Sinnvoll einsetzbar ist der Fermat Test aufgrund der Tatsache dass er auf Carmichael-Zahlen nicht funktioniert allerdings nicht. Ein Test der diesen Defekt nicht hat ist der Miller-Rabin Test. Dieser basiert auf dem folgenden Satz

Satz 9.6. *Sei $n \geq 1$ ungerade, sei $n-1 = 2^s d$, d ungerade, sei $a \geq 1$ mit $\mathrm{ggT}(a,n) = 1$. Falls $n \in \mathbb{P}$, dann ist*

$$a^d \equiv 1 \pmod{n}$$

oder es gibt ein $r \in \{0, \ldots, s-1\}$ so dass

$$a^{2^r d} \equiv -1 \pmod{n}.$$

Ohne Beweis.

Definition 9.1. Sei $n \geq 1$ ungerade, sei $n-1 = 2^s d$, d ungerade. Dann heißt $a \in \mathbb{N}$ *Miller-Rabin Zeuge gegen die Primalität von n* falls $\mathrm{ggT}(a,n) = 1$ und $a^d \not\equiv 1 \pmod{n}$ sowie $a^{2^r d} \not\equiv -1 \pmod{n}$ für alle $r \in \{0, \ldots, s-1\}$:

Gegeben n und $a \in \{1, \ldots, n-1\}$ kann in Zeit $O(\log n)$ festgestellt werden ob a ein Miller-Rabin Zeuge gegen die Primalität von n ist. Die Bedingung $\mathrm{ggT}(a,n) = 1$ kann mit dem euklidischen Algorithmus überprüft werden. Die Berechnung von a^d modulo n kann durch die auf Binärdarstellung basierende schnelle Exponentation in Zeit $O(\log n)$ durchgeführt werden. Die Berechnung der $a^{2^r d}$ modulo n geschieht durch sukzessives Quadrieren.

Satz 9.7. *Sei $n \geq 3$ eine ungerade zusammengesetzte Zahl, dann gibt es in $\{1, \ldots, n-1\}$ höchstens $\frac{n-1}{4}$ Zahlen, die zu n teilerfremd sind aber keine Zeugen gegen die Primalität von n sind.*

Ohne Beweis.

Algorithmus 42 Miller-Rabin Primzahltest

```
Prozedur MillerRabin(n, t)
    Für i := 1, ..., t
        a := Zufall(1, n − 1)
        Falls ggT(a, n) = 1 dann
            Falls a Miller-Rabin Zeuge gegen Primalität von n ist dann
                Antworte "zusammengesetzt"
            Ende Falls
        sonst
            Antworte "zusammengesetzt"
        Ende Falls
    Ende Für
    Antworte "wahrscheinlich prim"
Ende Prozedur
```

Korollar 9.1. *Sei $n \geq 3$ ungerade und zusammengesetzt, dann ist die Wahrscheinlichkeit dass* MillerRabin(n,t) *mit "wahrscheinlich prim" antwortet höchstens 2^{-2t}.*

Ein Primzahltest wie dieser wird zur Erzeugung von großen Primzahlen wie folgt eingesetzt. Man wählt eine zufällige Zahl n der gewünschten Größenordnung (also z.B. mit 2048 Bits). Für ein geeignetes t wird dann MillerRabin(n,t) ausgeführt. Ergibt das die Antwort "wahrscheinlich prim" wird n als Primzahl betrachtet und zurückgegeben. Ergibt das die Antwort "zusammengesetzt" wiederholt man das Verfahren für eine neue Zufallszahl n. Ein Wahl von z.B. $t = 15$ garantiert bereits eine Fehlerwahrscheinlichkeit von höchstens 1 zu einer Milliarde. Das ist für die meisten praktische Anwendungen ausreichend.

9.5 Der RSA-Algorithmus

Als Anwendung großer Primzahlen wollen wir das RSA-Verschlüsselungsverfahren betrachten. Das RSA-Verfahren ist ein public-key-Verfahren, d.h. der Schlüssel einer Person X besteht aus zwei Teilen: einem öffentlichen Schlüssel der publiziert wird und zur Verschlüsselung von Nachrichten an X verwendet werden kann und einem privatem Schlüssel der im Besitz von X ist und geheim bleibt. Die Schlüsselerzeugung funktioniert wie folgt: Person X wählt zufällig zwei (vermutete) Primzahlen p und q wie im vorherigen Abschnitt beschrieben. Dann wird $n = pq$ als RSA-Modul bezeichnet. Weiters wählt X eine natürliche Zahl e mit

$$1 < e < (p-1)(q-1) \text{ und } \mathrm{ggT}(e, (p-1)(q-1)) = 1$$

und[2] berechnet eine Zahl d mit

$$1 < d < (p-1)(q-1) \text{ und } ed \equiv 1 \pmod{(p-1)(q-1)}$$

was mit dem erweiterten euklidischen Algorithmus möglich ist. Dann ist (n, e) der öffentliche Schlüssel und d der private Schlüssel.

Will nun eine Person Y eine Nachricht an X schicken so geht sie zur Verschlüsselung wie folgt vor. Zunächst nehmen wir an, dass für die Nachricht m gilt: $0 \leq m < n$. Die Nachricht m wird dann verschlüsselt zu

$$c \equiv m^e \pmod{n}.$$

Um diese Verschlüsselung durchzuführen reicht es, den öffentlichen Schlüssel (n, e) zu kennen. Die Entschlüsselung basiert dann auf dem folgenden Satz

Satz 9.8. *Sei (n, e) ein öffentlicher Schlüssel, sei d der dazugehörige private Schlüssel und sei $0 \leq m < n$. Dann gilt $(m^e)^d \equiv m \pmod{n}$.*

Um diesen Satz zu beweisen machen wir noch kurz die folgende zahlentheoretische Beobachtung.

Lemma 9.4. *Seien $n_1, n_2 \in \mathbb{Z}$ relativ prim, seien $a, b \in \mathbb{Z}$ mit $a \equiv b \pmod{n_1}$ und $a \equiv b \pmod{n_2}$, dann ist $a \equiv b \pmod{n_1 n_2}$.*

Beweis. Da $a \equiv b \pmod{n_1}$ und $a \equiv b \pmod{n_2}$ gibt es $k_1, k_2 \in \mathbb{Z}$ so dass $a - b = n_1 k_1$ und $a - b = n_2 k_2$. Also ist $n_1 k_1 = n_2 k_2$ und da n_1 und n_2 relativ prim sind gilt $n_2 \mid k_1$, d.h. also es gibt ein $l_2 \in \mathbb{Z}$ mit $k_1 = n_2 l_2$. Damit ist $a - b = n_1 n_2 l_2$, d.h. also $a \equiv b \pmod{n_1 n_2}$. □

Beweis von Satz 9.8. Da $ed \equiv 1 \pmod{(p-1)(q-1)}$ existiert ein $l \in \mathbb{Z}$ so dass $ed = 1 + l(p-1)(q-1)$ und damit

$$(m^e)^d = m^{ed} = m^{1+l(p-1)(q-1)} = m(m^{(p-1)(q-1)})^l.$$

Dann gilt

$$(m^e)^d \equiv m(m^{p-1})^{(q-1)l} \equiv m \pmod{p}$$

wegen dem kleinen Satz von Fermat falls m kein Vielfaches von p ist. Falls m ein Vielfaches von p ist, dann gilt diese Gleichung ebenfalls, da beide Seiten kongruent 0 sind. Analog dazu sieht man dass $(m^e)^d \equiv m \pmod{q}$. Aus Lemma 9.4 folgt damit $(m^e)^d \equiv m \pmod{n}$. □

[2] Das Auftreten von $(p-1)(q-1)$ erklärt sich durch den Wert der Eulerschen φ-Funktion auf n: $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1)$.

Die Sicherheit dieses Verfahrens basiert darauf, dass d aus $n = pq$ und e nicht, bzw. nur mit unrealistisch großem Aufwand berechnet werden kann. Man beachte aber dass d sehr wohl effizient aus e und $(p-1)(q-1)$ berechnet werden kann. Könnte man also n schnell genug in p und q faktorisieren, wäre das RSA-Verfahren geknackt. Je nach Sicherheitsanforderungen ist es nach aktuellem Stand empfehlenswert für n eine Zahl mit zirka 500 - 3000 Bits zu wählen. Die beiden Primzahlen p und q sollten so gewählt werden, dass sie ungefähr die selbe Größenordnung haben, aber auch nicht zu nahe beieinander sind.

Kapitel 10

Lineare Optimierung

10.1 Einführung

Viele Optimierungsprobleme bestehen darin eine bestimmte Zielfunktion unter Einhaltung gewisser Bedingungen zu maximieren oder zu minimieren. Falls sowohl die Zielfunktion affin linear ist als auch die Bedingungen affin lineare Gleichungen und Ungleichungen sind, dann spricht man von einem *linearen Optimierungsproblem*. In diesem Kapitel werden wir einige Schritte in das weitläufige Thema der linearen Optimierung machen und beginnen dazu mit einem Beispiel.

Beispiel 10.1. Eine Fabrik hat eine Maschine zur Verfügung mit der zwei verschiedene Produkte erzeugt werden können. Der Einsatz der Maschine im kommenden Monat soll geplant werden, wie immer natürlich mit dem Ziel den Gewinn zu maximieren. Dabei unterliegt die Verwendung dieser Maschine gewissen Bedingungen und Einschränkungen, die wir im folgenden beschreiben und formalisieren werden. Sei x_1 die Stückzahl des ersten Produkts und x_2 die Stückzahl des zweiten Produkts. Somit ist also $x_1 \geq 0$ und $x_2 \geq 0$.

1. Mit einer Einheit des ersten Produkts lässt sich 15€ Gewinn erzielen, mit einer Einheit des zweiten Produkts 10€. Wir wollen also

   $$15x_1 + 10x_2$$

   maximieren.

2. Im kommenden Monat stehen auf dieser Maschine 20000 Arbeitsminuten (also ca. 16h pro Arbeitstag) zur Verfügung. Die Produktion einer Einheit des ersten Produkts benötigt 20 Minuten, einer Einheit des zweiten Produkts 10 Minuten. Also

   $$20x_1 + 10x_2 \leq 20000, \text{ d.h. } 2x_1 + x_2 \leq 2000.$$

3. Ein bestehender Vertrag verpflichtet zur Produktion von mindestens 200 Stück des zweiten Produkts, also

   $$x_2 \geq 200.$$

4. Die Produktion einer Einheit des ersten Produkts erzeugt 10 Gramm CO_2, einer Einheit des zweiten Produkts 40 Gramm. Der CO_2-Ausstoß pro Monat ist auf 38 kg beschränkt. Also

   $$10x_1 + 40x_2 \leq 38000, \text{ d.h. } x_1 + 4x_2 \leq 3800$$

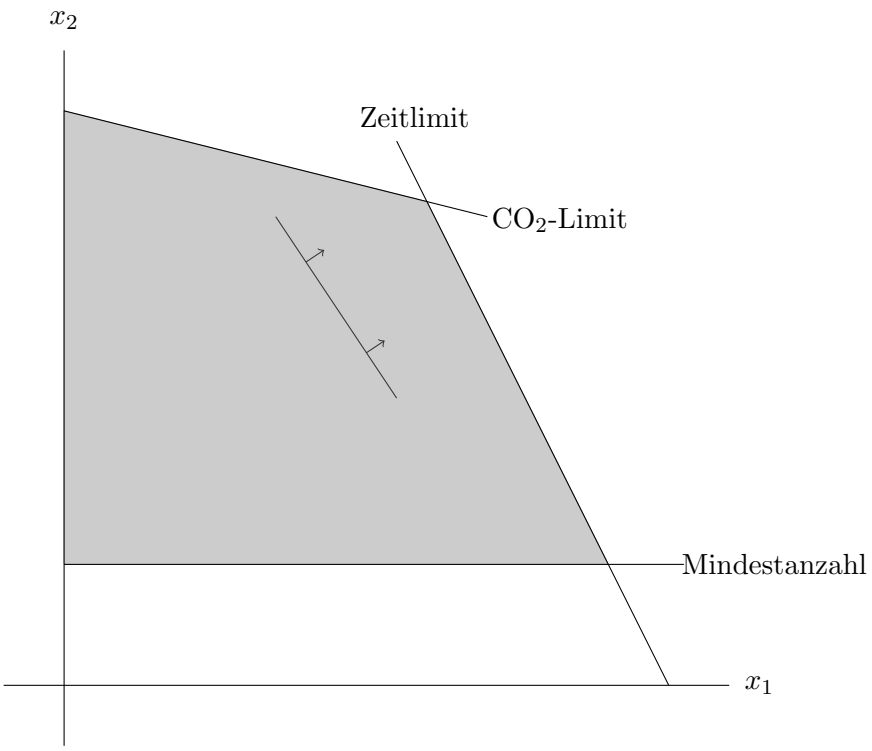

Abbildung 10.1: Beispiel für lineare Optimierung

Wir suchen also (x_1, x_2) so dass

$$\begin{aligned} 2x_1 + x_2 &\leq 2000 \\ x_2 &\geq 200 \\ x_1 + 4x_2 &\leq 3800 \end{aligned}$$

und, unter allen Paaren die diese Bedingungen erfüllen, soll $15x_1 + 10x_2$ maximal sein. Die Situation ist in Abbildung 10.1 graphisch dargestellt. Der graue Bereich besteht aus jenen (x_1, x_2) die alle Bedingungen erfüllen. Ein maximaler Punkt kann darin auf geometrische Weise wie folgt gefunden werden: Wir betrachten die Gerade $z = 15x_1 + 10x_2$ für wachsendes z und schieben sie auf diese Weise durch den zulässigen Bereich bis jede weitere Verschiebung um ein $\varepsilon > 0$ zu einem leeren Schnitt mit dem zulässigen Bereich führen würde. Hier geschieht das, wenn die Gerade den Schnittpunkt der Zeit- mit der CO_2-Gerade enthält. Dieser ist, wie man leicht anhand der Geradengleichungen berechnen kann, $(x_1, x_2) = (600, 800)$. Der maximale Gewinn wird also erreicht bei Erzeugung von 600 Einheiten des ersten Produkts und 800 Einheiten des zweiten Produkts und beträgt somit 17.000€.

Ausgehend von diesem Beispiel betrachten wir nun die allgemeine Problemstellung. Eine *affine Gleichung* ist ein Ausdruck der Form $\sum_{i=1}^{n} a_i x_i = b$ wobei $a_1, \ldots, a_n, b \in \mathbb{R}$ und $x_1, \ldots, x_n$ reellwertige Variablen sind. Eine *affine Ungleichung* ist ein Ausdruck der Form $\sum_{i=1}^{n} a_i x_i \leq b$ wobei $a_1, \ldots, a_n, b \in \mathbb{R}$ und $x_1, \ldots, x_n$ reellwertige Variablen sind. Eine *affine Funktion* $f : \mathbb{R}^n \to \mathbb{R}$ ist gegeben durch $f(x) = \sum_{i=1}^{n} a_i x_i + b$ wobei $a_1, \ldots, a_n, b \in \mathbb{R}$.

Definition 10.1. Ein *lineares Programm* in den Variablen $x_1, \ldots, x_n$ besteht aus einer endlichen Menge E von affinen Gleichungen und affinen Ungleichungen sowie einer affinen Funktion $f : \mathbb{R}^n \to \mathbb{R}$ und der Information ob f maximiert oder minimiert werden soll. Ein Vektor $x \in \mathbb{R}^n$

heißt *zulässige Lösung* falls x die Bedingungen E erfüllt. Eine zulässige Lösung x heißt *optimale Lösung* falls $f(x)$ optimal unter allen zulässigen Lösungen ist.

Die Funktion f wird auch als *Zielfunktion* bezeichnet. Für ein $x \in \mathbb{R}^n$ wird $f(x)$ auch als *Zielwert von* x bezeichnet. Zunächst kann man sich überlegen dass für ein gegebenes lineares Programm E, f einer der folgenden Fälle zutrifft.

1. E, f besitzt eine optimale Lösung mit einem Zielwert $z \in \mathbb{R}$. Das ist der interessanteste Fall der auch in Beispiel 10.1 auftritt. Man beachte, dass in diesem Fall zwar der Zielwert eindeutig ist, die optimale Lösung aber im Allgemeinen nicht.

2. E, f besitzt keine zulässigen Lösungen. In diesem Fall heißt E, f *unlösbar*. Ein Beispiel für diesen Fall erhielte man aus Beispiel 10.1 wenn (etwa aus einem weiteren Vertrag) eine Mindeststückzahl des ersten Produkts von z.B. 1000 Stück gefordert wäre.

3. E, f besitzt Lösungen mit beliebig großem (für Maximierung) bzw. beliebig kleinem (bei Minimierung) Zielwert. In diesem Fall heißt E, f *unbeschränkt*. Dieser Fall würde auftreten, wenn man in Beispiel 10.1 die limitierenden Zeit- und CO_2-Beschränkungen weglassen würde.

Wir können also das folgende Berechnungsproblem formulieren.

Lineare Optimierung
Eingabe: ein lineares Programm E, f
Ausgabe: "unbeschränkt" falls E, f unbeschränkt ist, "unlösbar" falls E, f unlösbar ist und eine optimale Lösung $x \in \mathbb{R}^n$ von E, f sonst

Es gibt polynomiale Algorithmen die dieses Problem lösen. Wir werden in Abschnitt 10.3 den Simplex-Algorithmus besprechen. Dieser ist zwar nicht polynomial, praktisch allerdings recht effizient, da für den Simplex-Algorithmus schlechte Eingaben in der Praxis selten sind. Außerdem ist der Simplex-Algorithmus in gewissem Sinn eine Verallgemeinerung des gaußschen Eliminationsverfahrens und alleine deswegen schon interessant.