sowie der (zu minimierenden) **Zielfunktion**

$$\sum_{i=1}^{n} b_i x_i \text{ (für } x_i \geq 0).$$

Will man die Zielfunktion nicht minimieren, sondern maximieren, so genügt es, die Vorzeichen der Koeffizienten b_i umzukehren. Anstelle von Ungleichungen kann man auch Gleichungen wählen, da

$$x + y \leq c \text{ gdw. } x + y + z = c \text{ für ein } z \geq 0.$$

Ein solches z heißt **Slack-Variable**.

Die **Simplex-Methode** ist das Standardverfahren, um lineare Programme zu lösen, d.h. um zulässige Lösungen zu finden, die die Zielfunktion minimieren bzw. maximieren. Diese Methode hat im schlechtesten Fall exponentielle Laufzeit, ist aber in allen praktischen Fällen „gutmütig“. Daneben gibt es auch ein Polynomialzeit-Lösungsverfahren, das jedoch in der Praxis kaum zum Einsatz kommt.

4.1.2 Anwendung auf Nullsummenspiele

Seien $A_1 = \{ a_1, \ldots, a_m \}$ und $A_2 = \{ b_1, \ldots, b_n \}$.

Spieler 1 sucht eine gemischte Strategie α_1. Bestimme für jedes α_1 von Spieler 1 den Nutzen bei der schlimmsten Antwort des Gegners, dann maximiere darüber. Das entsprechende lineare Programm ist

$$\begin{aligned} \alpha_1(a_i) &\geq 0 \qquad \text{für alle } 1 \leq i \leq m \\ \sum_{i=1}^{m} \alpha_1(a_i) &= 1 \\ U_1(\alpha_1, b_j) = \sum_{i=1}^{m} \alpha_1(a_i) \cdot u_1(a_i, b_j) &\geq u \qquad \text{für alle } 1 \leq j \leq n \\ \text{Maximiere } u. \end{aligned}$$

Die Lösung dieses linearen Programms ist ein Maximinimierer für Spieler 1. Die Lösung eines analogen Programmes führt zu einem Maximinimierer für Spieler 2.

Da bei Nullsummenspielen, die ein Nash-Gleichgewicht besitzen, Maximinimierung und Minimaximierung das gleiche Ergebnis liefern, kann man auch das lineare Programm mit den Ungleichungen

$$U_1(a_i, \alpha_2) \leq u \qquad \text{für alle } 1 \leq i \leq m$$

aufstellen und u minimieren. Für α_2 entsprechend.

4.2 Finden von Nash-Gleichgewichten bei allgemeinen Zwei-Personen-Matrixspielen

Für allgemeine Spiele funktioniert die LP-Methode nicht. Benutze stattdessen Instanzen des **Linear Complementarity Problem** (LCP), bei dem zu den linearen (Un-)Gleichungen ein weiterer Typ von Bedingungen hinzukommt: mit zwei Gruppen von Variablen $X = \{ x_1, \dots, x_k \}$ und $Y = \{ y_1, \dots, y_k \}$ können für $i \in \{ 1, \dots, k \}$ Bedingungen der Form $x_i \cdot y_i = 0$ (oder äquivalent $x_i = 0 \vee y_i = 0$) formuliert werden. Im Gegensatz zu linearen Programmen gibt es keine Optimierungsbedingung.

Damit sind Nash-Gleichgewichte für beliebige Zwei-Personen-Matrixspiele beschreibbar. Sei (α, β) ein Nash-Gleichgewicht mit Nutzenprofil (u, v) in dem Spiel $\langle \{ 1, 2 \}, (A_1, A_2), (u_1, u_2) \rangle$ mit $A_1 = \{ a_1, \dots, a_m \}$ und $A_2 = \{ b_1, \dots, b_n \}$. Dann muss gelten:

$$u - U_1(a_i, \beta) \geq 0 \quad \text{für alle } i \in \{ 1, \dots, m \} \tag{4.6}$$

$$v - U_2(\alpha, b_j) \geq 0 \quad \text{für alle } j \in \{ 1, \dots, n \} \tag{4.7}$$

$$\alpha(a_i) \cdot (u - U_1(a_i, \beta)) = 0 \quad \text{für alle } i \in \{ 1, \dots, m \} \tag{4.8}$$

$$\beta(b_j) \cdot (v - U_2(\alpha, b_j)) = 0 \quad \text{für alle } j \in \{ 1, \dots, n \} \tag{4.9}$$

$$\alpha(a_i) \geq 0 \quad \text{für alle } i \in \{ 1, \dots, m \}, \qquad \sum_{i=1}^{m} \alpha(a_i) = 1 \tag{4.10}$$

$$\beta(b_j) \geq 0 \quad \text{für alle } j \in \{ 1, \dots, n \}, \qquad \sum_{j=1}^{n} \beta(b_j) = 1 \tag{4.11}$$

Beachte in den Gleichungen (4.8) und (4.9), dass etwa $\alpha(a_i) \cdot (u - U_1(a_i, \beta)) = 0$ genau dann, wenn $\alpha(a_i) = 0$ oder $u - U_1(a_i, \beta) = 0$. Einer der beiden Faktoren muss aber immer verschwinden, da

1. $\alpha(a_i) = 0$ gilt, falls a_i nicht im Support der Gleichgewichtsstrategie liegt
2. $u - U_1(a_i, \beta) = 0$ gilt, falls a_i im Support der Gleichgewichtsstrategie liegt, weil dann a_i eine beste Antwort auf β ist (vgl. auch Bedingung (4.6)).

Die obige LCP-Formulierung kann mit zusätzlichen Variablen in LCP-Normalform umgeformt werden.

Satz 9. *Ein Strategieprofil (α, β) in gemischten Strategien mit Auszahlungsprofil (u, v) ist ein Nash-Gleichgewicht gdw. es eine Lösung des LCPs (4.6)-(4.11) über (α, β) und (u, v) ist.*

Beweis. Dass jedes Nash-Gleichgewicht eine Lösung des LCPs ist, ist wegen des Support-Lemmas (Lemma 8) klar. Sei also (α, β, u, v) eine Lösung des LCPs. Wegen der Bedingungen (4.10) und (4.11) sind α und β gemischte Strategien. Wegen (4.6) ist u mindestens so groß wie die maximale Auszahlung über alle möglichen reinen Antworten, wegen (4.8) ist u genau das Maximum der Auszahlungen. Wird die reine Strategie a_i mit positiver Wahrscheinlichkeit gespielt, dann hat die Auszahlung für Spieler i als Reaktion auf die Strategie β wegen (4.8) den Wert u. Mit der Linearität des erwarteten Nutzens folgt, dass α eine beste Antwort auf β ist. Analog zeigt man, dass auch β eine beste Antwort auf α und somit (α, β) ein Nash-Gleichgewicht mit Auszahlung (u, v) ist. □

4.2.1 Lösungsalgorithmus für LCPs

Wie löst man ein LCP? Eine Möglichkeit ist der folgende naive Algorithmus.

1. Zähle alle $(2^n - 1) \cdot (2^m - 1)$ möglichen Paare von Supportmengen auf. Für jedes solche Paar aus $supp(\alpha)$ und $supp(\beta)$:
2. Konvertiere das LCP in ein lineares Programm. Dabei sind (4.6), (4.7), (4.10) und (4.11) bereits lineare Ungleichungen, Bedingungen der Form $\alpha(a_i) \cdot (u - U_1(a_i, \beta)) = 0$ werden durch eine neue lineare Gleichung ersetzt, nämlich
   - $u - U_1(a_i, \beta) = 0$, falls $a_i \in supp(\alpha)$ und
   - $\alpha(a_i) = 0$, sonst,

   entsprechend für Bedingungen der Form $\beta(b_j) \cdot (v - U_2(\alpha, b_j)) = 0$. Da die Kriterien, die optimiert werden sollen, bereits in den Constraints stehen, benötigen wir eine beliebige, von den Constraints unabhängige Optimierungsfunktion, etwa die konstante Nullfunktion.
3. Wende einen Lösungsalgorithmus für lineare Programme, zum Beispiel den Simplex-Algorithmus, auf das transformierte Programm an.

Die Laufzeit des naiven Algorithmus beträgt $\mathcal{O}(p(n + m) \cdot 2^{n+m})$, wobei p ein geeignetes Polynom ist. In der Praxis besser geeignet ist der Lemke-Howson-Algorithmus. Die Frage, ob LcpSolve in **P** liegt, ist offen, aber es liegt auf jeden Fall in **NP**, da man den naiven Algorithmus auch als nichtdeterministischen Polynomialzeitalgorithmus betrachten kann, bei dem im ersten Schritt ein Paar aus $supp(\alpha)$ und $supp(\beta)$ „geraten“, im zweiten Schritt wie oben vorgegangen und im dritten Schritt ein Polynomialzeitalgorithmus für die Lösung linearer Programme angewandt wird.