_5 & \quad [B2] \\ x_3 = & 4/15 & -1/15s_4 & +2/15s_5 & -1/3s_6 \quad [B3] \end{array}$$

Nun wird y_5 aufgenommen und r_1 verlässt die Basis.

$$\begin{array}{lllll} y_5 = & 3/10 & +1/5r_3 & -2/5r_1 & -1/10r_2 \quad [A1] \\ y_6 = & 1/2 & & & -1/2r_2 \quad [A2] \\ y_4 = & 1/10 & -3/5r_3 & +1/5r_1 & +3/10r_2 \quad [A3] \end{array}$$

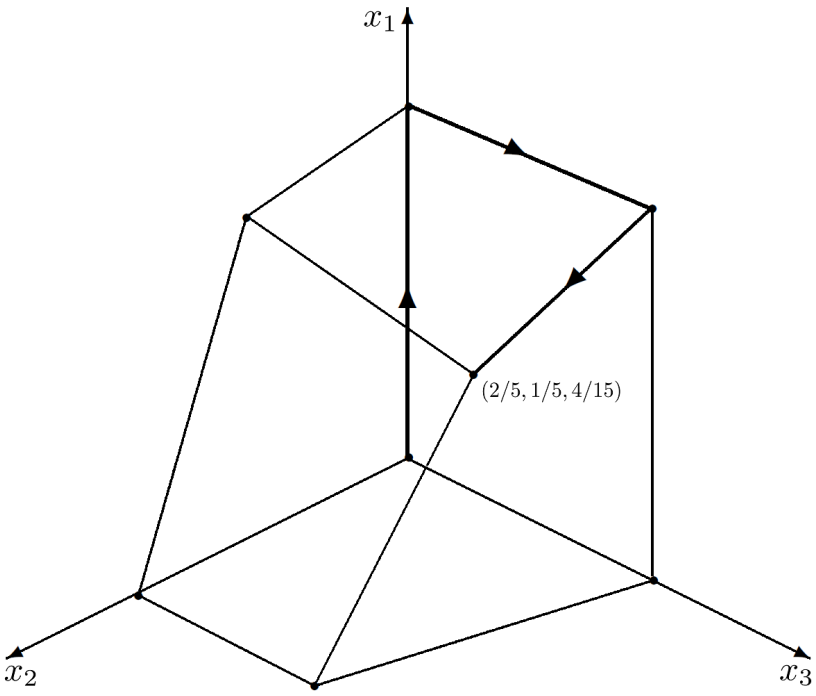

Abbildung 8: Der LH-Pfad startet im Ursprung und endet im Nash-Gleichgewicht

4. Output

Wie leicht zu sehen, sind jetzt beide komplementären Bedingungen erfüllt und der Algorithmus bricht somit hier ab. Man erhält die Werte für die Basisvariablen, indem alle Nicht-Basisvariablen auf Null gesetzt werden. Daher lautet das Ergebnis

$$\begin{aligned} r &= (0,0,0) \\ s &= (0,0,0) \\ x &= (2/5,1/5,4/15) \\ y &= (1/10,3/10,1/2). \end{aligned}$$

Der Output des Lemke-Howson-Algorithmus ist schließlich das Nash-Gleichgewicht

$$(nrml(x), nrml(y)) = (6/13,3/13,4/13),(1/9,3/9,5/9)).$$

Solche 2-Personen-Spiele können allerdings mehr als ein Nash-Gleichgewicht besitzen. Der Lemke-Howson-Algorithmus findet im Gegensatz zu den Algorithmen aus Kapitel 2.2 und Kapitel 2.3 eben nur eines der existierenden Gleichgewichte. Es ist jedoch möglich, durch Änderung der ersten fallen gelassenen Markierung, zu anderen Nash-Gleichgewichten zu gelangen. Manche Gleichgewichte bleiben aber auch mit dieser Abänderung dem Lemke-Howson-Algorithmus verborgen.

2.6 Algorithmus von Govindan und Wilson

S. Govindan und R. Wilson wählten einen interessanten Zugang, um Nash-Gleichgewichte zu berechnen. Sie bedienten sich zweier Algorithmen, die durch Abschreiten eines Pfades ein Gleichungssystem lösen, nämlich der Global Newton Method von Smale (1976) und der Homotopy Method von Eaves (1972, 1984). Diese Methoden werden verwendet, um Nash-Gleichgewichte endlicher Spiele in Normalform zu berechnen. Wie in [11] beschrieben, suchen beide Methoden entlang desselben Pfades. Diese erstaunliche Tatsache existiert aufgrund einer fundamentalen topologischen Eigenschaft des Nash-Gleichgewichts. Das Strukturtheorem von Kohlberg und Mertens (1986) zeigt, dass der Graph, welcher einem Nash-Gleichgewicht entspricht, homöomorph zu dem Raum der Spiele ist. Dieser Homöomorphismus ermöglicht es, dass jede Methode ausschließlich in einem Unterraum des Raums der Spiele durchgeführt wird, dessen Dimension gleich groß wie die Anzahl an reinen Strategien ist. Danach wird der Homöomorphismus angewendet um die gemischten Strategien im Gleichgewicht zu erhalten.

Die Homotopy Method, welche in [12] und [13] ausführlich beschrieben wird, deformiert ein gegebenes Gleichungssystem in ein leicht zu lösendes System. Nachdem dieses gelöst wurde, wird die Deformation rückgängig gemacht und die Lösungen der zugehörigen Systeme entlang des Weges dieser Rücktransformation aufgezeichnet. Somit erhält man die Lösung des ursprünglichen Gleichungssystems als die Lösung am Ende dieses Weges. Diese Methode kann man sich auch im Falle des Nash-Gleichgewichts zu Nutze machen. Das zu lösende Gleichungssystem besteht nun aus jenen Gleichungen, die das Nash-Gleichgewicht beschreiben. Dieses Spiel wird in ein anderes Spiel deformiert, dessen Gleichgewicht leicht berechnet werden kann, und die Rücktransformation liefert das gesuchte Nash-Gleichgewicht.

Ein Spiel wird im Folgenden häufig als Punkt im euklidischen Raum repräsentiert. Um die Vorteile der spieltheoretischen Struktur auszunützen, wird das Strukturtheorem angewendet. Dieses besagt, dass über jeden generischen Strahl, der vom gegebenen Spiel ausgeht, der dem Gleichgewicht entsprechende Graph eine eindimensionale Mannigfaltigkeit ist. Außerdem existiert in hinreichendem Abstand zum gegebenen Spiel ein anderes Spiel mit demselben, eindeutigen Gleichgewicht. Startet man daher von einem in hinreichendem Abstand gelegenen Spiel entlang eines generischen Strahls, kann man den Weg von dem gegebenen Spiel abgehen und findet ein Gleichgewicht am Ende des Weges, wenn man die eindimensionale Mannigfaltigkeit der Gleichgewichte entlang des Weges aufzeichnet. Das Strukturtheorem liefert eine weitere interessante Aussage. Der Homöomorphismus zeigt, dass es ausreicht, den Teilraum der deformierten Spiele mit derselben Dimension wie die des Raums der gemischten Strategien zu betrachten. Da nun die Dimensionen des Raumes der deformierten Spiele und des

Raumes der Gleichgewichte gleich sind, kann die Global Newton Method für Berechnungen verwendet werden.

Zusammenfassend kann man sagen, dass dem homöomorphen Bild der eindimensionalen Mannigfaltigkeit entlang eines generischen Strahls mithilfe der Global Newton Method gefolgt wird. Mit anderen Worten: Man erhält eine Prozedur, die analog zum Homotopy Principle die Rücktransformation verfolgt und aufzeichnet.

Da die genaue Ausführung des Strukturtheorems inklusive aller Auswirkungen für die Konstruktion der Global Newton Method bei weitem zu umfangreich und nicht Sinn dieser Arbeit ist, wird der interessierte Leser auf [11] verwiesen. Anstatt sich also in endloser Topologie zu verlieren, wird nun die Notation angegeben und ein Pseudocode des Algorithmus präsentiert.

2.6.1 Notation

Im Folgenden bezeichne

N	eine endliche Menge an Spielern
S_n	die endliche Menge reiner Strategien von Spieler n
S_{-n}	die Menge $\prod_{i \neq n} S_i$ reiner Strategieprofile der Gegner von Spieler n
S	die Menge $\prod_{n \in N} S_n$
m	die Zahl $\sum_n \lvert S_n \rvert$
Σ_n	die Menge gemischter Strategien von S_n
Σ	die Menge $\prod_{n \in N} \Sigma_n$, die im Euklidischen Raum mit Dimension m liegt
E	der Graph, der dem Nash-Gleichgewicht entspricht, also $E = \{(G, \sigma) \in \Gamma \times \Sigma \mid \sigma$ ist ein Gleichgewicht des Spiels $G\}$

2.6.2 Implementierung

Govindan und Wilson implementierten ihren Algorithmus mithilfe der APL-Programmiersprache und veröffentlichten numerische Ergebnisse in [11]. Im Folgenden wird der Pseudocode ihrer Global Newton Method zur Berechnung von Nash-Gleichgewichten angegeben:

Algorithmus 2.31. *(Algorithmus von Govindan und Wilson)*

1. *Berechne für jeden Spieler $n \in N$ die eindeutige, beste Strategie $s_n^* = Arg \max_{s \in S_n} g_s$, sodass die Kombination $s^* = (s_n^*)_{n \in N}$ reiner Strategien das eindeutige Gleichgewicht ist, falls λ hinreichend groß ist ($\lambda > \lambda^*$). Anschließend wird λ^* berechnet, indem jene Ungleichungen untersucht werden, welche dafür sorgen, dass s^* ein Gleichgewicht ist. Sei $\sigma^* \in \Sigma \subset \mathbb{R}^m$ jene Ecke, welche der*

gemischten Strategie mit Träger s^* entspricht. Beginne mit $t = 0$, $\lambda(0) = \lambda^*$ und $z(0) = z^* \in \mathbb{R}^m$, wobei

$$z^* = H(G, \lambda(0)g, \sigma^*) = \sigma^* + G(\sigma^*) + \lambda(0)g,$$

und mit dem Träger $B^* = \sigma^*$.

2. *Rufe die Funktion **Retract** auf, um den m-Vektor, welcher die gemischte Strategie $\sigma = r(z(t))$ ist, und den Träger $B = B(z(t))$ von σ in $z(t)$ zu berechnen. Mithilfe von B wird nun die $m \times m$-Matrix R berechnet, welche die Jacobi-Matrix Dr der Retraktion r in $z(t)$ ist.*

3. *Rufe die Funktion **Payoff** auf, um die $m \times m$-Matrix DG zu berechnen, welche die Jacobi-Matrix der Auszahlung der gemischten Strategie σ ist. Führe nun die Funktion **Adjoint** durch. Diese berechnet die adjungierte $m \times m$-Matrix $J = Adj(D\psi)$ und die skalare Determinante $|D\psi|$ der Jacobi-Matrix $D\psi = I - (I + DG) \cdot R$ in $z(t)$, wobei I die Einheitsmatrix bezeichnet.*

4. *Sei $\dot{z}(t) = -J \cdot g$ und $\dot{\lambda}(t) = -|D\psi|$. Dies sind die zwei Schlüsselgleichungen der Global Newton Method.*

5. *Berechne den m-Vektor $v(t) = DG \cdot \frac{\sigma}{|N|-1} + \lambda(t)g$ der erwarteten Auszahlungen der reinen Strategien und dessen Zeit-Derivative $\dot{z}(t) = DG \cdot R \cdot \dot{z}(t) + \dot{\lambda}(t)g$. Hierbei werden allerdings nur die reinen Strategien des Trägers verwendet.*

6. *Unter der Verwendung von $(v(t), z(t))$ und der Annahme, dass $(\dot{v}, \dot{z})$ konstant sind, extrapoliere zur nächten Grenze, wo eine reine Strategie $\hat{s}$ in den Träger aufgenommen wird oder diesen verlässt. In anderen Worten: Das maximale Zeitintervall $\Delta > 0$ wird berechnet, für welches $z(t) + \Delta\dot{z} \geq v(t) + \Delta\dot{v}$ gilt. Wenn $\Delta = \infty$, beende mit dem Hinweis, dass alle Gleichgewichte auf dem Pfad gefunden wurden.*

7. *Falls $|N| = 2$, ist die Annahme der Konstanz in Schritt 6 richtig und daher kann die volle Schrittgröße Δ verwendet werden. Wenn ein Kriterium der Nähe zur nächsten Grenze eintritt, so wird ebenfalls die volle Schrittgröße Δ verwendet, wodurch im nächsten Schritt zu dieser Grenze gesprungen wird. Ein mögliches Kriterium für die Nähe zu einer Grenze findet sich in [11].*

8. *Wenn $\lambda(t) + \delta\dot{\lambda}(t) \leq 0$, so beachte, dass ein Gleichgewicht in $\lambda(t + d\hat{e}lta) = 0$ existiert, wobei $d\hat{e}lta = -\frac{\lambda(t)}{\dot{\lambda}(t)}$. Springe zu Schritt 12, um diesen Fund aufzuzeichnen. Falls $\lambda(t) + \delta\dot{\lambda}(t) > 0$, gehe zu $z(t') = z(t) + \delta\dot{z}(t)$ und $\lambda(t') = \lambda(t) + \delta\dot{\lambda}(t)$ mit $t' = t + \delta$. Wenn $\lambda(t') < \lambda_* \ll 0$, beende mit dem Hinweis, dass die Suche nach zusätzlichen Gleichgewichten, welche über den Strahl g zugänglich sind, beendet ist.*

9. *Setze t auf den Wert von t' zurück. Wenn $\delta = \Delta$, so wurde eine Grenze erreicht. Springe zu Schritt 10. Wenn eine festgesetzte Anzahl an Iterationen durchgeführt wurde, können sich mehrere Fehler ansammeln. Daher wird die Funktion* ***LNM*** *aufgerufen, um die genaue Position auf dem Strahl wieder herzustellen. Dies geschieht durch das Lösen der Gleichung $z - r(z) - G(r(z)) + \lambda(t)g = 0$ für z, welches dann $z(t)$ ersetzt. Ansonsten, gehe zu Schritt 3 zurück, wo DG vor dem nachfolgenden Schritt neu berechnet wird.*

10. *Nachdem eine Grenze erreicht wurde, zeichne die Änderung im Träger B auf. Genauer gesagt, die im Schritt 6 gefundene reine Strategie $\hat{s}$ betritt oder verlässt die Basis.*

11. *Um sicher zu gehen, dass $z(t)$ einem Gleichgewicht des Spiels $(G, \lambda(t)g)$ entspricht, ändere den Strahl zu $g = (z(t) - r(z(t)) - \frac{G(r(z(t)))}{\lambda(t)}$. Setze mit Schritt 2 fort.*

12. *Nachdem ein ungefähres Gleichgewicht gefunden wurde, rufe* ***LNM*** *auf um die numerische Genauigkeit zu verbessern. Speichere das gefundene Gleichgewicht mitsamt seinem Index. Gehe zu Schritt 2 zurück, um den Pfad zum nächsten Gleichgewicht mit entgegengesetztem Index fortzusetzen.*

2.6.3 Numerische Untersuchung

Die von Professorin Daphne Koller gegründete Forschungsgruppe "DAGS" beschäftigte sich näher mit dem Algorithmus von Govindan und Wilson. Zusammen mit Ben Blum und Christian Shelton entwickelte Koller eine Implementierung in C++ und veröffentlichte den Code unter dem Namen "GameTracer" auf der Homepage ihrer Forschungsgruppe. Dieser befindet sich auch im Appendix dieser Arbeit.

Genau diese Implementierung verwendeten wir für die numerische Untersuchung des Algorithmus auf einem Intel Core Duo Prozessor mit 1.66 GHz. In Tabelle 4 sind die Laufzeiten des Algorithmus für zufällig generierte Spiele für zwei bis sechs Spieler mit jeweils zwei bis 15 Strategien in Sekunden angegeben. Es wurden immer zehn Spiele für jede Kombination von Anzahl an Spielern und Strategien getestet und das arithmetische Mittel gebildet, um sie vor Ausreißern zu schützen und die Werte aussagekräftiger zu machen.

Wie man in Tabelle 4 gut erkennen kann, geht die Berechnung der Gleichgewichte im Zwei-Personen-Spiel sehr rasch. Durch einen zusätzlichen Test wurde festgestellt, dass erst ab etwa 32 bis 34 Strategien eine Laufzeit im Minutenbereich benötigt wird. Der Grund für diese schnelle Berechnung liegt darin, dass der Algorithmus linear von

$\|N\|$ m_n	2	3	5	7	10	15
2	0,00	0,00	0,00	0,01	0,02	0,08
3	0,01	0,07	0,43	3,84	12,93	74,31
4	0,05	0,43	3,22	12,35	92,51	1050,56
5	0,10	1,42	8,95	95,39	776,23	
6	0,36	3,26	109,69	1778,43		
7	0,81	14,42	795,56			
8	1,24	60,34				
9	4,71	175,13				

Tabelle 4: Laufzeiten in Sekunden

einer Grenze zu einer anderen springt, indem er genau dem homöomorphen Bild des Pfades des Lemke-Howson-Algorithmus folgt.

2.7 Vergleich der Algorithmen

Nachdem sechs verschiedene Algorithmen um Nash-Gleichgewichte zu berechnen vorgestellt wurden, werden Eigenschaften von diesen nun kurz miteinander verglichen. Tabelle 5 liefert einen Überblick, für wie viele Spieler die Algorithmen verwendet werden können und wie viele Gleichgewichte jeweils gefunden werden.

Bezeichnung des Algorithmus	Spieler	Gleichgewichte
Träger-Test-Algorithmus	2	alle
Ecken-Test-Algorithmus	2	alle
Algorithmus 1	2	eines
Algorithmus 2	n	eines
Lemke-Howson-Algorithmus	2	eines
Algorithmus von Govindan und Wilson	n	mind. eines

Tabelle 5: Vergleich der Algorithmen

Sowohl der Träger-Test- als auch der Ecken-Test-Algorithmus finden alle Nash-Gleichgewichte, jedoch können sie nur bei Zwei-Personen-Spielen angewendet werden. Der Ecken-Test-Algorithmus ist in der Regel besser als der Träger-Test-Algorithmus, da weniger Ecken als Träger existieren. Beiden Methoden kann natürlich eine Abbruchbedingung hinzugefügt werden, sodass diese nach dem ersten gefundenen Nash-Gleichgewicht abbrechen. Durch diese Modifikation können sie nun mit dem Lemke-Howson-Algorithmus verglichen werden, der ebenso nur ein Nash-Gleichgewicht ausgibt. Meist erzielt der Lemke-Howson-Algorithmus schneller ein Ergebnis als der Ecken-Test-Algorithmus (und somit auch als der Träger-Test-Algorithmus). Algorithmus 1 findet ebenso ein Nash-Gleichgewicht bei gegebenem Zwei-Personen-Spiel. Da es eine recht simple Vorgehensweise ist, benötigt diese Methode bei allgemeinen Spielen oft sehr lange, weshalb sie dem Lemke-Howson-Algorithmus unterlegen ist. Die Autoren Porter, Nudelman und Shoham fanden allerdings heraus, dass bei den meisten in der Praxis auftretenden Beispielen ein Nash-Gleichgewicht mit kleinen Trägern existiert, wodurch Algorithmus 1 dieses außergewöhnlich schnell findet. Für Algorithmus 2, der ebenso wie der Algorithmus von Govindan und Wilson für n Spieler verwendet werden kann, gilt dieselbe Tatsache.

Da der Lemke-Howson-Algorithmus einen Pfad durchläuft, bis er auf ein Gleichgewicht stößt, findet er nur eines der existierenden Gleichgewichte. Es ist jedoch möglich, durch Änderung der ersten fallen gelassenen Markierung, zu anderen Nash-Gleichgewichten zu gelangen. Manche Gleichgewichte bleiben jedoch auch trotz dieser

Änderung vom Lemke-Howson-Algorithmus unentdeckt. Der Algorithmus von Govindan und Wilson sucht gleichermaßen einen gewissen Pfad ab, wodurch ebenfalls nicht garantiert ist, dass alle Nash-Gleichgewichte eines gegebenen Spiels gefunden werden. Da der Algorithmus nicht nach dem ersten berechneten Gleichgewicht abbricht, es aber auch nicht sicher ist, dass alle gefunden werden, kann zusammenfassend gesagt werden, dass er mindestens eines findet.

2.8 Degenerierte Spiele

Die Eindeutigkeit des LH-Pfads ist nur bei nicht-degenerierten Spielen garantiert. Der Grund hierfür ist, dass die Eindeutigkeit der Min-Ratio-Regel nicht gewährleistet ist, da mehrere Variable denselben Wert erzielen können. Es ist daher möglich, dass sich der Lemke-Howson-Algorithmus sich in einer unendlichen Schleife verliert. Um dieses Problem zu umgehen, dient sowohl [5] als auch [10] als Vorlage.

Ein Bimatrix-Spiel, bei dem eines der Polytope P und Q nicht einfach ist, ist *degeneriert*. Diese Aussage ist zu Definition 2.8 äquivalent.

Zur Veranschaulichung eines degenerierten Spiels dient folgendes Beispiel.

$$A = \begin{pmatrix} 3 & 3 \\ 2 & 5 \\ 0 & 6 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 3 \\ 2 & 6 \\ 3 & 1 \end{pmatrix}$$

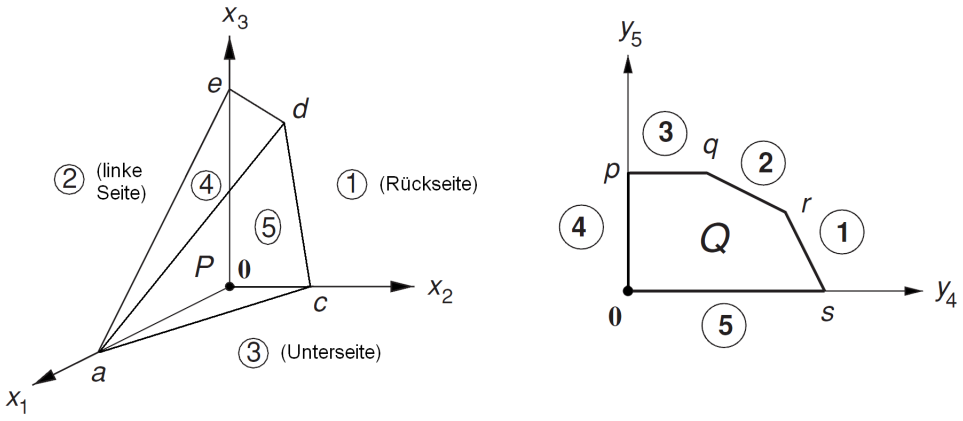

Abbildung 9: Die Beste-Antwort-Polytope P und Q.

Da das Polytop P nicht einfach ist (Punkt a besitzt mehr als 3 Facets), handelt es sich um ein degeneriertes Spiel.

In degenerierten Spielen können unendlich viele Gleichgewichte auftreten. In dem obigen Beispiel besitzt die Ecke a in Polytop P, die im Gegensatz zu Abbildung 7 hier mit der Ecke b zusammenfällt, gemeinsam mit einem beliebigen Punkt auf der Kante zwischen den Ecken r und s in Polytop Q alle Markierungen. Daher bildet die reine Strategie $(1, 0, 0)$ von Spieler 1 gemeinsam mit einer gemischten Strategie $(y_4, 1 - y_4)$ für $\frac{2}{3} \leq y_4 \leq 1$ von Spieler 2 ein Nash-Gleichgewicht. Diese gemischten Strategien sind die Elemente von der konvexen Hülle der Eckpunkte r und s. Im Folgenden wird die *konvexe Hülle* einer Menge U als $coH(U)$ bezeichnet.

Satz 2.32. *Sei $(x, y) \in P \times Q$, wobei P und Q die Polytope eines Bimatrix-Spiels (A, B) sind. Dann sind äquivalent:*

(i) *$(nrml(x), nrml(y))$ ist ein Nash-Gleichgewicht.*

(ii) *Es existiert eine Menge U von Ecken aus $P\backslash\{\mathbf{0}\}$ und eine Menge V von Ecken aus $Q\backslash\{\mathbf{0}\}$, sodass $x \in coH(U)$ und $y \in coH(V)$, und jedes Tupel u, v aus $U \times V$ ist vollständig markiert.*

Ein Beweis hierfür findet sich in [10].

Eine modifizierte Version des Ecken-Test-Algorithmus aus Kapitel 2.3 bedient sich der obigen Aussage und gibt eine vollständige Beschreibung aller Nash-Gleichgewichte für ein degeneriertes Bimatrix-Spiel aus.

Algorithmus 2.33. *(Algorithmus für degenerierte Bimatrix-Spiele)*

Input: *Ein degeneriertes Bimatrix-Spiel.*

Output: *Alle Nash-Gleichgewichte.*

for *jede Ecke x von $P\backslash\mathbf{0}$* ***do***
  for *jede Ecke y von $Q\backslash\mathbf{0}$* ***do***
    if *(x, y) ist vollständig markiert* ***then***
      Speichere (x, y) in einen Vektor w

for *jede Ecke x aus $P\backslash\mathbf{0}$, die mehrmals in einem Tupel (x, y) aus w vorkommt* ***do***
  Speichere (x, U) in einen Vektor u, wobei U die Menge aller zu x gehörigen Ecken y ist.

for *jede Ecke y aus $Q\backslash\mathbf{0}$, die mehrmals in einem Tupel (x, y) aus w vorkommt* ***do***
  Speichere (V, y) in einen Vektor v, wobei V die Menge aller zu y gehörigen Ecken x ist.

Alle (x, y) aus w vereinigt mit den Tupel $(x, coH(U))$ aus dem Vektor u und den Tupel $(coH(V), y)$ aus dem Vektor v ergeben normalisiert alle Nash-Gleichgewichte.

2.9 Extensive Spiele

Eine andere Art als die oben behandelten Normalform-Spiele sind Spiele in extensiver Form. Die wesentlichsten Grundlagen wurden bereits in Kapitel 2.1 angeführt. Um hier näher ins Detail zu gehen, wird den Arbeiten [5] und [14] gefolgt.

Zur Darstellung von extensiven Spielen wird ein Baumdiagramm verwendet, in welchem die Knoten den jeweiligen Zuständen des Spiels entsprechen. In jedem dieser Zustände setzt nur ein einziger Spieler entlang der Kanten seine Aktion. Das Spiel startet am Anfangsknoten (Wurzel) und endet an den Endknoten (Blätter), die die Information der Auszahlung enthalten. Alle Knoten außer den Endknoten werden als *Entscheidungsknoten* bezeichnet. Die möglichen Züge eines Spielers werden durch die nach unten wegführenden Kanten bestimmt.

Entscheidungsknoten können in Informationsmengen eingeteilt werden. Eine solche *Informationsmenge* beinhaltet alle Knoten, die demselben Spieler angehören und dem zusätzlich auch noch dieselben Strategien zur Auswahl stehen. Wenn der Spieler an der Reihe ist, weiß er lediglich in welcher Informationsmenge, nicht aber in welchem Entscheidungsknoten er sich befindet. In einem Spiel mit perfekter Information besteht jede Informationsmenge aus genau einem Knoten. In Folge bezeichne H_i die Menge an Informationsmengen von Spieler i, $h \in H_i$ eine Informationsmenge und C_h die Menge aller möglichen Züge in h. Zur Veranschaulichung dient folgendes Beispiel.

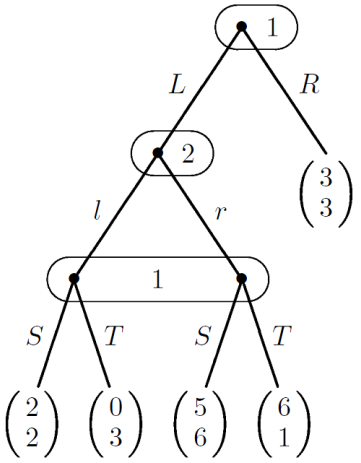

Abbildung 10: Baumdiagramm eines extensiven 2-Personen-Spiels

Es beginnt Spieler 1. Dieser hat die Wahl zwischen L und R. Bei einer Wahl von R endet das Spiel sofort mit der Auszahlung (3,3) ohne dass Spieler 2 an die Reihe kommt. Andernfalls muss sich Spieler 2 zwischen den Aktionen l und r entscheiden.

Spieler 1 erfährt nicht wie sich Spieler 2 entschieden hat und weiß deshalb nicht, in welchem der beiden Knoten der Informationsmenge er sich befindet. Dies wird durch die grafische Zusammenfassung der Entscheidungsknoten dargestellt. Nach einer Wahl zwischen S und T durch Spieler 1 endet das Spiel mit der jeweiligen Auszahlung.

Definition 2.34 (Verhaltensstrategie)**.** *Eine Verhaltensstrategie β von Spieler i ist eine Wahrscheinlichkeitsverteilung über die Handlungsalternativen C_h, die Spieler i in jeder Entscheidungssituation $h \in H_i$ zur Verfügung stehen.*

Das bedeutet, dass $\beta(c) \geq 0$ für $c \in C_h$ und $\sum_{c \in C_h} \beta(c) = 1$ für alle $h \in H_i$.

Eine *reine Strategie* ist eine Verhaltensstrategie, bei der jede Aktion deterministisch ausgewählt wird. Sie kann als Element $(c_h)_{h \in H_i}$ aus $\prod_{h \in H_i} C_h$ betrachtet werden. Im extensiven Spiel aus Abbildung 10 ist beispielsweise das Tupel (L, S) eine reine Strategie von Spieler 1.

Eine Auflistung aller reinen Strategien in einer Tabelle mit den zugehörigen erwarteten Auszahlungen definiert die *Normalform* des Spiels.

$$A = \begin{array}{cc} l & r \\ \hline 2 & 5 \\ 0 & 6 \\ 3 & 3 \\ 3 & 3 \end{array} \begin{array}{l} \langle L, S \rangle \\ \langle L, T \rangle \\ \langle R, S \rangle \\ \langle R, T \rangle \end{array} \qquad B = \begin{array}{cc} l & r \\ \hline 2 & 6 \\ 3 & 1 \\ 3 & 3 \\ 3 & 3 \end{array} \begin{array}{l} \langle L, S \rangle \\ \langle L, T \rangle \\ \langle R, S \rangle \\ \langle R, T \rangle \end{array}$$

Abbildung 11: Normalform des Spiels aus Abbildung 10

Bei gegebener Normalform können Spieler eine gemischte Strategie wählen, welche eine Wahrscheinlichkeitsverteilung über die reinen Strategien darstellt. Im Gegensatz dazu können auch Verhaltensstrategien verwendet werden. In diesem Fall wird die reine Strategie erst dann ausgewählt, wenn der Spieler die jeweilige Informationsmenge erreicht.

Algorithmen, die Nash-Gleichgewichte mit Hilfe der Normalform berechnen, sind meist sehr rechenaufwendig. Deshalb wurden zwei Lösungsmethoden entwickelt, auf die nun näher eingegangen wird.

2.9.1 Teilspielperfekte Gleichgewichte

Ein *Teilspiel* eines extensiven Spiels ist ein Teilbaum mitsamt den Auszahlungen des Spielbaums. Es beinhaltet alle Informationsmengen, die einen Knoten des Teilbaums enthalten. Die Wurzel eines Teilspiels ist eine Informationsmenge, welche nur einen

Knoten enthält. Das Teilspiel in Abbildung 12 kann als 2×2-Spiel in Normalform betrachtet werden, in dem die Spieler gleichzeitig ihre Strategie wählen. So lässt sich ein gemischtes Nash-Gleichgewicht für dieses Teilspiel berechnen. Nun wird das Teilspiel durch die erwarteten Auszahlungen ersetzt und übrig bleibt ein einfaches Spiel mit zwei Strategien L und R. Das auf diese Art bestimmte Nash-Gleichgewicht ist *teilspielperfekt*, was bedeutet, dass es in jedem Teilspiel ein weiteres Nash-Gleichgewicht induziert.

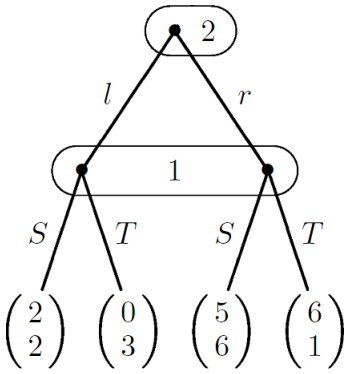

Abbildung 12: Ein Teilspiel des Spiels aus Abbildung 10

Algorithmus 2.35. *(Teilspielperfekte Gleichgewichte)*
Input: *Ein extensives Spiel.*
Output: *Ein teilspielperfektes Nash-Gleichgewicht.*
for *jedes Teilspiel in größer werdender Reihenfolge* ***do***
berechne ein Nash-Gleichgewicht des Teilspiels und ersetze das Teilspiel durch einen neuen Endknoten mit den erwarteten Auszahlungen des Gleichgewichts.

In einem Spiel mit vollkommener Information ist jeder Knoten die Wurzel eines Teilspiels. Da in jedem Teilspiel nur ein Spieler zum Zug kommt, ist immer eine deterministische Strategie optimal. Dadurch hat jedes Spiel mit vollkommener Information ein teilspielperfektes Nash-Gleichgewicht, in dem jeder Spieler eine reine Strategie wählt. In der Literatur wird der Algorithmus bei vollkommener Information als Rückwärts-Induktion oder auch Zermelo's Algorithmus bezeichnet. In Spielen mit unvollkommener Information kann man für das teilspielperfekte Nash Gleichgewicht auch gemischte Strategien benötigen.

2.9.2 Die sequentielle Form

Nun wird die Funktionsweise eines Algorithmus erarbeitet, der ein Nash-Gleichgewicht eines Zwei-Personen-Spiels in extensiver Form berechnet. Er bedient sich dabei den sogenannten Sequenzen, wodurch eine schnelle Berechnung möglich ist. Sollten mehrere Gleichgewichte existieren, so können manche von ihnen gefunden werden, indem man einen anderen Startpunkt für den Algorithmus wählt.

Definition 2.36 (Sequenz)**.** *Die Sequenz von Aktionen für einen Spieler ist die Menge aller Aktionen, die dieser Spieler wählen muss, um vom Wurzelknoten zu einem gewissen Knoten gelangen zu können.*

Wählt ein Spieler alle Aktionen einer Sequenz, bedeutet dies nicht zwingend, dass der gewünschte Knoten auch erreicht wird, da dies auch von den Aktionen des anderen Spielers abhängig ist.

Im Folgenden wird davon ausgegangen, dass beide Spieler *perfektes Erinnerungsvermögen* besitzen. Dies bedeutet, dass Informationen, die einem Spieler zum Zeitpunkt einer zuvor von ihm getroffenen Entscheidung bekannt waren, auch bei späteren Entscheidungen weiterhin bekannt sind. Das heißt weiters, dass alle Knoten in einer Informationsmenge h eines Spielers dieselbe Sequenz σ_h von Aktionen definieren. Unter dieser Voraussetzung ist jede Aktion c im Knoten h die letzte Aktion einer eindeutigen Sequenz $\sigma_h \cup c$. Der Einfachheit halber wird im Folgenden die Notation $\sigma_h c$ verwendet. Mit dieser Schreibweise lassen sich alle möglichen Sequenzen für einen Spieler außer der leeren Sequenz $\emptyset$ anschreiben. Die Menge aller Sequenzen S_i eines Spielers i ist daher

$$S_i = \{\emptyset\} \cup \{\sigma_h c \mid h \in H_i,\ c \in C_h\}.$$

Die Definition 2.34 einer Verhaltensstrategie β kann auf die Sequenzen σ in S_i ausgeweitet werden:

$$\beta[\sigma] = \prod_{c \in \sigma} \beta(c).$$

Eine reine Strategie π ist also eine Verhaltensstrategie mit $\pi(c) \in \{0, 1\}$ für alle Aktionen c. P_i bezeichne die Menge aller reinen Strategien. Folglich gilt $\pi[\sigma] = \prod_{c \in \sigma} \pi(c) \in \{0, 1\}$ für alle Sequenzen σ aus S_i.

Man kann auch die Normalform eines extensiven Spiels anschreiben, indem man die reine Strategien und davon gemischte Wahrscheinlichkeiten betrachtet. Eine *gemischte Strategie* μ eines Spielers i weist jedem $\pi \in P_i$ eine Wahrscheinlichkeit $\mu(\pi)$ zu. In der *sequentiellen Form* eines extensiven Spiels verwendet man anstelle von reinen Strategien eines Spielers seine Sequenzen. Eine Strategie eines Spielers i wird durch die Realisierungswahrscheinlichkeiten, mit welchen die Sequenzen σ gespielt werden, beschrieben. Für eine Verhaltensstrategie β sind diese offensichtlich $\beta[\sigma]$. Für eine

gemischte Strategie μ eines Spielers i lauten sie

$$\mu[\sigma] = \sum_{\pi \in P_i} \pi[\sigma]\mu(\pi). \tag{2}$$

Dadurch wird der Realisierungsplan x von μ bestimmt, der für Spieler 1 die Abbildung $x(\sigma) = \mu[\sigma]$ von S_1 nach (R) ist. Analog bezeichne y den Realisierungsplan von Spieler 2. Realisierungspläne besitzen zwei wichtige Eigenschaften.

Satz 2.37. *Für Spieler 1 ist x der Realisierungsplan einer gemischten Strategie genau dann, wenn $x(\sigma) \geq 0$ für alle $\sigma \in S_1$ und*

$$\begin{aligned} x(\emptyset) &= 1, \\ \sum_{c \in C_h} x(\sigma_h c) &= x(\sigma_h), \qquad h \in H_1 \end{aligned} \tag{3}$$

gilt.

Beweis. Da die Gleichung 3 für die Realisierungswahrscheinlichkeiten $x(\sigma) = \beta[\sigma]$ für eine Verhaltensstrategie β und daher auch für jede reine Strategie π erfüllt ist, gilt sie ebenso für ihre konvexen Kombinationen $\mu[\sigma] = \sum_{\pi \in P_i} \pi[\sigma]\mu(\pi)$. □

Zur leichteren Handhabung wird $(x_\sigma)_{\sigma \in S_1}$ als x und $(y_\sigma)_{\sigma \in S_2}$ als y bezeichnet. Für sie gilt

$$x \geq \mathbf{0}, \quad Ex = e, \quad y \geq \mathbf{0}, \quad Fy = f$$

mit geeigneten Matrizen E und F und Vektoren e und f gleich dem ersten kanonischen Einheitsvektor $(1, 0, \ldots, 0)^T$. E und e besitzen $1 + |H_1|$, F und f $1 + |H_2|$ Zeilen.

Die zweite oben erwähnte Eigenschaft bedient sich folgender Definition, die erstmals in [15] formuliert wurde.

Definition 2.38 (realisierungsäquivalent)**.** *Gemischte Strategien eines Spielers heißen realisierungsäquivalent, wenn sie bei gegebener Strategie des anderen Spielers dieselben Realisierungswahrscheinlichkeiten für alle Knoten des Baumes besitzen.*

Satz 2.39. *Zwei gemischte Strategien μ und μ' sind realisierungsäquivalent genau dann, wenn sie denselben Realisierungsplan besitzen. Das heißt $\mu[\sigma] = \mu'[\sigma]$ für alle $\sigma \in S_i$.*

Beweis. Betrachte die lineare Abbildung von $\mathbb{R}^{|P_i|}$ nach $\mathbb{R}^{|S_i|}$, die den Vektor $(\mu(\pi))_{\pi \in P_i}$ auf $(\mu[\sigma])_{\sigma \in S_i}$ abbildet. Gemischte Strategien mit demselben Bild sind offensichtlich realisierungsäquivalent. □

Die lineare Abbildung aus dem Beweis bildet den Simplex von gemischten Strategien eines Spielers auf das Polytop von Realisierungsplänen ab. Diese Polytope definieren die Strategie-Räume X und Y der Spieler in der sequentiellen Form:

$$X = \{x \mid x \geq \mathbf{0},\ Ex = e\}, \quad Y = \{y \mid y \geq \mathbf{0},\ Fy = f\} \tag{4}$$

Die Ecken von X und Y sind genau die reinen Strategien der Spieler bis auf Realisierungsäquivalenz.

Zur Veranschaulichung dient das extensive 2-Personen-Spiel aus Abbildung 13.

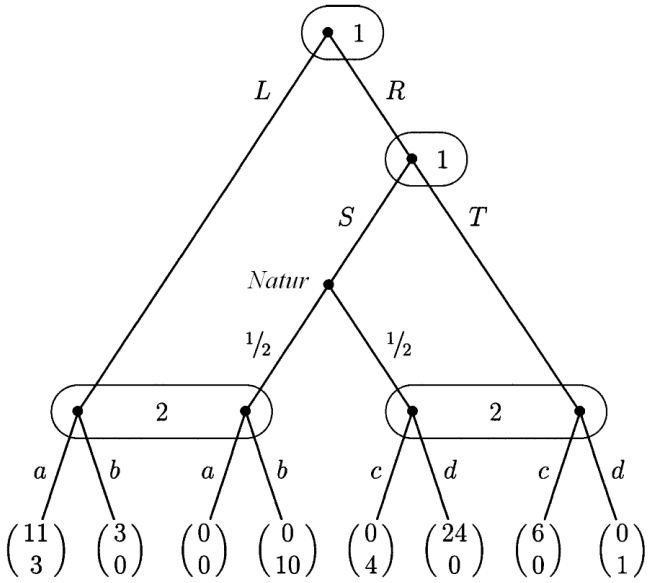

Abbildung 13: Baumdiagramm eines extensiven 2-Personen-Spiels

Es startet Spieler 1, der zwischen den Aktionen L und R wählen kann. Fällt seine Entscheidung auf R, kommt er gleich nochmals an die Reihe und muss sich zwischen S und T entscheiden. Wählt er S kommt die Natur zum Zug, die mit der angeführten Wahrscheinlichkeitsverteilung in das Spiel eingreift. Die Informationsmengen von Spieler 2 sind durch die zusammengefassten Knoten dargestellt. In der linken Informationsmenge kann dieser zwischen a und b entscheiden, in der rechten zwischen c und d.

Die Mengen der Sequenzen sind $S_1 = \{\emptyset, L, R, RS, RT\}$ und $S_2 = \{\emptyset, a, b, c, d\}$. Mit Hilfe von Satz 2.37 erhält man die Beziehungen

$$\begin{aligned} x(\emptyset) &= 1 \\ x(\emptyset) &= x(L) + x(R) \\ x(R) &= x(RS) + x(RT). \end{aligned}$$

Da $Ex = e = (1, 0, 0)^T$ gelten soll, formt man die Gleichungen um zu

$$\begin{array}{rrrrrrrr} x(\emptyset) & & & & & & = & 1 \\ -x(\emptyset) & +x(L) & +x(R) & & & & = & 0 \\ & & -x(R) & +x(RS) & +x(RT) & & = & 0 \end{array}$$

Wenn man analog für Spieler 2 vorgeht, kann man daraus die Matrizen

$$E = \begin{pmatrix} 1 & & & & \\ -1 & 1 & 1 & & \\ & & -1 & 1 & 1 \end{pmatrix} \quad F = \begin{pmatrix} 1 & & & & \\ -1 & 1 & 1 & & \\ -1 & & & 1 & 1 \end{pmatrix}$$

ablesen.

Wenn ein Paar von Sequenzen zu einem Endknoten führt, kann man die Auszahlungen der sequentiellen Form berechnen. Dazu multipliziert man die Auszahlungen am Endknoten mit der auf dem jeweiligen Pfad auftretenden Wahrscheinlichkeit, mit der die Natur eingreift. Die Ergebnisse stellt man in $|S_1| \times |S_2|$-Nutzen-Matrizen übersichtlich dar. Abbildung 14 zeigt diese Matrizen für obiges Beispiel.

$$A = \begin{array}{|cccc|c} \multicolumn{4}{c}{\emptyset \quad a \quad b \quad c \quad d} & \\ \hline & & & & \emptyset \\ 11 & 3 & & & L \\ & & & & R \\ 0 & 0 & 0 & 12 & RS \\ & & 6 & 0 & RT \\ \hline \end{array} \qquad B = \begin{array}{|cccc|c} \multicolumn{4}{c}{\emptyset \quad a \quad b \quad c \quad d} & \\ \hline & & & & \emptyset \\ 3 & 0 & & & L \\ & & & & R \\ 0 & 5 & 2 & 0 & RS \\ & & 0 & 1 & RT \\ \hline \end{array}$$

Abbildung 14: Nutzen-Matrizen der sequentiellen Form

Mithilfe der Dualität der linearen Programmierung kann gezeigt werden, dass jedes Nash-Gleichgewicht eines extensiven Spiels einem Paar (x, y) von Realisierungsplänen entspricht, sodass Vektoren u, v, r, s existieren, die die linearen Bedingungen

$$\begin{array}{rcrrrrcl} & & & & x & & \geq & \mathbf{0}, \\ & & & & & y & \geq & \mathbf{0}, \\ & & & & Ex & & = & e, \\ & & & & & Fy & = & f, \\ r & = & E^T u & & & -Ay & \geq & \mathbf{0}, \\ s & = & & F^T v & -B^T x & & \geq & \mathbf{0} \end{array}$$

und die komplementären Bedingungen

$$x^T r = 0, \quad y^T s = 0$$

erfüllen.

Die Schlupfvariablen s und r sind nichtnegativ und haben die Dimension $|S_1|$ bzw. $|S_2|$. Die Vektoren u und v hingegen sind nicht vorzeichenbeschränkt und haben die Dimension $1 + |H_1|$ bzw. $1 + |H_2|$.

Bei einem linearen Komplementaritätsproblem (kurz LKP) sind eine Matrix $M \in \mathbb{R}^{n \times n}$ und ein Vektor $b \in \mathbb{R}^n$ gegeben. Gesucht sind Vektoren $w,\ z \in \mathbb{R}^n$, sodass die folgenden drei Bedingungen gelten:

$$\begin{aligned} z, w &\geq \mathbf{0} \\ b + Mz &= w \\ z^T w &= 0 \end{aligned}$$

Das durch die obigen Gleichungen definierte LKP ist etwas allgemeiner. Hier ist $z = (u, v, x, y)^T$ und $w = (\mathbf{0}, \mathbf{0}, r, s)^T$. Die Komponenten u und v von z sind nicht vorzeichenbeschränkt, da die dazugehörigen Komponenten von w immer Null sind, wodurch die komplementären Bedingungen für diese Komponenten immer erfüllt sind.

Lemke veröffentlichte 1965 die Arbeit [16], in der er einen Algorithmus zur Lösung eines LKPs vorstellte. Unter der Verwendung eines zusätzlichen n-Vektors d und einer dazugehörigen skalaren Variable z_0, erhält man das System

$$\begin{aligned} z, w, z_0 &\geq \mathbf{0} \\ b + Mz + dz_0 &= w \\ z^T w &= 0 \end{aligned}$$

Zu Beginn besitzt z_0 einen positiven Wert. Bei der Pivotisierung verlässt eine Variable des komplementären Paares (z_i, w_i) die Basis und ihr Komplement wird aufgenommen. Bei dem oben erwähnten allgemeineren LKP verlassen die nicht vorzeichenbeschränkten Komponenten u und v nie die Basis. Die Pivotisierung wird solange durchgeführt, bis z_0 die Basis verlässt und daher den Wert Null annimmt. Dadurch ist das LKP gelöst.

In [17] findet sich eine genaue Beschreibung von Lemke's Algorithmus und ein Beweis, dass er ein Ergebnis für das LKP, welches man aus der sequentiellen Form erhält, liefert, wenn $d = (1, 1, \ldots, 1)^T$ gilt.

Als zufälliger Start-Vektor dient ein Paar von Realisierungsplänen (p, q), sodass

$$p \geq \mathbf{0}, \quad Ep = e, \quad q \geq \mathbf{0}, \quad Fq = f$$

gilt. Der zusätzliche Vektor d wird in Abhängigkeit des Start-Vektors als

$$d = \begin{pmatrix} e \\ f \\ -Aq \\ -B^T p \end{pmatrix}$$

definiert. Das allgemeine LKP wird mit d erweitert und man erhält

$$\begin{array}{rcccccccccc}
 & & & & x & & & & \geq & \mathbf{0}, \\
 & & & & & & y & & \geq & \mathbf{0}, \\
 & & & & & & & z_0 & \geq & \mathbf{0}, \\
 & & & & Ex & & & +ez_0 & = & e, \\
 & & & & & & Fy & +fz_0 & = & f, \\
r & = & E^T u & & & & -Ay & -(Aq)z_0 & \geq & \mathbf{0}, \\
s & = & & F^T v & -B^T x & & & -(B^T p)z_0 & \geq & \mathbf{0}
\end{array}$$

und die komplementären Bedingungen

$$x^T r = 0, \quad y^T s = 0.$$

Eine Anfangslösung ist durch $z_0 = 1$, $x = \mathbf{0}$, $y = \mathbf{0}$ und geeignete Vektoren u und v, sodass $E^T u \geq Aq$ und $F^T v \geq B^T p$, gegeben. Wie man leicht sieht, sind dadurch die Bedingungen $r \geq \mathbf{0}$ und $s \geq \mathbf{0}$ erfüllt.

Der Algorithmus berechnet einen stückweise linearen Pfad, auf dem die Bedingungen des erweiterten LKPs für alle Punkte des Pfades erfüllt sind. Im Folgenden wird gezeigt, dass dadurch ein Pfad in $X \times Y$ definiert wird, welcher bei dem Start-Vektor (p, q) beginnt und in einem Gleichgewicht endet. Der folgende Satz zeigt, wie die Punkte $(\bar{x}, \bar{y})$ auf dem Pfad berechnet werden.

Satz 2.40. *Für eine Lösung des erweiterten LKPs (u, v, x, y, z_0) sei*

$$\bar{x} = x + pz_0, \quad \bar{y} = y + qz_0.$$

Dann ist $\bar{x} \in X$, $\bar{y} \in Y$ und $x(\emptyset) = y(\emptyset) = 1 - z_0 \geq 0$.

Beweis. $\bar{x} \geq 0$, $\bar{y} \geq 0$ gilt offensichtlich. Da $E\bar{x} = E(x + pz_0) = Ex + (Ep)z_0 = Ex + ez_0 = e$ folgt $\bar{x} \in X$ und analog $\bar{y} \in Y$. Aus $1 = \bar{x}(\emptyset) = x(\emptyset) + p(\emptyset)z_0 = x(\emptyset) + z_0$ erhält man $x(\emptyset) = 1 - z_0 \geq 0$. Analoges Vorgehen führt zu $y(\emptyset) = 1 - z_0 \geq 0$. □

Nach Satz 2.40 erfüllt jede Lösung des erweiterten LKPs $0 \leq z_0 \leq 1$. Der Algorithmus bricht ab, sobald $z_0 = 0$, sodass $x = \bar{x} \in X$, $y = \bar{y} \in Y$ und (x, y) ein Gleichgewicht ist.

Während der Pivotisierung mit $0 < z_0 < 1$ kann das Paar $(\bar{x}, \bar{y})$ als konvexe Kombination von einem Paar (x^*, y^*) von Realisierungsplänen und dem Start-Vektor (p, q) mit den Gewichten $1 - z_0$ und z_0 angesehen werden. Dies gelingt, indem man

$$x^* = \frac{x}{1 - z_0}, \quad y^* = \frac{y}{1 - z_0}$$

setzt, sodass $\bar{x} = x + pz_0 = x \cdot (1 - z_0) + pz_0$ und $\bar{y} = y + qz_0 = y \cdot (1 - z_0) + qz_0$ gilt. Ersetzt man in $Ex = e(1 - z_0)$ das x durch $x^* \cdot (1 - z_0)$, erhält man nach dem Kürzen $Ex^* = e$. Die gleichen Schritte mit y durchgeführt ergeben $x^* \in X$ und $y^* \in Y$. Der folgende Satz zeigt einen wichtigen Zusammenhang zwischen $\bar{x}$ und x^*.

Satz 2.41. *Sei (u, v, x, y, z_0) wieder eine Lösung des erweiterten LKPs mit $z_0 < 1$. Dann ist x^* die beste Antwort auf $\bar{y}$ und y^* die beste Antwort auf $\bar{x}$.*

Beweis. Ein Realisierungsplan x ist die beste Antwort auf $\bar{y}$ genau dann, wenn er die erwartete Auszahlung $x^T(A\bar{y})$ unter den Nebenbedingungen $Ex = e$ und $x \geq \mathbf{0}$ maximiert. Das Duale Problem dazu ist, ein u zu finden, welches $e^T u$ unter den Nebenbedingungen $E^T u \geq A\bar{y}$ minimiert. Zulässige Lösungen x und u zu diesem Primal-Dual-Problem sind optimal genau dann, wenn sie die komplementäre Schlupfbedingung $x^T(E^T u - A\bar{y}) = 0$ erfüllen.
Die Komponenten x und u der gegebenen Lösung erfüllen alle diese Bedingungen, abgesehen von $Ex = e$. Ersetzt man x wieder durch $x^*(1 - z_0)$, so bleibt die komplementäre Schlupfbedingung immer noch erfüllt (da $z_0 < 1$). Es gilt nun aber zusätzlich $Ex^* = e$. Als Lösung des Primal-Dual-Problems optimiert x^* die erwartete Auszahlung $x^T A\bar{y}$ und ist daher die beste Antwort auf $\bar{y}$. Selbiges gilt für y^* und $\bar{x}$. □

3 Algorithmen zur Bestimmung evolutionär stabiler Strategien

In der ab den 1980er Jahren entwickelten evolutionären Spieltheorie fand die Algorithmische Spieltheorie auch eine Anwendung in der Biologie. Durch eine Zusammenarbeit von Mathematikern, Biologen und Informatikern entstanden mehrere Algorithmen in diesem Gebiet, wovon drei ausgewählt wurden und hier beschrieben werden.

3.1 Grundlagen

In der evolutionären Spieltheorie wählen die Spieler ihre Strategien nicht aufgrund rationalen Verhaltens, sondern jeder Spieler verfolgt die ihm bei der Geburt vererbten Strategien. Dieser Teil der Spieltheorie geht ursprünglich auf die Arbeiten [18] und [19] der Biologen John Maynard Smith und George R. Price zurück. Sie entwickelten gemeinsam den Begriff der evolutionär stabilen Strategie, welcher hier später definiert wird. Um den Zusammenhang mit der Biologie zu verdeutlichen, werden im Folgenden die Spieler auch als Individuen bezeichnet. Es wird untersucht, welche Verhaltensweisen das Potenzial haben, in der gesamten Population zu bestehen und welche von anderen vertrieben werden.

Die folgende Einführung und das erklärende Beispiel folgen [21]. Jedes Individuum interagiert mit den anderen Organismen einer Population und der Erfolg eines jeden Individuums hängt davon ab, wie sein Verhalten und das Verhalten der anderen aufeinander wirken. Die Fitness eines Organismus kann daher nicht in Isolation gemessen werden, sondern nur im direkten Zusammenhang mit der vollständigen Population, in der er lebt. In der Spieltheorie entsprechen die Fitness der Auszahlung und die genetisch veranlagten Verhaltensweisen den Strategien. Die Höhe der Auszahlung hängt von den Strategien der Organsimen ab, mit denen das jeweilige Individuum interagiert. Ebenso existiert ein tiefer Zusammenhang zwischen den Ergebnissen der Evolution einer Population und den Gleichgewichtskonzepten aus der herkömmlichen Spieltheorie.

Für das weitere Verständnis dient das folgende Beispiel.

Beispiel 3.1.

Betrachte eine Art von Käfer, dessen Fitness hauptsächlich durch das Auffinden von Nahrung und die Verarbeitung der darin enthaltenen Nährstoffe bestimmt wird. Durch eine Mutation entsteht eine neue Käferart, die sich durch einen deutlich größeren Körperbau auszeichnet. Durch seine Größe benötigt dieser Mutant mehr Nährstoffe als die ursprüngliche Käferart. Diese Tatsache hat eine negative Auswirkung auf ihre Fitness, was darauf hindeutet, dass sich der Mutant nicht durchsetzen können wird. Allerdings kommt noch eine zusätzliche Annahme ins Spiel. Die Mutanten haben im Kampf um die Nahrung einen Vorteil durch ihre Körpergröße.

Abbildung 15: Zwei Hirschkäfer im Kampf

Treffen zwei Käfer aufeinander, können folgende Szenarien entstehen:

- Wenn beide Käfer dieselbe Körpergröße besitzen, erhalten sie die gleiche Menge an Nahrung.
- Wenn sich Käfer unterschiedlicher Größe die Nahrung streitig machen, erhält der größere von ihnen den Hauptanteil des Fundes.

Große Käfer erhalten generell weniger Fitnessgewinn von einer bestimmten Menge an Nahrung, da ein Teil der daraus gewonnen Nährstoffe in die Aufrechterhaltung ihrer Körpergröße investiert wird.

Die Fitness, die jeder Käfer aus so einem Streit um Nahrung erhält, kann als Auszahlung in einem Zwei-Personen-Spiel zwischen einem ersten und einem zweiten Käfer angesehen werden. Der erste Käfer spielt eine der Strategien "Klein" oder "Groß" und der zweite Käfer antwortet ebenfalls mit einer der beiden Strategien. Die Auszahlungen für die zwei Käfer findet man in Tabelle 6.

		Käfer 2	
		Klein	Groß
Käfer 1	Klein	5, 5	1, 8
	Groß	8, 1	3, 3

Tabelle 6: Auszahlungsmatrix des Käferspiels

Angenommen die gefundene Nahrung besitzt einen Gesamtnutzen von zehn Einheiten. Wenn sich zwei kleine Käfer duellieren, erhalten beide Käfer wie gefordert den gleichen Nutzen - den halben Gesamtnutzen, fünf Einheiten. Im Gegensatz dazu erhält ein großer Käfer nach einem Kampf gegen einen kleinen Käfer den Großteil der Nahrung. Da große Käfer, wie oben erwähnt, mehr Nährstoffe benötigen, vermindert sich der Gesamtnutzen ein wenig, sodass der große Käfer acht Einheiten und der kleine Käfer eine Einheit erhält. Treffen zwei große Käfer aufeinander, tritt ein zusätzlicher Effekt ein. Durch die harte Gegenwehr des Konkurrenten haben beide Kontrahenten einen hohen Energieaufwand, wodurch sich der Gesamtnutzen abermals verringert. Somit erhalten beide Käfer eine Auszahlung von drei Einheiten.

Dieses Zwei-Personen-Spiel stellt also einen Streit um eine gefundene Nahrung dar. Ein derartiges Szenario wiederholt sich im Laufe des Lebens eines jeden Käfers immer wieder mit zufällig ausgewählten Kontrahenten. Die gesamte Fitness eines Käfers ist gleich der durchschnittlichen Fitness, die es von den Nahrungskämpfen erhält. Je höher eben diese Fitness ist, desto größer ist die Fähigkeit, Nachwuchs zu zeugen und somit seine Gene (Strategie) weiter zu verbreiten.

Die Auszahlungsmatrix des Käferspiels aus Tabelle 6 erinnert stark an die des Gefangenendilemmas aus Beispiel 2.1. Der gravierende Unterschied liegt in der Art der Auswahl der Strategien. Beim Gefangenendilemma können beide Akteure eine Strategie wählen. Im Käferspiel ist die Strategie eines jeden Individuums fix in den Genen verankert und wird das ganze Leben lang gespielt. Demzufolge wird anstelle des Nash-Gleichgewichts (welches auf dem Nutzenzuwachs durch Wechsel der eigenen Strategie beruht) ein an die Veränderung der Population durch evolutionäre Kräfte angepasster Begriff benötigt.

Ein Nash-Gleichgewicht in einem Zwei-Personen-Spiel hat die Eigenschaft, dass kein Spieler von der Strategie, die er gerade verwendet, abweichen wird. Tritt eine Kombination von Strategien auf, die ein Gleichgewicht darstellt, so bleibt sie daher auch

bestehen. Analog dazu existiert in evolutionären Spielen die *evolutionär stabile Strategie* (ESS). Sie ist eine genetisch veranlagte Strategie, welche bestehen bleibt, wenn sie in der Population vorherrschend ist.

Im Folgenden werden ein paar für das weitere Verständnis wichtige Begriffe formal definiert.

Definition 3.2 (Fitness)**.** *Die Fitness F eines Organismus in einer Population ist die erwartete Auszahlung, die er von der Interaktion mit einem zufälligen Mitglied der Population erhält.*

Definition 3.3 (eindringen)**.** *Eine Strategie T dringt in eine Strategie S mit einer Stärke x ein (für ein kleines x), wenn ein x-Bruchteil der Population die Strategie T und ein $1-x$-Bruchteil die Strategie S verwendet.*

Definition 3.4 (evolutionär stabile Strategie)**.** *Eine Strategie S ist evolutionär stabil, wenn eine positive Zahl y existiert, für die gilt: Für jede eindringende Strategie T mit einer Stärke $x < y$ ist die Fitness eines Organismus mit Strategie S strikt größer als die Fitness eines Organismus mit Strategie T.*

Man betrachte nun wieder Beispiel 3.1 und es bezeichne K die Erbanlage eines kleinen Käfers und G die eines großen. Welche dieser Erbanlagen ist evolutionär stabil?

Es sei zunächst die Strategie K vorherrschend. Fällt eine kleine Gruppe mit Strategie G mit der Stärke x ein, berechnet sich die Fitness einer jeden Käferart wie folgt.

$$\begin{aligned} F(K) &= (1-x)\cdot 5 + x\cdot 1 \\ F(G) &= (1-x)\cdot 8 + x\cdot 3 \end{aligned}$$

Damit K eine evolutionär stabile Strategie ist, muss $F(K) > F(G)$ sein.

$$\begin{aligned} (1-x)\cdot 5 + x\cdot 1 &> (1-x)\cdot 8 + x\cdot 3 \\ 5 - 4x &> 8 - 5x \\ x &> 3 \quad \lightning \end{aligned}$$

Da $0 < x < 1$ gelten muss, ist K keine evolutionär stabile Strategie. Betrachte man nun eine Population G in die eine Gruppe mit Strategie K mit Stärke x eindringt.

$$\begin{aligned} F(G) &= (1-x)\cdot 3 + x\cdot 8 \\ F(K) &= (1-x)\cdot 1 + x\cdot 5 \end{aligned}$$

Damit G eine evolutionär stabile Strategie ist, muss $F(G) > F(K)$ sein.

$$\begin{aligned}(1-x)\cdot 3 + x\cdot 8 &> (1-x)\cdot 1 + x\cdot 5\\ 3+5x &> 1+4x\\ x &> -2 \quad \checkmark\end{aligned}$$

Man sieht hier, dass unabhängig von der Wahl von x die Bedingung für eine evolutionär stabile Strategie erfüllt ist. Die Käfer mit der Erbanlage G werden die Invasion der anderen Art überstehen.

		Organismus 2	
		S	T
Organismus 1	S	$a,\ a$	$b,\ c$
	T	$c,\ b$	$d,\ d$

Tabelle 7: Allgemeines symmetrisches Spiel

Betrachte nun eine allgemeine Form dieses evolutionären Spiels wie in Tabelle 7. Um zu überprüfen, ob S eine ESS ist (bei Invasion von T), führt man analog zu obigem Beispiel dieselben Rechenschritte durch und erhält die Bedingung

$$a(1-x) + bx > c(1-x) + dx. \tag{5}$$

Satz 3.5. *In einem symmetrischen Zwei-Personen-Spiel mit zwei Strategien ist S evolutionär stabil genau dann, wenn eine der beiden Bedingungen erfüllt ist:*

(i) $a > c$

(ii) $a = c$ *und* $b > d$

Beweis. Lässt man in Bedingung 5 x gegen Null gehen, so wird die linke Seite zu a und die rechte Seite zu c. Daher ist die Bedingung für eine ESS erfüllt, wenn $a > c$ gilt. Ist $a = c$, dann ist die linke Seite größer genau dann, wenn $b > d$ gilt. □

Nach der allgemeinen Formulierung einer ESS wird nun der Zusammenhang mit dem Nash-Gleichgewicht in einem Zwei-Personen-Spiel erarbeitet.

Betrachtet man das Spiel aus Tabelle 7 lässt sich die Bedingung, dass (S, S) ein Nash-Gleichgewicht ist, als

$$a \geq c$$

anschreiben. Vergleicht man diese Ungleichung mit der Bedingung, dass S evolutionär stabil ist,

$$(i)\ a > c \quad \textit{oder}\ (ii)\ a = c \quad \textit{und}\ b > c,$$

erhält man folgende Implikation.

Satz 3.6. *Wenn S eine evolutionär stabile Strategie ist, dann ist (S, S) ein Nash-Gleichgewicht.*

Die Umkehrung muss nicht immer gelten. Als Gegenbeispiel betrachte das Zwei-Personen-Spiel:

		Organismus 2	
		S	T
Organismus 1	S	4, 4	0, 4
	T	4, 0	3, 3

Tabelle 8: Symmetrisches Zwei-Personen-Spiel

Der Punkt (S, S) ist ein Nash-Gleichgewicht, jedoch ist die Strategie S keine evolutionäre stabile Strategie, da $a = c$ und $b < d$. Weiters existiert ein Zusammenhang zwischen ESS und dem Konzept des strikten Nash-Gleichgewichts.

Definition 3.7 (striktes Nash-Gleichgewicht)**.** *Ein striktes Nash-Gleichgewicht ist ein Tupel (x, y) von gemischten Strategien, die gegenseitig strikt beste Antworten sind.*

Betrachtet man wieder das allgemeine Spiel aus Tabelle 7 so ist das Tupel (S, S) ein striktes Nash-Gleichgewicht wenn $a > c$ gilt. Somit folgt aber sofort, dass die Strategie S eine ESS ist. Wiederum muss die Umkehrung nicht gelten. Zusammenfassend lassen sich diese Implikationen wie folgt darstellen.

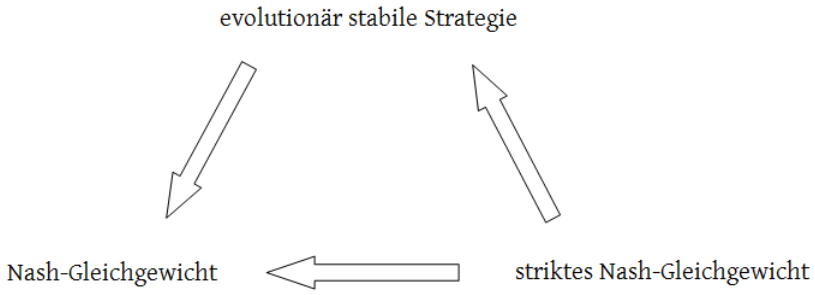

Abbildung 16: Zusammenhang zwischen ESS, Nash und striktem Nash-Gleichgewicht

Es ist möglich, dass weder S noch T evolutionär stabil sind. Das in der Literatur am häufigsten beschriebene Beispiel ist das Habicht-Taube-Spiel aus Tabelle 9. Wie im Käfer-Spiel duellieren sich zwei Tiere um Nahrung. Ein Tier mit der Strategie Habicht (H) verhält sich aggressiv, während hingegen die Strategie Taube (T) eine

passive Verhaltensweise symbolisiert. Treffen zwei Tauben aufeinander, so teilen sie sich die gefundene Nahrung gerecht auf. Im Streit zwischen Habicht und Taube erhält der aggressive Vogel den Großteil des Fundes. Im Unterschied zum Käfer-Spiel erhält niemand etwas, wenn zwei aggressive Kontrahenten aufeinander treffen, da sie sich gegenseitig im Kampf sehr stark verwunden.

		Tier 2	
		Taube	Habicht
Tier 1	Taube	3, 3	1, 5
	Habicht	5, 1	0, 0

Tabelle 9: Das Habicht-Taube-Spiel

Da weder H noch T die beste Antwort auf sich selbst ist, können beide Strategien nicht evolutionär stabil sein. Betrachtet man das Modell als Zwei-Personen-Spiel, in dem beide Kontrahenten ihre Strategien wählen anstatt diese in ihrer Erbanlage verankert haben, dann findet man schnell die beiden reinen Nash-Gleichgewichte (H, T) und (T, H). Um allerdings herauszufinden, wie sich das Habicht-Taube-Spiel unter evolutionären Kräften verhält, muss die Definition der evolutionären stabilen Strategie auf gemischte Strategien erweitert werden.

Der Begriff der evolutionär stabilen gemischten Strategie wird nun anhand des allgemeinen symmetrischen Spiels aus Tabelle 7 erarbeitet. Die Wahrscheinlichkeit, dass Organismus 1 die Strategie S spielt, ist p und die Wahrscheinlichkeit für T daher $1-p$. Wenn Organismus 1 die gemischte Strategie p und Organismus 2 die gemischte Strategie q verwendet, ist $p \cdot q$ die Wahrscheinlichkeit für die Wahl der Strategien (S, S). Durch analoges Berechnen der Wahrscheinlichkeiten der anderen Tupel erhält man den erwarteten Nutzen

$$N(p, q) = pqa + p(1-q)b + (1-p)qc + (1-p)(1-q)d$$

für Organismus 1. Die Fitness eines Organismus ist nach wie vor der erwartete Nutzen einer Interaktion mit einem zufälligen Mitglied der Population. Diese Erkenntnisse genügen für die nachfolgende Definition.

Definition 3.8 (evolutionär stabile gemischte Strategie)**.** *Im allgemeinen symmetrischen Spiel ist p eine evolutionär stabile gemischte Strategie, wenn eine positive Zahl y existiert, für die gilt: Für jede eindringende gemischte Strategie q mit einer Stärke $x < y$ ist die Fitness eines Organismus mit Strategie p strikt größer als die Fitness eines Organismus mit Strategie q.*

Da hier also zwei Definitionen vorliegen, muss im Vorhinein klar festgelegt werden, ob der Eindringling gemischte oder nur reine Strategien verwenden darf, damit eine Strategie auf evolutionäre Stabilität überprüft werden kann.

Es kann nun eine Bedingung aufgestellt werden, die angibt, ob p evolutionär stabil ist. Sei y eine kleine positive Zahl. Dann gilt für alle $x < y$ und alle gemischten Strategien $q \neq p$

$$(1-x)N(p,p) + xN(p,q) > (1-x)N(q,p) + xN(q,q). \tag{6}$$

Wie auch bei reinen Strategien impliziert eine evolutionär stabile gemischte Strategie p das Nash-Gleichgewicht (p,p), da $N(p,p) \geq N(q,p)$ gilt und somit p eine beste Antwort auf sich selbst ist. Wegen der strikten Ungleichung in 6 kann (p,p) ein gemischtes Nash-Gleichgewicht sein, jedoch p keine ESS. Die Umkehrung gilt hier also ebenfalls nicht.

Im Folgenden wird die händische Berechnung einer evolutionär stabilen gemischten Strategie vorgeführt.

Beispiel 3.9.

Nun betrachte man das Habicht-Taube-Spiel aus Tabelle 9. Um mögliche Kandidaten für eine evolutionär stabile gemischte Strategie zu finden, müssen zuerst die gemischten Nash-Gleichgewichte bestimmt werden. Welche Strategie p muss Tier 1 wählen, um Tier 2 zwischen seinen Strategien indifferent zu machen?

$$\begin{aligned} 3p + 1 \cdot (1-p) &= 5p + 0 \cdot (1-p) \\ 3p + 1 - p &= 5p \\ 1 &= 3p \\ p &= \frac{1}{3} \end{aligned}$$

Da es sich um ein symmetrisches Spiel handelt, muss Tier 2 dieselbe Strategie wählen, um Tier 1 indifferent zu machen. Das daraus resultierende Nash-Gleichgewicht lautet $(p,q) = (\frac{1}{3}, \frac{1}{3})$. Da die Strategie p so gewählt wurde, dass Tier 2 für jedes q den gleichen erwarteten Nutzen erhält, ist $N(p,p) = N(q,p)$. Deswegen kann die Bedingung 6 zu

$$N(p,q) > N(q,q)$$

vereinfacht werden. Will man nun feststellen, ob $p = \frac{1}{3}$ eine evolutionär stabile Strategie ist, genügt es, diese Bedingung zu überprüfen.

$$\begin{aligned} q + \frac{1}{3} - \frac{1}{3}q + \frac{10}{3}q &> 3q^2 + 6q - 6q^2 \\ 3q^2 - 2q + \frac{1}{3} &> 0 \\ \frac{1}{3}(9q^2 - 6q + 1) &> 0 \\ \frac{1}{3}(3q - 1)^2 &> 0 \end{aligned}$$

Die letzte Ungleichung gilt für alle $q \neq p$ und daher ist $p = \frac{1}{3}$ eine evolutionär stabile gemischte Strategie.

3.2 Bomzes Algorithmus

In [22] stellt I.M. Bomze einen Algorithmus vor, der alle evolutionär stabilen Strategien eines n-dimensionalen evolutionären Spiels findet. Es wird angenommen, dass n verschiedene reine Strategien existieren, die durch die Standardvektoren $e_1, e_2, \dots e_n$ repräsentiert werden. Die Menge aller gemischten Strategien ist somit der Standardsimplex

$$S^N = \left\{ x \in \mathbb{R}^N : \; x_i \geq 0, \text{ für alle } i, \; \sum_{i=1}^{n} x_i = 1 \right\},$$

wobei $N = \{1, 2 \dots, n\}$.

Der Zustand einer Population wird daher durch $x \in S^N$ beschrieben, wobei x_i die Häufigkeit des Auftretens der Strategie i angibt. Trifft ein Individuum mit Strategie i auf eine Strategie j, so erhält es die Auszahlung a_{ij} aus der Fitnessmatrix $A = (a_{ij})_{1 \leq i,j \leq n}$.

Weiters sei die gesamte Population so groß, dass sie als unendlich betrachtet werden kann, und die Kontrahenten werden zufällig ausgewählt.

Da im Zustand x ein Individuum mit Strategie i mit Wahrscheinlichkeit x_j auf eine Strategie j trifft, ist die erwartete Auszahlung für dieses Individuum

$$\sum_{j=1}^{n} a_{ij} x_j = (Ax)_i.$$

Die durchschnittliche erwartete Auszahlung innerhalb einer Population im Zustand x ist demnach

$$\sum_{i=1}^{n} x_i (Ax)_i = x^T A x.$$

Die Untersuchung eines Zustandes auf Stabilität in einem Spiel, in dem alle Individuen nur reine Strategien verwenden dürfen, ist äquivalent zur Überprüfung einer gemischten Strategie auf evolutionäre Stabilität. Wie auch bei I.M. Bomze wird hier nach evolutionär stabilen gemischten Strategien gesucht.

Ein Aufeinandertreffen zweier Strategien wird durch ein Bimatrix-Spiel mit symmetrischen Auszahlungen und denselben Strategien dargestellt. Definition 3.4, angewendet auf die hier geltenden Voraussetzungen, erlaubt folgenden

Satz 3.10. *Eine Strategie $p \in S^N$ ist eine ESS genau dann, wenn (i) und (ii) gelten:*

(i) $x^T A p \leq p^T A p$, *für alle* $x \in S^N$

(ii) $x^T A p = p^T A p \wedge x \neq p \Longrightarrow p^T A x > x^T A x$

Die erste Bedingung besagt, dass p beste Antwort auf sich selbst ist oder äquivalent dazu (p, p) ein symmetrisches Nash-Gleichgewicht ist. Falls eine Strategie x ebenfalls beste Antwort auf p ist, gibt Bedingung (ii) an, dass p besser gegen x abschneidet, als x auf sich selbst. Im Folgenden werden die beiden Bedingungen als Gleichgewichts- bzw. Stabilitätsbedingung bezeichnet.

3.2.1 Funktionsweise

Bevor der Pseudocode des Algorithmus angegeben werden kann, bedarf es noch einiger Vorarbeit.

Analog zu Definition 2.4 bezeichnet die Menge

$$I(x) = \{i \in N : \ x_i > 0\}$$

den Träger einer gemischten Strategie $x \in S^N$. Für die Funktionsweise des Algorithmus ist außerdem die folgende Definition nötig.

Definition 3.11 (erweiterter Träger)**.** *Der erweiterte Träger einer Strategie $p \in S^N$ ist die Menge*

$$J(p) = \{j \in N : \ (Ap)_j = p^T Ap\}.$$

Da jede Gleichgewichtsstrategie p die Ungleichung $(Ap)_j \leq p^T Ap$ für alle $j \in N$ erfüllt, ist $I(p) \subseteq J(p)$.
Der Beweis des folgenden Satzes kann in [23] (Theorem 13 (iii)) nachgelesen werden.

Satz 3.12. *Ist $x \in S^N$ eine Gleichgewichtsstrategie und $p \in S^N$ eine ESS mit $I(x) \subseteq J(p)$, dann gilt $x = p$.*

Für zwei Elemente I und I' der Menge

$$\mathcal{I}^* = \{I(p) : \ p \text{ ist eine ESS für } A\}$$

gilt weder $I \subseteq I'$ noch $I' \subseteq I$. Eine Menge, die diese Eigenschaft besitzt, wird in der Mengenlehre als *Antikette* bezeichnet. Weiters existiert für ein $I \in \mathcal{I}^*$ höchstens eine ESS p mit $I(p) = I$, was dazu führt, dass $\mathcal{I}^*$ die gesamte ESS-Struktur eines gegebenen Spiels bestimmt.

Der Algorithmus bedient sich der beiden Hilfsprogramme FINDEQ und CHECKSTAB. FINDEQ findet für eine gegebene nichtleere Menge $I \subseteq N$ eine Gleichgewichtsstrategie x mit $I(x) = I$, welche ein aussichtsreicher Kandidat für eine ESS ist, oder gibt eine negative Rückmeldung aus, falls keine solche Strategie existiert. CHECKSTAB

überprüft, ob eine gegebene Gleichgewichtsstrategie p die Stabilitätsbedingung aus Satz 3.10 erfüllt.

Algorithmus 3.13. *(Bomzes Algorithmus)*

Input: *Ein symmetrische Matrix $A \in \mathbb{R}^{n \times n}$.*

Output: *Alle evolutionär stabile Strategien.*

Initialisierung:

Suche nach reinen Gleichgewichtsstrategien e_i (das sind alle e_i für die gilt $a_{ii} = \max_{j \in N} a_{ji}$) und definiere $N' = N \backslash \{i \in N : \ e_i$ ist eine Gleichgewichtsstrategie$\}$.

Wegen Satz 3.12 erfüllt jede nichtreine ESS p die Ungleichungskette $I(p) \subseteq J(p) \subseteq N'$. Führe CHECKSTAB$(e_i)$ *für alle $i \notin N'$ aus. Speichere e_i, wenn die Antwort positiv ist und setze $\mathcal{I} = \{I \subseteq N' : \ I$ ist mehrelementig$\}$.*

Hauptphase:

Bezeichne $\mathcal{I}_{max}$ die Menge aller $I \in \mathcal{I}$, die maximal bezüglich der Mengeninklusion in $\mathcal{I}$ sind. Für jedes $I \in \mathcal{I}_{max}$ rufe FINDEQ(I) *auf und führe für gefundene Gleichgewichtsstrategien p* CHECKSTAB(p) *durch. Wenn die Rückmeldung positiv ist, speichere p.*

if *keine ESS mit Träger in $\mathcal{I}_{max}$ existieren* ***then***

$\mathcal{I}' = \mathcal{I} \backslash \mathcal{I}_{max}$

else

bezeichne die gefundenen ESS mit $p_1, \ldots, p_s$ und setze $\mathcal{I}' = \{I \in \mathcal{I} : \ I \not\subseteq J(p_i), \ 1 \leq i \leq s\} \backslash \mathcal{I}_{max}$.

Wenn $\mathcal{I}'$ leer ist, bricht der Algorithmus ab und die Liste der gespeicherten ESS ist vollständig. Ansonsten bezeichnet $\mathcal{I}_{min}$ die Menge aller $I \in \mathcal{I}'$, welche minimal bezüglich der Mengeninklusion in $\mathcal{I}'$ sind. Für jedes $I \in \mathcal{I}_{min}$ rufe FINDEQ(I) *auf.*

if *keine ESS mit Träger in $\mathcal{I}_{min}$ existieren* ***then***

$\mathcal{I}'' = \mathcal{I}' \backslash \mathcal{I}_{min}$

else

führe für gefundene Gleichgewichtsstrategien $p_i, \ 1 \leq i \leq t$ CHECKSTAB(p_i) *durch und setze $\mathcal{I}'' = \{I \in \mathcal{I}' : \ I(p_i) \not\subseteq I, \ 1 \leq i \leq t\} \backslash \mathcal{I}_{min}$.*

Wenn $\mathcal{I}''$ leer ist, bricht der Algorithmus ab und die Liste der gespeicherten ESS ist vollständig. Ansonsten wiederhole die Hauptphase indem $\mathcal{I}$ durch $\mathcal{I}''$ ersetzt wird.

Um nun den Hintergrund der Auswahl eines aussichtsreichen Kandidaten für eine ESS durch das Hilfsprogramm FINDEQ zu durchleuchten, wird der Zusammenhang der Polyedertheorie mit dem Begriff des *extremalen Gleichgewicht* betrachtet. Um dies zu verstehen, benötigt man zuerst die Definition eines *extremalen Punkts* einer Menge.

Definition 3.14 (extremaler Punkt)**.** *Ein extremaler Punkt einer konvexen Menge K ist ein Punkt $P \in K$, welcher nicht auf einer offenen Verbindungslinie zwischen zwei Punkten aus K liegt.*

Sei für eine nichtleere Menge $I \in N$

$$\mathcal{P}_1 = \left\{ \begin{bmatrix} x \\ v \end{bmatrix} \in S^N \times \mathbb{R} : \ (Ax)_i \leq v, \text{ für alle } i \in N; \ x_i = 0, \text{ für alle } i \notin I \right\}.$$

Satz 3.15. *Sei $p \in S^N$ mit $I(p) = I$.*

(a) *Ist p eine ESS, so ist p eine extremale Gleichgewichtsstrategie. Das bedeutet p ist ein extremaler Punkt der Menge der Strategien q, die ein Nash-Gleichgewicht (q, p) ergeben:*

$$p \in ex\left\{q \in S^N : \ x^T Ap \leq q^T Ap \text{ und } x^T Aq \leq p^T Aq, \text{ für alle } x \in S^N\right\}.$$

(b) *Die Strategie p ist eine extremale Gleichgewichtsstrategie genau dann, wenn $\begin{bmatrix} p \\ p^T Ap \end{bmatrix}$ eine Ecke von $\mathcal{P}_1$ ist.*

Beweis. Ein Beweis für **(a)** findet man in [23]. Für **(b)** siehe [22]. □

Das Hilfsprogramm FINDEQ (I) wird daher auf das Problem, einen Eckpunkt $\begin{bmatrix} x \\ v \end{bmatrix}$ von $\mathcal{P}_1$ mit

$$v = p^T Ap \text{ und } I(p) = I$$

zu finden, reduziert. Dieses Problem kann sehr leicht durch Lineare Programmierung gelöst werden.

Um das Hilfsprogramm CHECKSTAB genauer zu betrachten, benötigt man die folgende Notation. Für jede Index-Menge $J \subset N$ und $I \subseteq J$, sei

$$\mathbb{R}^J_+(K) = \left\{x \in \mathbb{R}^J : \ x_k \geq 0, \text{ für alle } k \in K\right\}$$

der Kegel in $\mathbb{R}^J$, der bei Vorzeichenbeschränkung einiger Koordinaten entsteht.

Definition 3.16 (strikt copositive Matrix)**.** *Sei $M \subseteq \mathbb{R}^n$ gegeben. Eine reelle Matrix B wird M-copositiv genannt, wenn*

$$y^T By > 0, \quad \text{für alle } y \in M \text{ mit } y \neq 0.$$

Satz 3.17. *Sei $p \in S^N$ eine Gleichgewichtsstrategie und wählt man für $J'(p) = J(p)\backslash\{m\}$ ein beliebiges $m \in I(p)$, dann gilt:*

(a) *Ist $J'(p) = \emptyset$, dann ist $p = e_m$ eine reine ESS.*

(b) *Ist $J'(p) \neq \emptyset$ und wählt man die Matrix $B = (b_{ij})_{i,j \in J'(p)}$, sodass*

$$b_{ij} = a_{mj} + a_{jm} + a_{im} + a_{mi} - a_{ij} - a_{ji} - 2a_{mm}$$

und setzt man $K = J(p)\backslash I(p)$, dann ist p eine ESS genau dann, wenn B eine strikt $\mathbb{R}_+^{J'(p)}(K)$-copositive Matrix ist.

Beweis. Für den Beweis wird wieder auf [23] verwiesen. □

Das Hilfsprogramm CHECKSTAB und damit der gesamte Algorithmus sind vollständig, wenn CHECKSTAB eine Funktion beinhaltet, die überprüft, ob eine gegebene symmetrische Matrix B strikt $\mathbb{R}_+^J(K)$-copositiv ist oder nicht. Es existieren mehrere Algorithmen um diese Aufgabenstellung auf allgemeinen Polyeder-Kegel zu lösen. In diesem Fall besitzt der Kegel $\mathbb{R}_+^J(K)$ jedoch eine spezielle Struktur, die es ermöglicht, den Rechenaufwand mit dem folgenden rekursiven Kriterium erheblich zu reduzieren.

Satz 3.18. *Sei $B = (b_{ij})_{i,j \in J}$ eine symmetrische Matrix und für $K \subset J$ sei*

$$J\backslash K = \{i_1, \ldots, i_r\}.$$

Setze

$$K(0) = J \text{ und } b_{jk}^{(0)} = b_{jk} \quad j,k \in K(0)$$

und definiere für $1 \leq v \leq r$

$$K(v) = K(v-1)\backslash\{i_v\}$$

und

$$b_{jk}^{(v)} = b_{i_v i_v}^{(v-1)} b_{jk}^{(v-1)} - b_{i_v j}^{(v-1)} b_{i_v k}^{(v-1)} \quad j,k \in K(v).$$

Dann ist B strikt $\mathbb{R}_+^J(K)$-copositiv genau dann, wenn

(a) $b_{i_v i_v}^{(v-1)} > 0$ *für alle* $v \in \{1, \ldots, r\}$

(b) $b^{(r)}$ *ist strikt $\mathbb{R}_+^K(K)$-copositiv (ist $K = \emptyset$ sollte diese Bedingung ignoriert werden)*

Beweis. Der Beweis findet sich in [22]. □

Üblicherweise besitzt die Menge $K = J(p)\backslash I(p)$ nur wenige Elemente, wodurch die Matrix $B^{(r)}$ niedrigdimensional ist. Die Überprüfung, ob $B^{(r)}$ strikt $\mathbb{R}_+^K(K)$-copositiv ist, kann durch das rekursive Determinanten-Kriterium überprüft werden. Nähere Informationen dazu finden sich in [24]. Je nach Aussehen von K entsteht ein größerer oder kleinerer Rechenaufwand. Der schlechteste Fall entspricht dem Aufwand eines linearen Komplementaritätsproblem oder eines linear-quadratischen Optimierungsproblems, welche jeweils NP-vollständig sind. Dieses Resultat unterstreicht die Wichtigkeit eines effizienten Suchvorgangs um unnötige Aufrufe der Funktion CHECKSTAB zu vermeiden.

Um die Arbeitsweise des Algorithmus zu verstehen, dient das

Beispiel 3.19 (5-dimensionales evolutionäres Spiel)**.**

$$A = \begin{pmatrix} 1 & 0 & 2 & 2 & 2 \\ 0 & 1 & 2 & 2 & 2 \\ 2 & 2 & 1 & 0 & 0 \\ 2 & 2 & 0 & 1 & 0 \\ 2 & 2 & 0 & 0 & 1 \end{pmatrix}$$

Da es fünf verschiedene reine Strategien gibt, ist

$$N = \{1, 2, 3, 4, 5\}.$$

Initialisierung

Man findet in diesem Spiel keine reinen Gleichgewichtsstrategien, da

$$\nexists i : \ a_{ii} = \max_{j \in N} a_{ji}$$

gilt. Daher folgt

$$N' = N\backslash\{i \in N : \ e_i \text{ ist eine Gleichgewichtsstrategie}\} = \{1, 2, 3, 4, 5\}$$

Setze $\mathcal{I} = \{I \subseteq N',\ I \text{ mehrelementig}\} = \{\ \{1,2\}, \{1,3\}, \{1,4\}, \{1,5\}, \{2,3\}, \{2,4\}, \{2,5\}, \{3,4\}, \{3,5\}, \{4,5\}, \{1,2,3\}, \{1,2,4\}, \{1,2,5\}, \{1,3,4\}, \{1,3,5\}, \{1,4,5\}, \{2,3,4\}, \{2,3,5\}, \{2,4,5\}, \{3,4,5\}, \{1,2,3,4\}, \{1,2,3,5\}, \{1,2,4,5\}, \{1,3,4,5\}, \{2,3,4,5\}, \{1,2,3,4,5\}\ \}$.

Hauptphase 1

Das maximale Element bezüglich der Mengeninklusion in $\mathcal{I}$ ist $\{1,2,3,4,5\} =: \mathcal{I}_{max}$.

FINDEQ($\{1,2,3,4,5\}$) liefert das Gleichgewicht $p = (\frac{5}{19}, \frac{5}{19}, \frac{3}{19}, \frac{3}{19}, \frac{3}{19})$.

CHECKSTAB(p) wird durchgeführt: Da

$$x^T Ap = \frac{23}{19} \cdot (x_1 + x_2 + x_3 + x_4 + x_5) = \frac{23}{19} = p^T Ap$$

für alle $x \neq p$ erfüllt ist, müsste

$$p^T Ax > x^T Ax$$

für alle $x \neq p$ gelten, damit p eine ESS ist. Dies ist allerdings nicht der Fall, da $x = (\frac{1}{4}, 0, \frac{3}{4}, 0, 0)$ zu dem Widerspruch $\frac{23}{19} > \frac{22}{16}$ führt.

Definiere $\mathcal{I}' := \mathcal{I} \backslash \mathcal{I}_{max}$.

Die minimalen Elemente bezüglich der Mengeninklusion in $\mathcal{I}'$ sind $\{\ \{1,2\}, \{1,3\}, \{1,4\}, \{1,5\}, \{2,3\}, \{2,4\}, \{2,5\}, \{3,4\}, \{3,5\}, \{4,5\}\ \} =: \mathcal{I}_{min}$.

FINDEQ(I), $I \in \mathcal{I}_{min}$ ergibt eine positive Rückmeldung für die Träger $\{1,3\}$, $\{1,4\}$, $\{1,5\}$, $\{2,3\}$, $\{2,4\}$, $\{2,5\}$, wobei jeweils beide Strategien mit der Wahrscheinlichkeit $\frac{1}{2}$ gespielt werden. Die Gleichgewichtsstrategie für den ersten Träger $\{1,3\}$ ist zum Beispiel $p_1 = (\frac{1}{2}, 0, \frac{1}{2}, 0, 0)$.

CHECKSTAB(p_i) $i = 1, \ldots 6$ ergibt, dass jede dieser Strategien auch die Stabilitätsbedingung erfüllt und man somit sechs evolutionär stabile Strategien gefunden hat.

Wähle

$$\mathcal{I}'' = \{I \in \mathcal{I}' :\ I(p_i) \nsubseteq I,\ 1 \leq i \leq 6\} \backslash \mathcal{I}_{min} = \{\{3,4,5\}\}.$$

Hauptphase 2

Setze $\mathcal{I} = \mathcal{I}''$.

$\mathcal{I}_{max}$ bezeichne wieder das maximale Element bezüglich der Mengeninklusion in $\mathcal{I}$, welches offensichtlich $\{3,4,5\}$ ist.

FINDEQ findet kein Gleichgewicht mit dem Träger $\{3,4,5\}$, woraufhin $\mathcal{I}' = \mathcal{I} \backslash \mathcal{I}_{max}$ gesetzt wird.

Da $\mathcal{I}' = \emptyset$, bricht der Algorithmus ab und liefert als Output alle sechs evolutionär stabilen Strategien des Spiels.

Obwohl die Menge aller Träger aus 31 Elementen besteht, benötigt Bomzes Algorithmus nur fünf Überprüfungen für die Initialisierung, ruft zwölfmal FINDEQ und trotz 21 Gleichgewichtsstrategien im Spiel nur siebenmal CHECKSTAB auf. Weiters sei erwähnt, dass jede reine Gleichgewichtsstrategie die Anzahl der notwendigen Überprüfungen um den Faktor zwei verringert.

3.3 Zwei einfache Algorithmen von Z. Lin

Die Ausarbeitungen dieses Unterkapitels folgen Z. Lin, der in seiner Arbeit [25] einen Algorithmus zur Bestimmung von ESS für ein 2×2 - und einen für ein $n \times n$ - Spiel vorstellt. Betrachtet man wieder ein evolutionäres Spiel mit der Auszahlungsmatrix $A \in \mathbb{R}^{n \times n}$ und bezeichnet (x, y) ein Paar gemischter Strategien. Schreibt man $W(x, y) = x^T Ay$, so lassen sich die Bedingungen für eine ESS aus Satz 3.10 ausdrücken durch

(i) $W(x, x) \geq W(y, x)$

(ii) gilt $W(x, x) = W(y, x)$ für $y \neq x$, dann folgt $W(x, y) > W(y, y)$

3.3.1 Funktionsweise für das 2×2 - Spiel

Betrachte ein 2×2 - evolutionäres Spiel mit $S^2 = \{(x_1, x_2) : \ x_1 + x_2 = 1, \ x_1, x_2 \geq 0\} = \{(x_1, 1 - x_1) : \ 0 \leq x_1 \leq 1\}$ und Auszahlungsmatrix $A \in \mathbb{R}^{2 \times 2}$.

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

Unter diesen Voraussetzungen lassen sich sehr leicht alle möglichen evolutionär stabilen Strategien bestimmen.

Algorithmus 3.20. *(2×2 - Algorithmus)*
Input: *Eine Matrix $A \in \mathbb{R}^{2 \times 2}$.*
Output: *Alle evolutionär stabile Strategien.*
(1) ***if*** $a_{11} - a_{12} - a_{21} + a_{22} = 0$ *und* $a_{11} \neq a_{21}$ *(daher* $a_{22} \neq a_{12}$*)* ***then***
  if $a_{11} > a_{21}$ *(daher* $a_{22} < a_{12}$*)* ***then***
    $(1, 0)$ *ist die einzige ESS.*
  if $a_{11} < a_{21}$ *(daher* $a_{22} > a_{12}$*)* ***then***
    $(0, 1)$ *ist die einzige ESS.*
(2) ***if*** $a_{11} - a_{12} - a_{21} + a_{22} = 0$ *und* $a_{11} = a_{21}$ *(daher* $a_{22} = a_{12}$*)* ***then***
  keine ESS existiert.
(3) ***if*** $a_{11} - a_{12} - a_{21} + a_{22} \neq 0$ *und* $\frac{a_{22} - a_{12}}{a_{11} - a_{12} - a_{21} + a_{22}} \notin [0, 1]$ ***then***
  if $a_{11} > a_{21}$ *und* $a_{22} < a_{12}$ ***then***
    $(1, 0)$ *ist die einzige ESS.*
  if $a_{11} < a_{21}$ *und* $a_{22} > a_{12}$ ***then***
    $(0, 1)$ *ist die einzige ESS.*
(4) ***if*** $a_{11} - a_{12} - a_{21} + a_{22} < 0$ *und* $\frac{a_{22} - a_{12}}{a_{11} - a_{12} - a_{21} + a_{22}} \in [0, 1]$ ***then***
  $(\frac{a_{22} - a_{12}}{a_{11} - a_{12} - a_{21} + a_{22}}, \frac{a_{11} - a_{21}}{a_{11} - a_{12} - a_{21} + a_{22}})$ *ist die einzige ESS.*

(5) ***if*** $a_{11} - a_{12} - a_{21} + a_{22} > 0$ *und* $\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}} \in [0,1]$ ***then***
if $a_{11} > a_{21}$ *und* $a_{22} > a_{12}$ ***then***
$(1,0)$ *und* $(0,1)$ *sind beide eine ESS.*
if $a_{11} > a_{21}$ *und* $a_{22} = a_{12}$ ***then***
$(1,0)$ *ist die einzige ESS.*
if $a_{11} = a_{21}$ *und* $a_{22} > a_{12}$ ***then***
$(0,1)$ *ist die einzige ESS.*

Satz 3.21. *Algorithmus 3.20 findet alle evolutionär stabilen Strategien eines* 2×2 *- evolutionären Spiels.*

Beweis. (1) Angenommen $x = (x_1, x_2)$ ist eine ESS, dann gilt für jedes $y = (y_1, y_2) \in S^2$:

$$\begin{aligned} W(x,x) - W(y,x) &= (x_1 - y_1, y_1 - x_1) \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ 1 - x_1 \end{pmatrix} \\ &= (x_1 - y_1)[(a_{11} - a_{12} - a_{21} + a_{22})x_1 + a_{12} - a_{22}] \\ &= (x_1 - y_1)(a_{12} - a_{22}) \geq 0. \end{aligned}$$

Wenn $a_{11} > a_{21}$ (daher $a_{22} < a_{12}$) gilt, erhält man $x_1 = 1$, da die obige Ungleichung für alle y gelten muss. Nach der Definition der ESS folgt, dass $(1,0)$ die einzige ESS ist. Gilt aber $a_{11} < a_{21}$ (daher $a_{22} > a_{12}$), erhält man $x_1 = 0$ und abermals nach der Definition die einzige ESS $(0,1)$.

(2) Sei wieder angenommen, dass $x = (x_1, x_2)$ eine ESS ist. Für alle $y = (y_1, y_2) \in S^2$ gilt zwar $W(x,x) - W(y,x) = 0$, wobei aber

$$\begin{aligned} W(x,y) - W(y,y) &= (x_1 - y_1, y_1 - x_1) \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} y_1 \\ 1 - y_1 \end{pmatrix} \\ &= (x_1 - y_1)[(a_{11} - a_{12} - a_{21} + a_{22})y_1 + a_{12} - a_{22}] = 0 \not> 0 \end{aligned}$$

einen Widerspruch ergibt.

(3) Angenommen $x = (x_1, x_2)$ ist eine ESS, dann gilt für jedes $y = (y_1, y_2) \in S^2$ $W(x,y) - W(y,y) = (x_1 - y_1)[(a_{11} - a_{12} - a_{21} + a_{22})y_1 + a_{12} - a_{22}x_1 + a_{12} - a_{22}] \geq 0$. Da $\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}} \notin [0,1]$ gilt, muss $(a_{11} - a_{12} - a_{21} + a_{22})y_1 + a_{12} - a_{22}x_1 + a_{12} - a_{22} \neq 0$ sein. Daher gilt entweder $x_1 = 1$ oder $x_1 = 0$.

(a) Gilt $a_{11} > a_{21}$ und $a_{22} \geq a_{12}$, dann folgt $a_{11} - a_{12} - a_{21} + a_{22} > a_{22} - a_{12} \geq 0$. Ein Widerspruch zu $\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}} \notin [0,1]$.

(b) Gilt $a_{11} > a_{21}$ und $a_{22} < a_{12}$, so ist $(1,0)$ die einzige ESS.

(c) Gilt $a_{11} = a_{21}$ und $a_{22} \neq a_{12}$, so ist $\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}} \in [0,1]$ ein Widerspruch.

(d) Gilt $a_{11} = a_{21}$ und $a_{22} = a_{12}$, erhält man den Widerspruch $a_{11}-a_{12}-a_{21}+a_{22} = 0$.

(e) Gilt $a_{11} < a_{21}$ und $a_{22} > a_{12}$, so ist $(0,1)$ die einzige ESS.

(f) Gilt $a_{11} < a_{21}$ und $a_{22} \leq a_{12}$, erhält man einen Widerspruch zu $\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}} \notin [0,1]$.

(4) Für jedes $y = (y_1, y_2) \in S^2 \neq (\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}}, \frac{a_{11}-a_{21}}{a_{11}-a_{12}-a_{21}+a_{22}})$ gilt $W(x,x) - W(y,x) = 0$ und $W(x,y)-W(y,y) = -(a_{11}-a_{12}-a_{21}+a_{22})(\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}}-y_1)^2 > 0$. Daher ist $(\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}}, \frac{a_{11}-a_{21}}{a_{11}-a_{12}-a_{21}+a_{22}})$ eine ESS.

Sei nun $z = (z_1, 1-z_1) \neq x$ ebenfalls eine ESS, dann folgt durch $W(z,z)-W(x,z) = (a_{11} - a_{12} - a_{21} + a_{22})(\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}} - y_1)^2 < 0$ ein Widerspruch.

(5) $(\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}}, \frac{a_{11}-a_{21}}{a_{11}-a_{12}-a_{21}+a_{22}})$ kann keine ESS sein, da für $y \neq x,\ y \in S^2$, $W(x,x) - W(y,x) = 0$, aber $W(x,y) - W(y,y) = -(a_{11} - a_{12} - a_{21} + a_{22}) \cdot (\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}} - y_1)^2 < 0$.

Angenommen $x = (x_1, 1-x_1)$ ist eine ESS. Für jedes $y = (y_1, y_2) \in S^2$ gilt $W(x,x) - W(y,x) = (x_1 - y_1)[(a_{11} - a_{12} - a_{21} + a_{22})x_1 + a_{12} - a_{22}] \geq 0$. Der Term $(a_{11} - a_{12} - a_{21}+a_{22})x_1+a_{12}-a_{22}$ muss ungleich Null sein, andernfalls wäre $x_1 = \frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}}$ ein Widerspruch. Daher ist entweder $x_1 = 1$ oder $x_1 = 0$.

(a) Gilt $a_{11} > a_{21}$ und $a_{22} > a_{12}$, so sind sowohl $(1,0)$ als auch $(0,1)$ evolutionär stabil.

(b) Gilt $a_{11} > a_{21}$ und $a_{22} = a_{12}$, so ist $(1,0)$ die einzige ESS.

(c) Gilt $a_{11} > a_{21}$ und $a_{22} < a_{12}$, erhält man durch $\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}} \in [0,1]$ einen Widerspruch.

(d) Gilt $a_{11} = a_{21}$ und $a_{22} > a_{12}$, so ist $(0,1)$ die einzige ESS.

(e) Gilt $a_{11} = a_{21}$ und $a_{22} \leq a_{12}$, dann führt $a_{11} - a_{12} - a_{21} + a_{22} \leq 0$ zu einem Widerspruch.

(f) Gilt $a_{11} < a_{21}$ und $a_{22} > a_{12}$, erhält man durch $\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}} \in [0,1]$ einen Widerspruch.

(g) Gilt $a_{11} < a_{21}$ und $a_{22} \leq a_{12}$, dann führt $a_{11} - a_{12} - a_{21} + a_{22} \leq a_{22} - a_{12} \leq 0$ zu einem Widerspruch.

□

3.3.2 Funktionsweise für das $n \times n$ - Spiel

Im Folgenden bezeichne $\Gamma^n = \{S^n, A^{n\times n}\}$ ein $n \times n$ - symmetrisches Zwei-Personen-Spiel. Wenn $x_i = 0$ für ein i, so erhält man das $(n-1) \times (n-1)$ - evolutionäre Spiel $\Gamma_i^{n-1} = \{S_i^{n-1}, A_i^{(n-1)\times(n-1)}\}$, wobei

$$S_i^{n-1} = \{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) : x_1 + \cdots + x_{i-1} + x_{i+1} + \cdots + x_n = 1,\ x_j \geq 0,\ j \neq i\}$$

und

$$A_i^{(n-1)\times(n-1)} = \begin{pmatrix} a_{11} & \cdots & a_{1(i-1)} & a_{1(i+1)} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{(i-1)1} & \cdots & a_{(i-1)(i-1)} & a_{(i-1)(i+1)} & \cdots & a_{(i-1)n} \\ a_{(i+1)1} & \cdots & a_{(i+1)(i-1)} & a_{(i+1)(i+1)} & \cdots & a_{(i+1)n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{n(i-1)} & a_{n(i+1)} & \cdots & a_{nn} \end{pmatrix}.$$

Γ_i^{n-1} nennt man *Teilspiel* von Γ^n.

Das nachstehende Lemma wird von Satz 3.23 benötigt.

Lemma 3.22. *Die Menge $\mathcal{E}$ aller ESS eines Spiels Γ^n ist endlich. Des Weiteren besteht $\mathcal{E}$ nur aus der Menge x, wenn x eine evolutionär stabile Strategie aus dem Inneren $int(S^n)$ von S^n ist.*

Satz 3.23. *Sei x eine ESS für ein $n \times n$ - evolutionäres Spiel $\Gamma^n = \{S^n, A^{n\times n}\}$. Wenn $x = (x_1, x_2, \ldots, x_{n-1}, 1 - \sum_{i=1}^{n-1} x_i) \in int(S^n)$, dann ist x die einzige ESS und für $i = 1, 2, \ldots, n-1$ gilt*

$$\frac{\partial W(y,x)}{\partial y_i} = \frac{\partial W(y_1, y_2, \ldots, x_1, x_2, \ldots, x_n - 1)}{\partial y_i} = 0.$$

Ist $x \in \partial(S^n)$ (d.h. x aus dem Rand von S^n), so muss $x_i = 0$ für mindestens ein i gelten. Weiters ist $\bar{x} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ eine ESS vom Teilspiel Γ_i^{n-1}.

Man nennt x den zu $\bar{x}$ *dazugehörigen Punkt in* S^n.
Mit dem soeben Erarbeiteten kann man nun den Pseudocode des Algorithmus angeben, welcher systematisch alle Spiele und dessen Teilspiele durchläuft und bei 2×2-Spielen den zuvor vorgestellten Algorithmus 3.20 anwendet. Die gefundenen ESS der kleinen Spiele werden dann im großen Spiel auf evolutionäre Stabilität überprüft.

Algorithmus 3.24. *($n \times n$ - Algorithmus)*

Input: *Eine Matrix* $A \in \mathbb{R}^{n \times n}$.

Output: *Alle evolutionär stabile Strategien.*

Schritt 1:

Sei $I = n,\ J = 0,\ C(I) = 1,\ \mathcal{E} = \emptyset$.

Schritt 2:

Betrachte das evolutionäre Spiel $\Gamma_J^I = \{S_J^I, A_J^{I \times I}\}$.

Schritt 3:

1.Fall ($I > 2$)*:*

Löse das lineare Gleichungssystem

$$\frac{\partial W_J^I(y,x)}{\partial y_i} = \frac{\partial W_J^I(y_1, y_2, \ldots, y_{n-1}, x_1, x_2, \ldots, x_{n-1})}{\partial y_i} = 0,\ i = 1, \ldots, n-1,$$

wobei $W_J^I(x,y) = yA_J^I x^T,\ x, y \in S_J^I$.

(1) ***if*** *eine eindeutige Lösung* $x_0 \in int(S_J^I)$ *existiert* ***then***

if x_0 *ist keine ESS von* Γ_J^I ***then***

setze $C(I) = J,\ I = I - 1,\ J = 1$ *und fahre mit Schritt 2 fort.*

else

if $I = n$ ***then***

$\mathcal{E} = \{x_0\}$ *und springe zu Schritt 7.*

else

if $\bar{x_0}$ *ist eine ESS von* Γ_0^n ***then***

Setze $\mathcal{E} = \mathcal{E} \cup \{\bar{x}_0\}$ *und gehe zu Schritt 4.*

else

Fahre mit Schritt 6 fort.

(2) ***if*** *eine unendliche Lösungsmenge* $Q \subset int(S_J^I)$ *existiert* ***then***

if *keine ESS* $x_0 \in Q$ *für* Γ_J^I *exisitert* ***then***

setze $C(I) = J,\ I = I - 1,\ J = 1$ *und fahre mit Schritt 2 fort.*

else

if $I = n$ ***then***

$\mathcal{E} = \{x_0\}$ *und springe zu Schritt 7.*

else

if $\bar{x_0}$ *ist eine ESS von* Γ_0^n ***then***

Setze $\mathcal{E} = \mathcal{E} \cup \{\bar{x}_0\}$ *und gehe zu Schritt 4.*

else

Fahre mit Schritt 6 fort.

(3) ***if*** *keine Lösung für das Gleichungssystem existiert* ***then***

setze $C(I) = J,\ I = I - 1,\ J = 1$ *und fahre mit Schritt 2 fort.*

2.Fall $(I = 2)$*:*

Wende Algorithmus 3.20 an. Überprüfe, ob die zu den erhaltenen ESS gehörigen Punkte evolutionär stabil sind, speichere sie in die Menge $\bar{E}_0$ *und setze* $\mathcal{E} = \mathcal{E} \cup \{\bar{E}_0\}$

if $I = n - 1$ *und* $J > I + 1$ ***then***

springe zu Schritt 7.

if $I < n - 1$ *und* $J > I + 1$ ***then***

gehe zu Schritt 4.

if $I = n - 1$ *und* $J \leq I + 1$ ***then***

gehe zu Schritt 2.

Schritt 4:

Setze $I = I + 1,\ J = C(I) + 1$

Schritt 5:

if $I = n - 1$ *und* $J > I + 1$ ***then***

springe zu Schritt 7.

if $I < n - 1$ *und* $J > I + 1$ ***then***

gehe zu Schritt 4.

if $I < n - 1$ *und* $J \leq I + 1$ ***then***

gehe zu Schritt 2.

Schritt 6:

if $J = I + 1$ ***then***

gehe zu Schritt 4.

else

setze $J = J + 1$ *und gehe zu Schritt 2.*

Schritt 7:

Der Algorithmus bricht ab und gibt $\mathcal{E}$ *als die Menge aller ESS aus.*

Die Abfolge des Algorithmus, dessen Code auf den ersten Blick vielleicht etwas verwirrend wirkt, soll nun in einem Beispiel verdeutlicht werden.

Beispiel 3.25 (4-dimensionales evolutionäres Spiel)**.**

$$A = \begin{pmatrix} 3 & -2 & 4 & -1 \\ 4 & -2 & 1 & 4 \\ -1 & 3 & 3 & -2 \\ 1 & 2 & -1 & 3 \end{pmatrix}$$

Betrachte das evolutionäre Spiel $\Gamma_0^4 = \{\{(x_1, x_2, x_3, 1 - x_1 - x_2 - x_3) : x_1, x_2, x_3 \geq 0, x_1 + x_2 + x_3 \leq 1\}\}$ Schritt 3: Wegen $I > 2$, muss das lineare Gleichungssystem

$$\frac{\partial W_0^4(y,x)}{\partial y_i} = 0, \quad i = 1, 2, 3$$

gelöst werden.

Da $W_0^4(y,x) = y_1 \cdot (4x_1 - x_2 + 5x_3 - 1) + y_2 \cdot (-6x_2 - 3x_3 + 4) + y_3 \cdot (x_1 + 5x_2 + 5x_3 - 2) + (1 - y_1 - y_2 - y_3) \cdot (-2x_1 - x_2 - 4x_3 + 3)$, besteht das LGS aus den Gleichungen

$$\begin{aligned} \frac{\partial W_0^4(y,x)}{\partial y_1} &= 6x_1 + 9x_3 - 4 = 0 \\ \frac{\partial W_0^4(y,x)}{\partial y_2} &= 2x_1 - 5x_2 + x_3 + 1 = 0 \\ \frac{\partial W_0^4(y,x)}{\partial y_3} &= 3x_1 + 6x_2 + 9x_3 - 5 = 0 \end{aligned}$$

und besitzt die eindeutige Lösung $x_1 = \frac{11}{21}$, $x_2 = \frac{9}{21}$, $x_3 = \frac{2}{21}$. Da $x = (\frac{11}{21}, \frac{9}{21}, \frac{2}{21}, -\frac{1}{21})$ nicht in S_0^4 liegt, muss dieser Punkt nicht weiter untersucht werden und es kann zu den vier Teilspielen von Γ_0^4 übergegangen werden.

Das Teilspiel Γ_1^3

$$A_1^{3\times 3} = \begin{pmatrix} -2 & 1 & 4 \\ 3 & 3 & -2 \\ 2 & -1 & 3 \end{pmatrix}$$

Das LGS setzt sich aus den Gleichungen

$$\begin{aligned} \frac{\partial W_1^3(y,x)}{\partial y_2} &= -5x_2 + x_3 + 1 = 0 \\ \frac{\partial W_1^3(y,x)}{\partial y_3} &= 6x_2 + 9x_3 - 5 = 0 \end{aligned}$$

zusammen und besitzt die eindeutige Lösung $x_1 = \frac{14}{51},\ x_2 = \frac{19}{51}$. Da $x = (\frac{14}{51}, \frac{19}{51}, \frac{18}{51}) \in int(S_1^3)$, wird dieser Punkt nun auf evolutionäre Stabilität überprüft.
Für jedes $y \neq (\frac{14}{51}, \frac{19}{51}, \frac{18}{21})$ gilt $W_1^3(x,x) - W_1^3(y,x) = 0$, aber $W_1^3(x,y) - W_1^3(y,y) = 5y_2^2 - 7y_2y_3 - 9y_3^2 - \frac{7}{51}y_2 + \frac{440}{51}y_3 - \frac{27}{17}$ ist für $y = (0,0,1)$ kleiner als Null. Daher ist $(\frac{14}{51}, \frac{19}{51}, \frac{6}{17})$ keine ESS für Γ_1^3 und somit auch nicht für Γ_0^4. Als Nächstes müssen die drei Teilspiele von Γ_1^3 betrachtet werden.

- $\Gamma_{12}^2 = \{S_{12}, A_{12}^{2\times 2}\}$, wobei

$$A_{12}^{2\times 2} = \begin{pmatrix} 3 & -2 \\ -1 & 3 \end{pmatrix}$$

  Nach Zhi Lins Algorithmus für 2×2-Spiele hat Γ_{12}^2 die zwei ESS $(1,0)$ und $(0,1)$, welche den Punkten $(0,0,1,0)$ bzw. $(0,0,0,1)$ in S_0^4 entsprechen. Bei der Definition einer evolutionär stabilen Strategie findet man aber sowohl für $(0,0,1,0)$ als auch für $(0,0,0,1)$ einen Widerspruch.

- $\Gamma_{13}^2 = \{S_{13}, A_{13}^{2\times 2}\}$, wobei

$$A_{13}^{2\times 2} = \begin{pmatrix} -2 & 4 \\ 2 & 3 \end{pmatrix}$$

  Hier wird wieder der Algorithmus für 2×2-Spiele angewendet und man erhält die ESS $(\frac{1}{5}, \frac{4}{5})$, welche dem Punkt $(0, \frac{1}{5}, 0, \frac{4}{5})$ in S_0^4 entspricht. Für jedes $y = (y_1, y_2, y_3, 1 - y_1 - y_2 - y_3),\ y \neq x$ gilt $W_0^4(x,x) - W_0^4(y,x) = 4y_1 + \frac{19}{5}y_3$. Wenn $y_1 > 0$ oder $y_3 > 0$ ist $W_0^4(x,x) - W_0^4(y,x) > 0$. Ist $y_1 = y_2 = 0$, so gilt $W_0^4(x,x) - W_0^4(y,x) = 0$, aber wegen $W_0^4(x,y) - W_0^4(y,y) = 5(\frac{1}{5} - y_2)^2 > 0$ ist $(0, \frac{1}{5}, 0, \frac{4}{5})$ eine evolutionär stabile Strategie für Γ_0^4.

$$\mathcal{E} = \left\{ \left(0, \frac{1}{5}, 0, \frac{4}{5}\right) \right\}$$

- $\Gamma_{14}^3 = \{S_{14}^2, A_{14}^{2\times 2}\}$, wobei

$$A_{13}^{2\times 2} = \begin{pmatrix} -2 & 1 \\ 3 & 3 \end{pmatrix}$$

  Dieses Spiel besitzt die einzige ESS $(0,1)$. Das dazu korrespondierende Gleichgewicht $(0,0,1,0) \in S_0^4$ wurde schon oben auf evolutionäre Stabilität überprüft.

Das Teilspiel Γ_2^3

$$A_2^{3\times 3} = \begin{pmatrix} 3 & 4 & -1 \\ -1 & 3 & -2 \\ 1 & -1 & 3 \end{pmatrix}.$$

Das LGS setzt sich aus den Gleichungen

$$\begin{aligned} \frac{\partial W_2^3(y,x)}{\partial y_1} &= 6x_1 + 9x_3 - 4 = 0 \\ \frac{\partial W_2^3(y,x)}{\partial y_3} &= 3x_1 + 9x_3 - 5 = 0 \end{aligned}$$

zusammen und besitzt die eindeutige Lösung $x_1 = -\frac{1}{3},\ x_3 = \frac{2}{3}$. Da $x = (-\frac{1}{3}, \frac{2}{3}, \frac{2}{3}) \notin int(S_2^3)$, muss dieser Punkt nicht weiter untersucht werden und es kann zu den drei Teilspielen von Γ_2^3 übergegangen werden.

- $\Gamma_{21}^2 = \Gamma_{12}^2$ wurde bereits behandelt.
- $\Gamma_{23}^2 = \{S_{23}^2, A_{23}^{2\times 2}\}$, wobei

  $$A_{12}^{2\times 2} = \begin{pmatrix} 3 & -1 \\ 1 & 3 \end{pmatrix}$$

  Der Algorithmus für 2×2-Spiele findet die zwei ESS $(1,0)$ und $(0,1)$, welche den Punkten $(1,0,0,0)$ bzw. $(0,0,0,1)$ in S_0^4 entsprechen. $(0,0,0,1)$ wurde zuvor oben überprüft und $y = (0,1,0,0) \in S_0^4$, $W_0^4(x,x) - W_0^4(y,x) = -1 < 0$ zeigt, dass $(1,0,0,0)$ keine evolutionär stabile Strategie sein kann.
- $\Gamma_{24}^2 = \{S_{24}^2, A_{24}^{2\times 2}\}$, wobei

  $$A_{24}^{2\times 2} = \begin{pmatrix} 3 & 4 \\ -1 & 3 \end{pmatrix}$$

  Dieses Spiel besitzt die einzige ESS $(1,0)$. Die ihr entsprechende Strategie $(1,0,0,0)$ wurde bereits untersucht.

Das Teilspiel Γ_3^3

$$A_3^{3\times 3} = \begin{pmatrix} 3 & -2 & -1 \\ 4 & -2 & 4 \\ 1 & 2 & 3 \end{pmatrix}$$

Das zu lösende LGS besteht aus den Gleichungen

$$\begin{aligned} \frac{\partial W_3^3(y,x)}{\partial y_1} &= 6x_1 - 4 = 0 \\ \frac{\partial W_3^3(y,x)}{\partial y_2} &= 2x_1 - 5x_2 - 1 = 0 \end{aligned}$$

und besitzt die eindeutige Lösung $x_1 = \frac{10}{15},\ x_2 = \frac{7}{15}$. Der Punkt $x = (\frac{10}{15}, \frac{7}{15}, -\frac{2}{15})$ ist nicht in $int(S_3^3)$, weshalb sofort zu den drei Teilspielen von Γ_3^3 übergegangen werden kann.

- $\Gamma_{31}^2 = \Gamma_{13}^2$ wurde bereits behandelt.
- $\Gamma_{32}^2 = \Gamma_{23}^2$ wurde ebenfalls bereits untersucht.
- $\Gamma_{34}^2 = \{S_{34}^2, A_{34}^{2\times 2}\}$, wobei

  $$A_{34}^{2\times 2} = \begin{pmatrix} 3 & -2 \\ 4 & -2 \end{pmatrix}$$

  Dieses Spiel besitzt die einzige ESS $(0,1)$. Es muss nun untersucht werden, ob die ihr entsprechende Strategie $(1,0,0,0)$ evolutionär stabil ist. Der Punkt $y = (0,0,1,0) \in S_0^4$ ergibt den Widerspruch $W_0^4(x,x) - W_0^4(y,x) = -5 < 0$, also kann $(1,0,0,0)$ keine evolutionär stabile Strategie sein.

Das Teilspiel Γ_4^3

$$A_4^{3\times 3} = \begin{pmatrix} 3 & -2 & 4 \\ 4 & -2 & 1 \\ -1 & 3 & 3 \end{pmatrix}$$

Das zu lösende LGS besteht aus den Gleichungen

$$\begin{aligned} \frac{\partial W_4^3(y,x)}{\partial y_1} &= 3x_1 - 6x_2 + 1 = 0 \\ \frac{\partial W_4^3(y,x)}{\partial y_2} &= 7x_1 - 3x_2 - 2 = 0 \end{aligned}$$

und besitzt die eindeutige Lösung $x_1 = \frac{15}{33},\ x_2 = \frac{13}{33}$. Der Punkt $x = (\frac{15}{33}, \frac{13}{33}, \frac{5}{33})$ liegt im Inneren von S_4^3 und muss daher näher betrachtet werden.
Für jedes $y = (y_1, y_2, 1 - y_1 - y_2) \in S_4^3$ ist $W_4^3(x, x) - W_4^3(y, x) = 0$ und $W_4^3(x, y) - W_4^3(y, y) = -3y_1 r + 3y_2^2 - y_1 y_2 + \frac{101}{33} y_1 - \frac{21}{11} y_2 - \frac{1}{3}$. Setzt man in diesen Ausdruck $y = (0, 0, 1) \in S_4^3$ ein, erhält man $W_4^3(x, y) - W_4^3(y, y) = -\frac{1}{3} < 0$. Daher ist auch $x = (\frac{15}{33}, \frac{13}{33}, \frac{5}{33}, 0)$ keine evolutionär stabile Strategie.

Die drei 2×2-Teilspiele sind schnell abgehandelt, da

- $\Gamma_{41}^2 = \Gamma_{14}^2$,
- $\Gamma_{42}^2 = \Gamma_{24}^2$ und
- $\Gamma_{43}^2 = \Gamma_{34}^2$

allesamt bereits behandelt wurden.

Das Spiel Γ_0^4 besitzt mit $\mathcal{E} = \left\{\left(0, \frac{1}{5}, 0, \frac{4}{5}\right)\right\}$ nur eine evolutionär stabile Strategie, welche der Algorithmus auf systematische Weise findet.

4 Algorithmen in der kooperativen Spieltheorie

Die Anfänge der kooperativen Spieltheorie gehen auf Neumann und Morgenstern [26] zurück. Die Darstellungen des folgenden Kapitels halten sich an die Ausarbeitung von K. Jain und M. Mahdian in [27].

4.1 Grundlagen

In der kooperativen Spieltheorie versuchen die Spieler günstige Koalitionen einzugehen um ihren eigenen Nutzen zu maximieren. Jeder möglichen Koalition ist durch die Koalitionsfunktion eine Menge von Auszahlungsvektoren zugeordnet. Im Unterschied zu nicht-kooperativen Spielen, wo das Hauptaugenmerk auf der Menge der möglichen Strategien der einzelnen Spieler liegt, lässt die kooperative Spieltheorie alle Aspekte des Spiels, ausgenommen den kombinatorischen Aspekt über die möglichen Koalitionen, außer Acht.

Sei $A = \{1, \ldots, n\}$ die Menge der Spieler, die versuchen zu kooperieren, $S \subseteq A$ bezeichne eine mögliche Koalition und $V(S)$ die Koalitionsfunktion. Treten alle Spieler in eine Koalition, so spricht man von der *großen Koalition*.

Definition 4.1 (kooperatives Spiel ohne transferierbaren Nutzen)**.** *Das Tupel* $\{A, V\}$ *heißt kooperatives Spiel ohne transferierbaren Nutzen (OTN-Spiel).*

Im Fall, dass die Spieler sich den erhaltenen Nutzen beliebig untereinander aufteilen können, heißt das Spiel *kooperatives Spiel mit transferierbarem Nutzen* (TN-Spiel). Hier gibt die Funktion $v : 2^A \mapsto \mathbb{R}$ für jede mögliche Koalition S einen Wert $v(S) \in \mathbb{R}$ aus. Es gilt $v(\emptyset) = 0$. Die Menge der möglichen Nutzen ist dann

$$V(S) = \left\{ x \in \mathbb{R}^S : \sum_{i \in S} x_i \leq v(S) \right\}.$$

In dem eben definierten kooperativen Spiel müssen die Werte nicht zwingenderweise nicht-negativ sein. In diesem Kapitel wird der Fall behandelt, wo alle Werte nicht-positiv sind, was dem Problem, die Kosten eines Service zwischen den Verbrauchern aufzuteilen, entspricht. Dieses *Kostenverteilungs-Problem* kann sowohl im OTN- als auch im TN-Spiel behandelt werden. Als Beispiel mit transferierbarem Nutzen betrachte man einen Internetanbieter, der durch das Erstellen eines Netzwerkes, welches einer Menge S an Verbrauchern den Internetzugang ermöglicht, die Kosten $c(S)$ trägt. Sein Ziel ist es nun, diese Kosten unter den Verbrauchern in S aufzuteilen.

Die folgende Definition gibt ein weiteres Beispiel an, welches später noch genauer behandelt wird.

Definition 4.2 (Facility-Location-Spiel)**.**

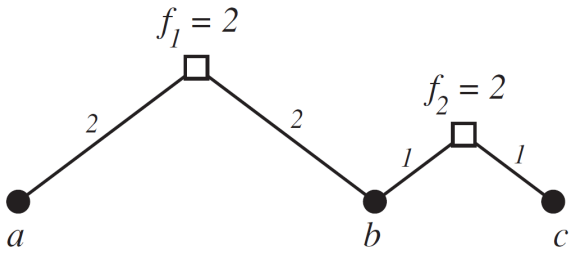

Abbildung 17: Ein Beispiel für das Facility-Location-Spiel

Gegeben sei die Menge A aller Verbraucher (oder Nachfrage-Punkte), eine Menge F an Versorgungseinrichtungen (VE), Eröffnungskosten f_i für jede VE $i \in F$ und eine Distanz d_{ij} zwischen jedem Paar (i, j) von Punkten in $A \cup F$, die die Verbindungskosten zwischen i und j darstellt. Weiters sind die Distanzen symmetrisch und sie erfüllen die Dreiecksungleichung, welche besagt, dass eine Dreiecksseite höchstens so lang wie die Summe der beiden anderen Seiten ist (Abbildung 18).

Die Kosten einer Menge $S \subseteq A$ an Verbrauchern ist definiert als

$$c(S) = \min_{F' \subseteq F} \left\{ \sum_{i \in F'} f_i + \sum_{j \in S} \min_{i \in F'} d_{ij} \right\}.$$

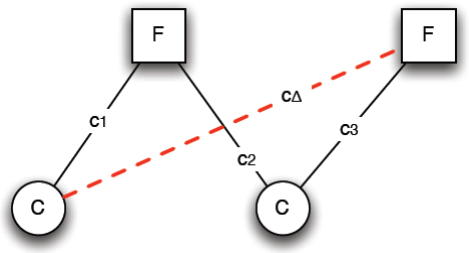

Abbildung 18: Dreiecksungleichung: es muss $c_\Delta \leq c_1 + c_2 + c_3$ gelten

Betrachtet man nun das Beispiel aus Abbildung 17, erkennt man drei Verbraucher $\{a, b, c\}$ und zwei Versorgungseinrichtungen $\{1, 2\}$. Die Eröffnungskosten für jede VE lassen sich ebenso leicht ablesen, wie ein Teil der Distanzen. Die nicht angegebenen

Distanzen erhält man, indem man die Dreiecksungleichung zur Gleichung macht. Zum Beispiel zwischen Verbraucher a und VE 2 ergibt sich: $d_{a2} = 2 + 2 + 1 = 5$. Über die Formel der Kostenfunktion $c(S)$ erhält man für jede mögliche Menge S

$$c(\{a\}) = 4, \quad c(\{b\}) = 3, \quad c(\{c\}) = 3,$$
$$c(\{a, b\}) = 6, \quad c(\{b, c\}) = 4, \quad c(\{a, c\}) = 7, c(\{a, b, c\}) = 8.$$

Im Beispiel des Facility-Location-Spiels lassen sich die monetären Kosten sehr leicht auf alle Verbraucher aufteilen. Es ist leicht einzusehen, dass es genau aus diesem Grund als TN-Spiel behandelt wird. Sieht man die längeren Verbindungen hier aber als Verzögerungen an, welche die Verbraucher tolerieren müssen, wird klar, dass die entstehenden Kosten nicht mehr so leicht aufteilbar sind. Dank dieser Tatsache ist hier das OTN-Model angebracht.

4.1.1 Der Kern

Wie in der nicht-kooperativen Spieltheorie existieren in der kooperativen Spieltheorie mehrere Lösungsansätze. Der *Kern* eines kooperativen Spiels ist einer dieser Ansätze. Grob gesprochen ist der Kern ein Ergebnis der Zusammenarbeit zwischen allen Spielern, in dem keine Koalition von Spielern durch Loslösung von der großen Koalition profitieren kann. Der Kern des Spiels entspricht also Situationen, wo es möglich ist, die Zusammenarbeit zwischen allen Akteuren in einer wirtschaftlich stabilen Art aufrecht zu erhalten.

Nun wird wieder ein Kostenverteilungs-Spiel (A, c) mit transferierbarem Nutzen betrachtet.

Definition 4.3. *Ein Vektor $\alpha \in \mathbb{R}^A$ (auch Kostenverteilung genannt) ist in dem Kern des Spiels, wenn er folgende zwei Bedingungen erfüllt:*

- *Budget-Ausgleich:* $\sum_{j \in A} \alpha_j = c(A)$.
- *Kern-Eigenschaft: für jedes* $S \subseteq A$, $\sum_{j \in S} \alpha_j \leq c(S)$.

In dem Beispiel aus Abbildung 17 sieht man sehr leicht, dass der Vektor $(4, 2, 2)$ im Kern liegt. Diese Kostenverteilung ist jedoch nicht die einzige, die im Kern enthalten ist. Der Vektor $(4, 1, 3)$ erfüllt ebenfalls alle notwendigen Bedingungen. Fügt man dem Spiel jedoch eine dritte Versorgungseinrichtung mit $f_3 = 3$ und $d_{a3} = d_{c3} = 1$ hinzu, hat das daraus entstandene Spiel einen leeren Kern. Durch diese dritte VE erhält man $c(\{a, c\}) = 5$.

Gibt es nun einen im Kern liegenden Vektor

$$\alpha = \begin{pmatrix} \alpha_a \\ \alpha_b \\ \alpha_c \end{pmatrix},$$

so muss er

$$\begin{aligned} \alpha_a + \alpha_b &\leq c(\{a,b\}) = 6 \\ \alpha_b + \alpha_c &\leq c(\{b,c\}) = 4 \\ \alpha_a + \alpha_c &\leq c(\{a,c\}) = 5 \end{aligned}$$

erfüllen. Addiert man diese drei Ungleichungen und dividiert beide Seiten durch den Faktor Zwei, so ergibt sich $\alpha_a + \alpha_b + \alpha_c \leq 7.5 \leq c(\{a,b,c\})$, ein Widerspruch zur Budget-Ausgleich-Bedingung.

Die folgende Definition ist für Satz 4.5 erforderlich.

Definition 4.4 (balancierter Gewichtvektor)**.** *Man nennt einen Vektor λ, der jedem $S \subseteq A$ einen nicht-negativen Wert λ_S zuordnet, einen balancierten Gewichtsvektor, wenn für jedes $j \in A$*

$$\sum_{S:j\in S} \lambda_S = 1$$

gilt.

Satz 4.5 (Bondareva-Shapley)**.** *Ein TN-Kostenverteilungs-Spiel (A,c) hat nichtleeren Kern genau dann, wenn*

$$\sum_{S\subseteq A} \lambda_S c(S) \geq c(A)$$

für jeden balancierten Gewichtsvektor λ gilt.

Beweis. Nach der Definition des Kerns hat das Spiel (A,c) nichtleeren Kern genau dann, wenn die Lösung des Linearen Programm (LP)

$$\begin{aligned} &\max \sum_{j\in A} \alpha_j \\ &\text{s.t } \forall S \subseteq A : \sum_{j\in A} \alpha_j \leq c(S) \end{aligned}$$

genau $c(A)$ ist. Die Lösung dieses LPs ist wiederum äquivalent zur Lösung des dualen Programms

$$\begin{aligned} &\min \sum_{S\subseteq A} \lambda_S c(S) \\ &\text{s.t } \forall j \in A : \sum_{S:j\in S} \lambda_S = 1 \\ &\qquad \text{und } \ \forall S \subseteq A : \ \lambda_S \geq 0. \end{aligned}$$

Also gilt auch, dass der Kern nichtleer ist, wenn die Lösung des dualen Programms gleich $c(A)$ ist. Da mögliche Lösungen dieses Programms balancierte Gewichtsvektoren sind, folgt, dass der Kern von (A, c) genau dann nichtleer ist, wenn für jeden solchen balancierten Gewichtsvektor

$$\sum_{S \subseteq A} \lambda_S c(S) \geq c(A)$$

gilt. □

Betrachtet man nun wieder das Facility-Location-Spiel mit der zusätzlichen dritten Versorgungseinrichtung, definiert λ als

$$\lambda_{\{a,b\}} = \lambda_{\{b,c\}} = \lambda_{\{a,c\}} = \frac{1}{2}$$

und $\lambda_S = 0$ für jede andere Menge, so ist klar, dass λ ein balancierter Gewichtsvektor ist. Außerdem gilt

$$\sum_{S \subseteq A} \lambda_S c(S) < c(A).$$

Diese Tatsache hat zur Folge, dass das Spiel (A, c) einen leeren Kern besitzt.

Ein großes Problem dieses Lösungskonzepts ist es also, dass bei vielen Spielen der Kern leer ist. Zusätzlich ist es oft computertechnisch nahezu unmöglich zu überprüfen, ob der Kern nun leer ist oder nicht, besonders bei Spielen, bei denen die Kostenfunktion schwierig zu berechnen ist. Um dieses Problem zu umgehen, dient der *γ-Kern* (in der Literatur oft γ-approximativer-Kern).

Definition 4.6. *Ein Vektor $\alpha \in \mathbb{R}^A$ ist in dem γ-Kern des Spiels (A, c), wenn er folgende zwei Bedingungen erfüllt:*

- *γ-Budget-Ausgleich: $\gamma c(A) \leq \sum_{j \in A} \alpha_j \leq c(A)$.*
- *Kern-Eigenschaft: für jedes $S \subseteq A$, $\sum_{j \in S} \alpha_j \leq c(S)$.*

Im Facility-Location-Spiel wäre der Vektor (3.5, 2.5, 1.5) im $\frac{7.5}{8}$-Kern, also in dem γ-Kern mit $\gamma = \frac{7.5}{8}$. Da $\frac{7.5}{8}$ auch noch die obere Schranke aller γ ist, für die der Kern nichtleer ist, folgt, dass für jedes $\gamma > \frac{7.5}{8}$ der γ-Kern leer ist. Sehr leicht einzusehen ist die Tatsache, dass je kleiner $|1 - \gamma|$ ist, desto besser ist die daraus resultierende Kostenverteilung.

Der Satz von Bondareva-Shapley kann nun ohne weitere Voraussetzungen auf eine approximative Version geändert werden.

Satz 4.7. *Für jedes $\gamma \leq 1$ hat ein TN-Kostenverteilungs-Spiel (A, c) nichtleeren γ-Kern genau dann, wenn*

$$\sum_{S \subseteq A} \lambda_S c(S) \geq \gamma c(A)$$

für jeden balancierten Gewichtsvektor λ gilt.

Beweis. Der Beweis verläuft analog zu dem von Satz 4.5 mit dem Unterschied, dass die Lösung des LP nur mindestens $\gamma c(S)$ sein muss. □

4.1.2 Kreuz-Monotonie

Moulin definierte in [28] die *Kreuz-Monotonie*. Um diese Eigenschaft formal zu definieren benötigt man noch die folgende

Definition 4.8 (Kostenverteilungsmethode)**.** *Sei (A, c) ein Kostenverteilungs-Spiel. Dann heißt eine Funktion $\xi : A \times 2^A \mapsto \mathbb{R}$ eine Kostenverteilungsmethode (KVM), wenn für jedes $S \subseteq A$ und jedes $i \notin S$, $\xi(i, S) = 0$ gilt.*

Weiters sagt man, eine KVM ξ ist γ-*Budget-balanciert*, wenn für jede Menge $S \subseteq A$

$$\gamma c(S) \leq \sum_{i \in S} \xi(i, S) \leq c(S)$$

gilt.

Definition 4.9 (kreuz-monoton)**.** *Eine Kostenverteilungsmethode ξ ist kreuz-monoton, wenn für alle $S, T \subseteq A$ und jedes $i \in S$,*

$$\xi(i, S) \geq \xi(i, S \cup T)$$

gilt.

Die Kreuz-Monotonie ist die Eigenschaft, die besagt, dass jeder besser dran ist, wenn sich die Menge der Verbraucher ausweitet. Oder anders gesagt, dass die Kosten, die jeder Verbraucher zu tragen hat, sich nicht erhöhen, wenn die Menge der Verbraucher größer wird.

Der folgende Satz zeigt, dass Kreuz-Monotonie eine stärkere Eigenschaft als der Kern darstellt.

Satz 4.10. *Ist ξ eine γ-Budget-balancierte Kostenverteilungsmethode von dem Spiel (A, c), dann ist $\xi(., A)$ in dem γ-Kern von diesem Spiel.*

Beweis. Zu zeigen ist, dass $\xi(., A)$ die Kern-Eigenschaft erfüllt, d.h. für jede $S \subseteq A$ gilt

$$\sum_{i \in S} \xi(i, A) \leq c(S).$$

Durch die Kreuz-Monotonie folgt $\xi(i, A) \leq \xi(i, S)$ für jedes $i \in S$ und dadurch, dass ξ γ-Budget-balanciert ist folgt $\sum_{i \in S} \xi(i, S) \leq c(S)$. Das Zusammensetzen dieser beiden Tatsachen ergibt die geforderte Ungleichung:

$$\sum_{i \in S} \xi(i, A) \leq \sum_{i \in S} \xi(i, S) \leq c(S)$$

□

4.2 Kostenverteilung über das Primal-Dual-Verfahren

Wie in dem Beweis von Satz 4.5 schon gezeigt wurde, kann eine Kostenverteilung durch das Lösen eines LPs berechnet werden. Weiters bestimmen die dualen Variablen des dualen Programms diese Kostenverteilung. Im folgenden Abschnitt wird erklärt, wie das Primal-Dual-Verfahren verwendet werden kann um eine Kostenverteilung zu berechnen, die nicht nur im approximierten Kern liegt, sondern auch die Kreuz-Monotonie erfüllt. Das Primal-Dual-Verfahren ist eine gewöhnliche Methode in dem Gebiet der Approximationsalgorithmen, wo eine nahezu optimale primale Lösung berechnet wird und die dualen Variablen (Kostenverteilung) mehr ein Nebenprodukt des Algorithmus sind.

Der Grundgedanke dieses Verfahrens ist, das Optimierungsproblem als ein mathematisches Programm zu schreiben, welches in ein LP vereinfacht werden kann. Das duale Problem von diesem LP gibt eine untere Schranke für den Wert der optimalen Lösung für das Problem. Primal-Dual-Algorithmen konstruieren gleichzeitig eine Lösung für das primale und eine Lösung für das duale Problem. Meistens werden hierfür alle dualen Variablen auf Null gesetzt und dann gleichmäßig erhöht, bis eine der Bedingungen im dualen Programm bindend wird. Danach werden die in der bindenden Beschränkung vorkommenden Variablen eingefroren und der Algorithmus fährt mit dem Erhöhen der anderen Variablen fort, bis eine komplette Lösung für das primale Problem konstruiert wurde. Die primale und duale Lösungen sollten nahe bei einander liegen, damit sie beide dem Optimum nahe kommen.

Es folgt nun ein Algorithmus für submodulare Spiele, der ohne Modifikation kreuzmonotone Kostenverteilungen errechnet, und ein Algorithmus für das Facility-Location-Spiel, der etwas schwieriger zu konstruieren ist, damit er Kreuz-Monotonie erzeugt.

4.2.1 Submodulare Spiele

Definition 4.11 (submodulares Spiel)**.** *Ein Kostenverteilungs-Spiel* (A, c) *heißt submodulares Spiel, wenn die Kostenfunktion* c

$$c(S) + c(T) \geq c(S \cup T) + c(S \cap T)$$

für alle $S, T \subseteq A$ *erfüllt.*

Submodulare Spiele werden in der Literatur auch häufig als *konkave Spiele* bezeichnet.

Betrachtet man nun ein submodulares Spiel (A, c), das primale LP

$$\begin{aligned} &\min \sum_{S \subseteq A} \lambda_S c(S) \\ &\text{s.t } \forall j \in A : \sum_{S : j \in S} \lambda_S = 1 \\ &\qquad \text{und } \ \forall S \subseteq A : \ \lambda_S \geq 0 \end{aligned}$$

und das dazugehörige duale LP

$$\begin{aligned} &\max \sum_{j \in A} \alpha_j \\ &\text{s.t } \forall S \subseteq A : \sum_{j \in A} \alpha_j \leq c(S). \end{aligned}$$

Es ist leicht einzusehen, dass bei einem submodularen Spiel die Lösung des primalen LPs immer $c(A)$ ist. Das duale LP dagegen ist nicht trivial und die optimalen Lösungen entsprechen den Kostenverteilungen im Kern. Im Folgenden sei α immer eine zulässige Lösung. Man nennt eine Menge $S \subseteq A$ *bindend*, wenn die dazugehörige Ungleichung im LP bindend ist (z.B.: wenn $\sum_{j \in S} \alpha_j = c(S)$ gilt).

Satz 4.12. *Sind zwei Mengen $S_1, S_2 \subseteq A$ bindend, dann ist auch die Menge $S_1 \cup S_2$ bindend.*

Beweis. Da S_1 und S_2 bindend sind und wegen der Submodularität folgt

$$\begin{aligned} c(S_1 \cup S_2) &\leq c(S_1) + c(S_2) - c(S_1 \cap S_2) \\ &\leq \sum_{j \in S_1} \alpha_j + \sum_{j \in S_2} \alpha_j - \sum_{j \in S_1 \cap S_2} \alpha_j \\ &= \sum_{j \in S_1 \cup S_2} \alpha_j \end{aligned}$$

Die Umkehrung gilt, da α eine zulässige Lösung ist. Es ergibt sich

$$c(S_1 \cup S_2) = \sum_{j \in S_1 \cup S_2} \alpha_j$$

und somit ist die Menge $S_1 \cup S_2$ bindend. □

Aus dieser Tatsache ergibt sich ohne viel Arbeit der folgende Satz.

Satz 4.13. *Es existiert eine maximale (bezüglich der Mengeninklusion) bindende Menge. Sie ist die Vereinigung aller bindenden Mengen.*

Beweis. Der Beweis ergibt sich direkt aus Satz 4.12. □

Der folgende Algorithmus berechnet die Kostenverteilung für ein submodulares Spiel.

Algorithmus 4.14. *(Algorithmus für submodulare Spiele)*

Input: *Ein submodulares Kostenverteilungs-Spiel (A, c) und eine Menge $T \subseteq A$ an Verbrauchern, die das Service erhalten.*

Output: *Kostenverteilung α_j für jedes $j \in T$.*

for $j \in T$ ***do***

Initialisiere $\alpha_j = 0$

Setzte $F = \emptyset$

while $T \backslash F \neq \emptyset$ ***do***

erhöhe für $j \in T\backslash F$ alle α_j mit derselben Rate bis eine neue Menge bindend wird. Setzte F als die maximale bindende Menge.

Wenn ein Element $i \in T$ einmal in der eingefrorenen Menge F enthalten ist, so bleibt es auch dort, bis der Algorithmus abbricht. Weiters lässt der Algorithmus α nie aus dem zulässigen Bereich fallen und endet erst wenn die gesamte Menge T bindend ist. Also liegt α im Kern des Spiels. Es bleibt zu zeigen, dass auch die Kreuz-Monotonie erfüllt ist.

Satz 4.15. *Die durch Algorithmus 4.14 definierte Kostenverteilungsmethode ist kreuz-monoton.*

Beweis. Sei $T_1 \subset T_2 \subseteq A$. Man lässt den Algorithmus gleichzeitig für T_1 und T_2 laufen und nennt die zwei Durchläufe T_1-Lauf und T_2-Lauf. Es ist ausreichend zu zeigen, dass zu jedem Zeitpunkt t die Menge F_1 eine Teilmenge von F_2 ist. Es gilt

$$\begin{aligned} c(F_1 \cup F_2) &\leq c(F_1) + c(F_2) - c(F_1 \cap F_2) \\ &\leq \sum_{j \in F_1} \alpha_j^1 + \sum_{j \in F_2} \alpha_j^2 - \sum_{j \in F_1 \cap F_2} \alpha_j^1 \\ &= \sum_{j \in F_1 \backslash F_2} \alpha_j^1 + \sum_{j \in F_2} \alpha_j^2 \\ &\leq \sum_{j \in F_1 \cup F_2} \alpha_j^2. \end{aligned}$$

Die erste Ungleichung folgt aus der Submodularität von c, die zweite daraus, dass beide F_i bindend sind und aus der Zulässigkeit von α^1. Die letzte resultiert aus der Tatsache, dass für jedes $i \in F_1 \backslash F_2$, solange $i \in T_1 \subset T_2$ und i nicht eingefroren in t im T_2-Lauf ist, $\alpha_j^2 = t \geq \alpha_j^1$ gilt. Daher ist die Menge $F_1 \cup F_2$ bindend. F_2 ist aber nach Definition schon die maximale bindende Menge, daher folgt mit $F_1 \subseteq F_2$ das gewünschte Ergebnis. □

4.2.2 Das Facility-Location-Spiel

Das Facility-Location-Spiel aus Definition 4.2 kann auch als ein LP geschrieben werden. In der folgenden Darstellung gibt x_i an, ob die VE i offen ist und y_{ij} gibt an, ob Verbraucher j mit VE i verbunden ist.

$$\begin{aligned}
\min\ & \sum_{i\in F} f_i x_i + \sum_{i\in F}\sum_{j\in A} d_{ij} y_{ij} \\
\text{s.t}\quad & \forall j \in A: \sum_{i\in F} y_{ij} \geq 1 \\
& \forall i \in F,\ j \in A:\ x_i \geq y_{ij} \\
& \forall i \in F,\ j \in A:\ x_i, y_{ij} \in \{0,1\}
\end{aligned}$$

Schwächt man die zweite Bedingung zu $x_i, y_{ij} \geq 0$, erhält man ein LP mit dem zugehörigen duales LP

$$\begin{aligned}
\min\ & \sum_{j\in A} \alpha_j \\
\text{s.t}\quad & \forall i \in F,\ j \in A:\ \beta_{ij} \geq \alpha_j - d_{ij} \\
& \forall i \in F: \sum_{j\in A} \beta_{ij} \leq f_i \\
& \forall i \in F,\ j \in A:\ \alpha_j, \beta_{ij} \geq 0.
\end{aligned}$$

Ohne Beschränkung der Allgemeinheit kann man annehmen, dass in einer zulässigen Lösung des oben angeführten LP $\beta_{ij} = \max(0, \alpha_j - d_{ij})$ gilt.

Im Facility-Location-Spiel muss der Algorithmus etwas modifiziert werden, da das gewöhnliche Primal-Dual-Verfahren die Kreuz-Monotonie nicht erfüllt. Es sind nun zwei Definitionen notwendig. Als die *Abgabe* von Verbraucher j an Verbraucher i bezeichnet man den Wert $\max(0, \alpha_j - d_{ij})$. Weiters heißt eine Versorgungseinrichtung i *bindend* im Bezug auf die duale Lösung α, wenn die komplette Abgabe, die i in α erhält, gleich den Eröffnungskosten ist, d.h.:

$$\sum_{j\in A} \max(0, \alpha_j - d_{ij}) = f_i$$

Die Modifikation des Algorithmus betrifft die in ihm vorkommende Variable α'_j, welche die *Geister-Kostenverteilung* (G-KV) bezeichnet. Nachdem ein Verbraucher eingefroren wurde, erhöht sich trotzdem seine G-KV. Sie zählen am Schluss nicht zu den

Endkosten des Verbrauchers, aber können den anderen Verbrauchern helfen, für die Eröffnungskosten von Versorgungseinrichtungen aufzukommen.

Algorithmus 4.16. *(Algorithmus für das Facilitiy-Location-Spiel)*

Input: *Ein Facilitiy-Location-Spiel (A, c), definiert durch Eröffnungskosten f_i und Distanzen d_{ij} und eine Menge $T \subseteq A$ von Verbraucher.*

Output: *Kostenverteilung α_j für jedes $j \in T$.*

for $j \in T$ ***do***
  Initialisiere $\alpha_j = 0$, $\alpha'_j = 0$
  Setzte $F = \emptyset$
while $T \backslash F \neq \emptyset$ ***do***
  erhöhe für $j \in T \backslash F$ *alle* α_j *und* α'_j *mit derselben Rate* ***until***
    für eine geschlossene VE i gilt: $\sum_{j \in T} \max(0, \alpha'_j - d_{ij}) = f_i$ ***then***
      öffne VE i und ergänze jedes j mit positiver Abgabe an i zu F
    für eine offene VE i und Verbraucher j gilt: $a_j = d_{ij}$ ***then***
      ergänze j zu F

Mit dieser kleinen Änderung ist es einfach zu beweisen, dass die errechnete Kostenverteilung die Kreuz-Monotonie erfüllt. Auch die Vermutung, dass durch die G-KV, die nicht zu den Endkosten zählt aber sehr wohl hilft, die VE zu öffnen, die Kostenverteilung die Budget-Ausgleichs Bedingung nicht erfüllt, kann wiederlegt werden.

Satz 4.17. *Die von Algorithmus 4.16 berechnete Kostenverteilung ist $\frac{1}{3}$-Budget-balanciert.*

Beweis. Der Beweis findet sich in [3], Kapitel 15, Seite 399-400. □

5 Konklusion

In diesem Kapitel wird eine kurze Zusammenfassung über die in der Diplomarbeit bearbeiteten Themen gegeben und zusätzlich ein paar persönliche Eindrücke, die wir während des Schreibens gewonnen haben, geschildert.

Die grundsätzliche Reihenfolge bei der Darstellung eines Algorithmus war, zuerst die benötigten Grundlagen dem Leser so einfach wie möglich zu vermitteln. Wir haben uns stets bemüht, relativ wenig an speziellem Wissen über die Spieltheorie vorauszusetzen, aber haben dennoch versucht, die Aussagen auf den Punkt zu bringen, um auch erfahrene Spieltheoretiker nicht zu langweilen. Nachdem dieses Vorwissen erarbeitet wurde, richteten wir den Fokus auf die genaue Funktionsweise des Algorithmus. Da hier von Algorithmus zu Algorithmus verschiedenste Zugänge verwendet werden, ist dies wohl der speziellste Teil und ein Schlüsselpunkt zum Verständnis eines jeden Algorithmus. Durch die Angabe des Pseudocodes sollte die genaue Abfolge der Arbeitsschritte übersichtlich zusammengefasst werden. Um genau diese Reihenfolge auf eine weniger abstrakte Weise zu verdeutlichen, wurde der Algorithmus anhand eines Beispiels vorgeführt.

Nach einem kurzen, geschichtlichen Rückblick von den Anfängen der Spieltheorie bis in die Gegenwart in Kapitel 1, wurde in Kapitel 2 das Auffinden von Nash-Gleichgewichten behandelt. Eine Auswahl an Algorithmen, die Gleichgewichte in Spielen in Normalform berechnen, wurde vorgestellt und einige Merkmale verglichen. Auch dem degenerierten Fall und Spielen in extensiver Form wurde Aufmerksamkeit geschenkt.

In Kapitel 3 widmeten wir uns der evolutionären Spieltheorie und dem Entdecken evolutionär stabiler Strategien. Der Algorithmus von Bomze findet ebenso wie die Algorithmen von Lin sämtliche evolutionär stabile Strategien eines gegebenen evolutionären Spiels.

Die kooperative Spieltheorie war Thema von Kapitel 4. Anhand von submodularen Spielen und dem Facility-Location-Spiel wurde gezeigt, wie über das Primal-Dual-Verfahren Kostenverteilungen bestimmt werden können, welche die Kreuz-Monotonie sowie die Kern-Eigenschaften erfüllen.

Das Interessante an dem Verfassen der Diplomarbeit war für uns einerseits das Kennenlernen vieler neuer Zugänge der Mathematik, wie zum Beispiel das der Polyedertheorie, und die für uns unerwarteten Verbindungen von unterschiedlichen Gebieten der Mathematik mit der Spieltheorie. Das Berechnen von Nash-Gleichgewichten über Homöomorphismen aus der Topologie ist hier wohl ein gutes Beispiel für die-

se Verbindungen. Auch die Zusammenhänge der unterschiedlichen spieltheoretischen Themen, beispielsweise die Implikationen zwischen dem Nash-Gleichgewicht und der evolutionär stabilen Strategie, haben unsere Begeisterung geweckt.

Andererseits wiederum war es sehr lehrreich, sich mit den aktuellen Arbeiten der führenden Wissenschaftler auf diesem sehr jungen Gebiet der Algorithmischen Spieltheorie auseinander zu setzen, aber ebenso erstaunlich, dass sich auch ältere Methoden, wie der Lemke-Howson-Algorithmus aus dem Jahr 1964, noch immer bewähren und daher ein sehr großes Ansehen genießen.

Abschließend sei noch bemerkt, dass das Implementieren und die numerische Untersuchung des Algorithmus von Govindan und Wilson trotz anfänglicher Schwierigkeiten, aufgrund eines für uns neuen Betriebssystems und einer relativ unbekannten Programmiersprache, sehr spannend war. Auch das Experimentieren mit unterschiedlichen Spielen und der Beobachtung ihrer Laufzeiten war aufschlussreich und faszinierend.

6 Appendix

Der Vollständigkeit halber wird hier im Anhang noch der C++ Code des Algorithmus von Govindan und Wilson angeführt. Numerische Resultate finden sich in Kapitel 2.6.

```
/* Copyright 2002 Ben Blum, Christian Shelton
 *
 * This file is part of GameTracer.
 *
 * GameTracer is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published
 * by the Free Software Foundation; either version 2 of the License,
 * or (at your option) any later version.
 *
 * GameTracer is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GameTracer; if not, write to the Free Software
 * Foundation,Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307
 * USA
 */

#include "cmatrix.h"
#include "gnm.h"
#include "gnmgame.h"

// gnm(A,g,Eq,steps,fuzz,LNMFreq,LNMMax,LambdaMin,wobble,threshold)
// ----------------------------------------------------------------
// This executes the GNM algorithm on game A.
// Interpretation of parameters:
// g: perturbation ray.
// Eq: an array of equilibria will be stored here
// steps: number of steps to take within a support cell; higher
//        values of this parameter slow GNM down, but may help it
//        avoid getting off the path.
// fuzz: a small floating point cutoff for a variety of things.
```

```
//       can probably be left at 1e-12.
// LNMFreq: a Local Newton Method subroutine will be run every
//          LNMFreq steps to decrease accumulated errors.  This
//          executes fairly quickly, so LNMFreq can be around 3.
// LNMMax: the maximum allowed iterations within the LNM algorithm.
// LambdaMin: should always be negative.  Once the trajectory
//            gets this far out, the algorithm terminates, assuming
//            that there are no more equilibria on the path.
// wobble: this is a boolean value indicating whether or not to use
//         "wobbles" of the perturbation vector to remove
//         accumulated errors.  This removes the theoretical guarantee
//         of convergence, but in practice may help keep GNM on the
//         path.
// threshold: the equilibrium error threshold for doing a wobble. If
//            wobbles are disabled, GNM will terminate if the error
//            reaches this threshold.

int GNM(gnmgame &A, cvector &g, cvector **&Eq, int steps,
        double fuzz, int LNMFreq, int LNMMax, double LambdaMin,
        int wobble, double threshold) {
  int i, // utility variables
    bestAction,
    k,
    j,
    n,
    n_hat, // player whose pure strategy next enters or leaves the
           // support
    s_hat_old=-1, // the last pure strategy to enter or leave the
                  // support
    s_hat, // the next pure strategy to enter or leave the support
    Index = 1, // index of the equilibrium we're moving towards
    numEq = 0, // number of equilibria found so far
    stepsLeft; // number of linear steps remaining until we hit the
               // boundary

  int N = A.getNumPlayers(),
    M = A.getNumActions(); // the two most important cvector sizes,
                           // stored locally for brevity
  double bestPayoff,
    det, // determinant of the jacobian
```

```
  newV, // utility variable
  lambda, // current position along the ray
  dlambda, // derivative of lambda w.r.t time
  minBound, // distance to closest change of support
  bound, // utility variable
  del, // amount of time required to reach the next support
       // boundary, assuming linear cvector field
  delta, // the actual amount of time we will step forward
         // (smaller than del)
  x0,
  ee,
  V = 0.0; // scale factor for perturbation

int s[M]; // current best responses
int B[M]; // current support

memset(B, 0, M * sizeof(int));

cmatrix DG(M,M), // jacobian of the payoff function
  R(M,M), // jacobian of the retraction operator
  I(M,M,1,1), // identity
  Dpsi, // jacobian of the cvector field
  J(M,M); // adjoint of Dpsi

cvector sigma(M), // current strategy profile
  g0(M), // original perturbation ray
  z(M), // current position in space of games
  v(M), // current cvector of payoffs for each pure strategy
  dz(M), // derivative of z w.r.t. time
  dv(M), // derivative of v w.r.t. time
  nothing(M,0),// cvector of all zeros
  err(M),
  backup(M);


// utility variables for use as intermediate values in computations
cmatrix Y1(M,M), Y2(M,M), Y3(M,M);
cvector G(N), yn1(N), ym1(M), ym2(M), ym3(M);
```

```
  // INITIALIZATION
  Eq = (cvector **)malloc(sizeof(cvector *));

  // Find the lone equilibrium of the perturbed game
  for(n = 0; n < N; n++) {
    bestPayoff = g[A.firstAction(n)];
    bestAction = A.firstAction(n);
    for(j = bestAction+1; j < A.lastAction(n); j++) {
      if(g[j] > bestPayoff) {
bestPayoff = g[j];
bestAction = j;
      }
    }
    s[n] = bestAction;
    B[bestAction] = 1;
    G[n] = bestPayoff;
  }

  // initialize sigma to be the pure strategy profile
  // that is the lone equilibrium of the perturbed game
  for(i = 0; i < M; i++)
    sigma[i] = (double)B[i];

  A.payoffMatrix(DG, sigma, fuzz);
  DG.multiply(sigma, v);
  v /= (double)(N-1);

  // Scale g until the equilibrium sigma calculated above
  // is in fact the one unique equilibrium, and set lambda
  // equal to 1

  V = 0;

  for(n = 0; n < N; n++) {
    yn1[n] = v[s[n]];
    for(i = A.firstAction(n); i < A.lastAction(n); i++) {
      if(!B[i]) {
        newV = (v[i]-yn1[n]) / (G[n]-g[i]);
        if(newV > V)
          V = newV;
```

```
      }
    }
  }

  lambda = 1.0;  // we scale g instead
  V = V+1; // a little extra padding
  g *= V;
/*
  for(n = 0; n < N; n++) {
    yn1[n] = v[s[n]]; // yn1[n] is the payoff n receives for the
                      // action we wish to make dominant
    for(i = A.firstAction(n); i < A.lastAction(n); i++) {
      if(B[i]) // if i is the action we wish to make dominant
newV = yn1[n]-G[n];
      else
newV = yn1[n]-G[n]*(v[i]-yn1[n])/(g[i]-G[n]);
      if(newV>V)
V = newV;
    }
  }

  lambda = 1.0;  // we scale g instead
  V = V+1; // a little extra padding
  for(n = 0; n < N; n++)
    for(i = A.firstAction(n); i < A.lastAction(n); i++) {
      g[i] *= (V-yn1[n])/G[n];
    }
*/
  if(N <= 2) { // ensure we don't do small steps and LNM
    LNMFreq = 0;
    steps = 1;
  }

  z = g;
  z += v;
  z += sigma;
  //  z=sigma+v+g*lambda;
```

```
A.retractJac(R,B);

// this outer while loop executes once for each support boundary
// that the path crosses.
while(1) {
  k = 0; // iteration counter; when k reaches LNMFreq, run LNM
   // within a single boundary, support unchanged

  // take the specified number of steps within these support
  // boundaries.
  for(stepsLeft = steps; stepsLeft > 0; stepsLeft--) {
    //find J = Adj psi
    J = I;
    J += DG;
    J *= R;
    J -= I;
    J.negate();
    // J = I-((I+DG)*R);
    det = J.adjoint(); // sets J = adjoint(J)

    // find derivatives of z and lambda
    J.multiply(g,dz);
    dz.negate();
     //dz = -(J*g);
    dlambda = -det;
    R.multiply(dz, ym1);
    DG.multiply(ym1,dv);
    //dv = (DG*(R*dz));
    ym1 = g;
    ym1 *= dlambda;
    dv += ym1;
    //dv += g*dlambda;

    //Calculate payoff cvector
    DG.multiply(sigma, v);
    v /=  (double)(N-1);
    ym1 = g;
    ym1 *= lambda;
    v += ym1;
    // v = DG*sigma / (double)(N-1) + g * lambda;
```

```
      //Find next action that will enter or leave the support
      //This bit pretends that z and v change linearly and calculates
      //at what point z will equal v at a certain action; this
      //indicates that the action's probability is either
      //becoming 0 or becoming positive.
      minBound = BIGFLOAT;
      for(n = 0; n < N; n++) {
for(i = A.firstAction(n); i < A.lastAction(n); i++) {
  // do not cross the same boundary we just crossed
  if(dz[i] != dv[s[n]] && s_hat_old != i) {
    bound= (z[i]-v[s[n]])/(dv[s[n]]-dz[i]);
    if(bound > 0.0) { // forward in time
      if(bound< minBound) {
minBound = bound;
s_hat = i;
n_hat = n;
      }
    }
  }
}
      }

      delta = del = minBound;

      // if the path doesn't seem to cross any more support
      // boundaries, and there is no equilibrium in sight,
      // then quit; we don't know how big a step size to take,
      // and anyway there's a good chance there are no more
      // equilibria on the path.  This could be handled
      // differently.
      if(minBound == BIGFLOAT && Index*(lambda+dlambda*delta) > 0) {
return numEq;
      }

      // each step covers 1.0/steps of the distance to the boundary
      delta = del / stepsLeft;

      // test whether lambda will become 0 in the course of this
      // step, which means there's an equilibrium there
```

```
      if(Index*(lambda+dlambda*delta) <= 0.0) {
// if there's no next support boundary, treat the equilibrium
// as the next support boundary and step up to it incrementally
if(minBound == BIGFLOAT && N > 2 && stepsLeft > 1) {
  del = -lambda / dlambda;
  delta = del / stepsLeft;
} else {
  delta -= -lambda / dlambda; // delta is now just big enough
  ym1 = dz;                   // to get us to the equilibrium
  ym1 *= (-lambda / dlambda);
  z += ym1;
  //  z += dz*delta;
  lambda = 0;
  A.retract(sigma, z);
  A.payoffMatrix(DG, sigma, fuzz);
  ee = 0.0;
  if(N > 2) { // if N=2, the graph is linear, so we are at a
    //precise equilibrium.  otherwise, refine it.
    J = DG;
    J += I;
    J *= R;
    J -= I;
    J.negate();
    //J=I-((I+DG)*R);
    det = J.adjoint();
    ee = A.LNM(z, nothing, det, J, DG, sigma, LNMMax, fuzz,
               ym1, ym2, ym3);
  }
  if(ee < fuzz) { // only save high quality equilibria;
    // this restriction could be removed.
    Eq = (cvector **)realloc(Eq, (numEq+2)*sizeof(cvector *));
    Eq[numEq] = new cvector(M);
    *(Eq[numEq++]) = sigma;
  }
  Index = -Index;
  s_hat_old = -1;
  stepsLeft++;
  continue;
}
      }
```

```
      if(del == BIGFLOAT) {
return numEq;
      }

      backup = z;

      // do the step
      ym1 = dz;
      ym1 *= delta;
      z += ym1;
      // z = z+dz*delta;
      lambda += dlambda*delta;

      // if we're sufficiently far out on the ray in the reverse
      // direction, we're probably not going back
      if(lambda < LambdaMin && Index == -1) {
return numEq;
      }
      A.retract(sigma,z);
      A.payoffMatrix(DG, sigma,fuzz);

      if(N <= 2)
break; // already at the support boundary

      DG.multiply(sigma,err);
      err /= (double)(N-1);
      g0 = g;
      g0 *= lambda;
      err += g0;
      err += sigma;
      err -= z;
      err.negate();
      ee = max(err.max(),-err.min());
      if(ee < fuzz && stepsLeft > 2) { // path is probably
                                       // near-linear;
        stepsLeft = 2;                 // step all the way
         // to boundary
k = LNMFreq - 1;               // then run LNM
      }
      if(ee > threshold) { // if we've accumulated too much
```

```
                          // error, either
if(wobble) {       // wobble or quit.
  DG.multiply(sigma, ym1);
  ym1 /= (double)(N-1);
  g = z;
  g -= sigma;
  g -= ym1;
  g /= lambda;
  // g = ((z-sigma)-((DG*sigma) / (double)(N-1)))/lambda;
} else
  return numEq;
      }

      // if we've done LNMMax repetitions, time to get back
      // on the path
      if(stepsLeft > 1 && (++k == LNMFreq)) {
A.LNM(z, g0, det, J, DG, sigma, LNMMax, fuzz,ym1,ym2,ym3);
k = 0;
      }
    } // end of for loop

    // now we've reached a support boundary

    // if a player's current best response is leaving the
    // support, we must find a new one for that player
    if(s[n_hat] == s_hat)
      for(i = A.firstAction(n_hat); i < A.lastAction(n_hat); i++)
if(B[i] && i != s_hat) {
  s[n_hat] = i;
  break;
}
    B[s_hat] = !B[s_hat];
    A.retractJac(R,B);
    s_hat_old = s_hat;
    A.retract(ym1, z);
    sigma = ym1;
    sigma.support(B);
    sigma.unfuzz(fuzz);
    A.normalizeStrategy(sigma);
    z -= ym1;
```

```
    z += sigma;
    // z = (z-x)+sigma;

    // wobble the perturbation cvector to put us back on an
    // equilibrium
    if(N > 2 && wobble) {
      A.payoffMatrix(DG, sigma, fuzz);
      DG.multiply(sigma, ym1);
      ym1 /= (double)(N-1);
      g = z;
      g -= sigma;
      g -= ym1;
      g /= lambda;
      // g = ((z-sigma)-((DG*sigma) / (double)(N-1)))/lambda;

    }
  }
}
```

Literatur

[1] L. GILES, 1910. Sun Tzu on the Art of War - The Oldest Military Treatise in the World.

[2] M. A. DIMAND & R. W. DIMAND, 1996. The History of Game Theory, Volume 1: From the Beginnings to 1945. *Routledge*.

[3] N. NISAN, T. ROUGHGARDEN, É. TARDOS & V. V. VAZIRANI, 2007. Algorithmic Game Theory. *Cambridge University Press*.

[4] J. F. NASH JR., 1951. Non-Cooperative Games. *Annals of Mathematics 54*, 286-295.

[5] B. VON STENGEL, 2007. Equilibrium Computation for Two-Player Games in Strategic and Extensive Form. *Nisan et al,* [3], Kapitel 3, 53-78.

[6] R. PORTER, E. NUDELMAN & Y. SHOHAM, 2004. Simple Search Methods for Finding a Nash Equilibrium. *Proc. of AAAI-04*, 664-669.

[7] J. DICKHAUT & T. KAPLAN, 1991. A Program for Finding Nash Equilibria. *The Mathematica Journal*, 87-93.

[8] A. MCLANNEN & J. BERG, 2002. The Asymptotic Expected Number of Nash Equilibria of Two Player Normal Form Games. *Games and Economic Behavior*, Vol.51: 264-295.

[9] C. E. LEMKE & J.J.T. HOWSON, 1964. Equilibrium Points of Bimatrix Games. *SIAM Journal on Applied Mathematics*, 12(2): 413-423.

[10] B. VON STENGEL, 2002. Computing Equilibria for Two-Person Games. *Handbook of Game Theory with Economic Applications*, Vol.3: 1723-1759.

[11] S. GOVINDAN & R. WILSON, 2001. A Global Newton Method to Compute Nash Equilibria. *Journal of Economic Theory*, Vol.110: 65-86.

[12] B. EAVES, 1972. Homotopies for the Computation of Fixed Points. *Mathematical Programming*, Vol. 3: 25-237.

[13] B. EAVES & K. SCHMEDDERS, 1999. General Equilibrium Models and Homotopy Methods. *Journal of Economic Dynamics and Control*, Vol. 23: 1249-1279.

[14] B. VON STENGEL, A. VAN DEN ELZEN & D. TALMAN, 2002. Computing Normal Form Perfect Equilibria for Extensive Two-Person Games. *Econometrica*, Vol.70, No.2: 693-715.

[15] H. W. KUHN, 1953. Extensive Games and the Problem of Information. *Contributions to the Theory of Games II, Annals of Mathematics Studies*, Vol.28: 193-216.

[16] C. E. LEMKE, 1965. Bimatrix Equilibrium Points and Mathematical Programming. *Management Science*, Vol.11: 681-689.

[17] D. KOLLER, N. MEGIDDO & B. VON STENGEL, 1996. Efficient Computation of Equilibria for Extensive Two-Person Games. *Games and Economic Behavior*, Vol.14: 247-259.

[18] J. M. SMITH, 1972. On Evolution. *Edinburgh University Press.*

[19] J. M. SMITH & G. R. PRICE, 1973. The Logic of Animal Conflict. *Nature*, Vol.246: 15-18.

[20] D. EASLEY & J. KLEINBERG, 2010. Networks, Crowds, and Markets: Reasoning about a Highly Connected World. *Cambridge University Press.*

[21] D. EASLEY & J. KLEINBERG, 2010. Evolutionary Game Theory. *Easley and Kleinberg,* [20], Kapitel 7: 225-243.

[22] I. M. BOMZE, 1992. Detecting All Evolutionarily Stable Strategies. *Journal of Optimization Theory and Applications*, Vol.75: 313-329.

[23] I. M. BOMZE, 1986. Non-Cooperative Two-Person Games in Biology: A Classification. *International Journal of Game Theory*, Vol.15: 31-57.

[24] K. P. HADELER, 1983. On Copositive Matrices. *Linear Algebra and Applications*, Vol.49: 79-89.

[25] Z. LIN, 2006. An Algorithm of Evolutionarily Stable Strategies for the Single-Population Evolutionary Game. *Journal of Computational and Applied Mathematics*, Vol.217: 157-165.

[26] J. VON NEUMANN & O. MORGENSTERN, 1944. Theory of Games and Economic Behavior. *Princeton University Press.*

[27] K. JAIN & M. MAHDIAN, 2007. Cost Sharing. *Nisan et al,* [3], Kapitel 15, 385-410.

[28] H. MOULIN, 1999. Incremental Cost Sharing: Characterization by Coalition Strategy-Proofness. *Social Choice and Welfare*, Kapitel 16, 279-320.