Lehrstuhl für Informatik I
Optimierung unter Ressourcenbeschränkung
Universität Würzburg

Prof. Dr. Marie Schmidt
Kendra Reiter, M. Sc.
SS 2023

Einführung in IBM ILOG CPLEX Optimization Studio
Graphen und diskrete Optimierung

Inhaltsverzeichnis

1 Worum geht's?

Das IBM ILOG CPLEX Optimization Studio, im Folgenden nur 'CPLEX' genannt, ist ein weit genutztes Tool um (ganzzahlige) lineare Programme zu lösen. Es ist eins von mehreren leistungsstarken Softwarelösungen und bietet eine kostenlose Lizenz für akademische Zwecke. Wir werden im zweiten Teil der Vorlesung 'Graphen und diskrete Optimierung' mehrere lineare Programme formulieren und lösen, hierfür können natürlich auch andere (freie) LP-Solver genutzt werden.

Das Paket beinhaltet eine Entwicklungsumgebung (IDE) basierend auf Eclipse sowie die Programmiersprache OPL, mit der die linearen Programme formuliert und gelöst werden können. Folgende Anleitung gibt eine Übersicht über den Zugriff und die Formulierung mithilfe von OPL.

In diesem Dokument werden das Aufsetzen von CPLEX und die ersten Schritte zum Aufschreiben und Lösen von linearen Programmen mithilfe von CPLEX beschreiben. Dies ist keinesfalls eine vollständige Dokumentation des Tools, sondern soll als Einführung in der Vorlesung dienen. Ein (englischsprachiges) Handbuch steht Ihnen unter https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/opl_langref.pdf zur Verfügung.

2 Zugriff über den CIP-Pool

Die CIP-Pool-Rechner der Universität Würzburg verfügen über eine aktuelle Version von IBM ILOG CPLEX auf Linux. Diese kann vor Ort sowie per ssh genutzt werden, wie das funktioniert, wird hier kurz erklärt. Es ist außerdem möglich, unter der akademischen Lizenz auf https://www.ibm.com/products/ilog-cplex-optimization-studio eine eigene Variante auf dem persönlichen Rechner zu installieren, welche für die zu bearbeitenden Übungsaufgaben ausreichend ist.
Für den Zugriff vor Ort wird das Benutzerkonto des Rechenzentrums verwendet, für Studierende entspricht dies dem JMU-Account. Mehr Informationen dazu gibt es auf https://www.rz.uni-wuerzburg.de/dienste/jmu-account/wie-werde-ich-benutzer/.

2.1 Verbinden mit einem CIP-Pool-Rechner per SSH

Sie können sich sowohl auf der Linux-Kommandozeile (bash) als auch auf der Windows-Kommandozeile (cmd), sofern dort der ssh-Befehl verfügbar ist, mit einem CIP-Pool-Rechner verbinden. Sobald Sie über SSH mit einem Linux-Rechner des Instituts verbunden sind, sollte dort der standardmäßige Linux-Befehlsatz für die Kommandozeile verfügbar sein. Wie dies funktioniert, wird im Folgenden beschrieben.
Für mehr Informationen zum Verbinden mit den CIP-Pool-Rechnern des Instituts, besuchen Sie https://www.mathematik-informatik.uni-wuerzburg.de/dienste/cip-pool/raeume-zutritt-und-account/externer-linux-zugang/ oder https://www.mathematik-informatik.uni-wuerzburg.de/dienste/cip-pool/sonstiges/faq/linux/.

2.1.1 VPN-Verbindung ins Universitäts-Netzwerk

Falls Sie sich nicht im Universitäts-Netzwerk befinden, müssen Sie sich zunächst über VPN mit dem Uni-Netzwerk verbinden. Für mehr Informationen dazu, besuchen Sie https://www.rz.uni-wuerzburg.de/dienste/it-sicherheit/vpn/.

2.1.2 SSH-Verbindung zum Sprungbrettrechner

Führen Sie folgenden Befehl aus, um sich mit dem Sprungbrettrechner des Computerpools zu verbinden, wobei Sie `ss000000` durch Ihren eigenen JMU-Account („s-Nummer“) ersetzen.

```
ssh s000000@cipgate.informatik.uni-wuerzburg.de -X -Y
```

Sie werden aufgefordert, ihr zugehöriges Passwort einzugeben.

2.1.3 SSH-Verbindung zu einem CIP-Pool-Rechner

Wenn Sie sich mit dem Sprungbrettrechner des Computerpools verbunden haben,wird Ihnen eine Liste von eingeschalteten Rechnern der CIP-Pools (und wie viele Nutzer dort angemeldet sind) angezeigt. Sollte die Liste leer sein, ist gerade kein Rechner eingeschaltet. Sie können selbst einen Rechner von außen „aufwecken“, also einschalten. Hierfür müssen Sie den folgenden Befehl abschicken, wenn Sie mit dem Sprungbrettrechner verbunden sind:

```
wakeAP rechnername
```

Die Rechnernamen sind benannt als CIP-Pool-Name minus Rechnernummer. Verwenden Sie die Rechner im CIP-Pool a001, hier gibt es die Rechner 01 bis 20, also z. B.:

```
wakeAP a001-01
```

Achtung – da montags Updates auf Windows eingespielt werden, sind an diesem Tag die Rechner in a001 nicht verfügbar. Weichen Sie auf den CIP-Pool `bibsem` oder einen anderen Tag aus. Wenn Sie einen Befehl zum „Aufwecken“ ausgeführt haben, warten Sie einen Moment bis der Rechner gestartet ist. Sie können sich nun mittels SSH mit einem CIP-Pool-Rechner verbinden, wobei Sie `rechnername` durch den Namen des Rechners, z. B. `a001-01`, ersetzen. Geben Sie anschließend erneut Ihr Passwort ein.

```
ssh rechnername
```

Sobald Sie mit einem Rechner verbunden sind, können sie das Programm starten.

2.1.4 Zugriff beenden

Ihre SSH-Session können Sie Beenden, in dem Sie zwei Mal den Befehl

```
exit
```

eingeben.

2.2 Programm Starten

Die Software ist in allen CIP-Pools im Informatikgebäude unter Linux installiert. Achten Sie daher darauf, Ihren CIP-Pool-Rechner unter Linux zu starten.

Um das IBM ILOG CPLEX Optimization Studio zu öffnen, können Sie nach dem Programm `oplide` suchen bzw. im Verzeichnis `/opt/ibm/ILOG/CPLEX_Studio1262/opl/oplide/` das Programm `oplide` ausführen. Von der Kommandozeile aus kann die IDE mittels dem Befehl

```
oplide
```

gestartet werden.
Nachdem das Programm gestartet ist, werden Sie aufgefordert, einen Arbeitsbereich auszuwählen. Geben Sie einen Pfad Ihrer Wahl an oder bestätigen Sie den vorgeschlagenen Pfad.

3 Einführung in OPL

Sie sollten nun startklar sein, um Ihr erstes lineares Programm anzulegen! Dafür wird die eigene Programmiersprache *Optimization Programming Language*, kurz OPL, vom IBM benutzt. OPL kann sowohl über die Konsole sowie über die IDE von CPLEX benutzt werden.
Jede Instanz eines mathematischen Programms, welche wir mithilfe von CPLEX lösen wollen, besteht aus folgenden Bestandteilen:

1. Einer Modell-Datei (`.mod`) welche das lineare Modell, also die Zielfunktion und Nebenbedingungen, beschreibt
2. Einer Daten-Datei (`.dat`) mit allen Eingabedaten für die Instanz, die wir lösen wollen
3. Optionale Parameter für die Optimierung

3.1 Die Modell-Datei

3.1.1 Anlegen einer Modell-Datei

Um ein erstes lineares Programm zu erstellen, wollen wir zunächst ein Verzeichnis und eine `.mod`-Datei anlegen. Diese kann direkt auf dem CIP-Pool-Rechner bearbeitet werden oder alternativ zunächst lokal erstellt und dann anschließend auf dem CIP-Pool-Rechner ausgeführt werden.

OPL unterscheidet zwischen dem *Modell* (in der `.mod`-Datei) und seiner Instanz (in einer `.dat`-Datei), welches die Daten eines Problems beinhaltet. Jedes lineares Program folgt derselben Struktur, lediglich die Daten verändern sich.

Über die Konsole Wenn Sie sich im Home-Verzeichnis Ihres Rechenzentrums-Accounts befinden, legen Sie beispielsweise den Ordner `GudO23` und darin den Ordner test_lp an, in den wir das lineare Programm legen werden.

```
mkdir GudO23
cd GudO23
mkdir test_lp
cd test_lp
```

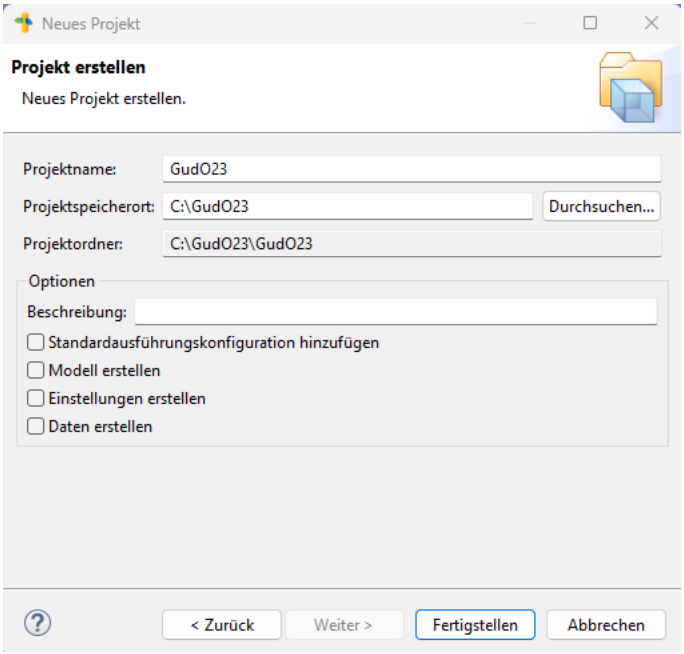
Jetzt legen wir eine leere Datei an, in die wir unser lineares Programm schreiben werden.

```
touch test.mod
```

Sie können diese Datei mit einem Textbearbeitungsprogramm Ihrer Wahl editieren. Auf der Kommandozeile bietet sich hierfür nano oder vi an. Öffnen Sie die Datei also beispielsweise mit:

```
nano test.mod
```

Über die IDE Legen Sie mittels Datei → Neu → OPL-Projekt ein neues Projekt an und vergeben Sie einen passenden Namen dafür:



Ein Modell entspricht der Beschreibung eines linearen Programms. Fügen Sie dem Projekt ein Modell hinzu mittels Datei → Neu → Modell und vergeben Sie einen Dateinamen. Modelldateien enden auf „.mod“. Öffnen Sie das neue Modell.

3.1.2 Syntax eines linearen Programms

Wir betrachten folgendes lineares Programm als Beispiel, welches wir gerne mithilfe von CPLEX lösen wollen:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & 30x_1 + 50x_2 \\ \text{subject to} \quad & 4x_1 + 11x_2 \leq 880, \\ & x_1 + x_2 \leq 150, \\ & x_2 \leq 60, \\ & x_1, x_2 \in \mathbb{R}_{\geq 0} \end{aligned} \tag{1}$$

Dies werden wir zunächst in einer Modelldatei mit der Endung `.mod` modellieren.

Variablen und Typen In OPL können Konstante- und Entscheidungsvariablen definiert werden.
Entscheidungsvariablen haben die Struktur:

```
dvar <Typ> <name>
```

Folgende Variablentypen sind verfügbar, wobei die Varianten mit '+' eine positive (≥ 0) Variable bezeichnet:

- `float`
- `float+`
- `int`
- `int+`
- `string`

Jede Zeile in OPL wird mit einem Semikolon beendet.

Für unser Beispiel (1) würden wir also zwei Variablen x_1 und x_2 wie folgt definieren:

```
dvar float+ x1;
dvar float+ x2;
```

Zielfunktion In unserem Beispiel (1) lautet die Zielfunktion $30x_1+50x_2$, welche wir maximieren wollen. In OPL schreiben wir:

```
maximize 30*x1 + 50*x2;
```

Sollten wir minimieren wollen, wird das `maximize` durch ein `minimize` ersetzt.

Nebenbedingungen Die Nebenbedingungen, welche unseren Suchraum beschreiben, werden in einem Block definiert, welcher wie folgt strukturiert ist:

```
subject to {
        erstelle Nebenbedingungen;
}
```

Für das Beispiel (1) schreiben wir also:

```
subject to {
        4*x1 + 11*x2 <= 880;
        x1 + x2 <= 150;
        x2 <= 60;
}
```

Kommentare Kommentare können wie in C oder Java mit `//` bzw. `/* */` gesetzt werden.

Zusammenfassung Insgesamt sieht unser Modell nun wie folgt aus:

Modell

```
//zwei positive nicht-ganzahlige Variablen
dvar float+ x1;
dvar float+ x2;

//Zielfunktion
maximize 30*x1 + 50*x2;
```

```
//Nebenbedingungen
subject to {
        4*x1 + 11*x2 <= 880;
        x1 + x2 <= 150;
        x2 <= 60;
}
```

3.1.3 Ausführen des Programms

Nachdem Sie Ihr erstes lineares Programm in einer `.mod`-Datei angelegt haben, können Sie dieses nun lösen.

Über die Konsole Benutzen Sie folgenden Befehl mit Ihrer `.mod`-Datei (hier `test.mod`):

```
oplrun test.mod
```

Nun sollte IBM ILOG CPLEX Ihr lineares Programm lösen und die folgende Ausgabe erzeugen:

```
<<< setup
<<< generate
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 0 columns.
Reduced LP has 2 rows, 2 columns, and 4 nonzeros.
Presolve time = 0,00 sec. (0,00 ticks)
Iteration log . . .
Iteration: 1 Scaled dual infeas = 0,000000
Iteration: 2 Dual objective = 5300,000000
<<< solve
OBJECTIVE: 5300
x1: 110
x2: 40
<<< post process
<<< done
```

Uns interessiert die Ausgabe nach `<<< solve`, also:

```
OBJECTIVE: 5300
x1: 110
x2: 40
```

Über die IDE Als Erstes muss eine Ausführungskonfiguration für das Programm angelegt werden: Rechter Mausklick auf das Modell → Zur Ausführungskonfiguration hinzufügen → Neue Ausführungskonfiguration. Der Name kann beliebig gewählt werden.
Die neue Ausführungskonfiguration sollten Sie oben links im Projektordner unter *Ausführungskonfigurationen* sehen können. Währen Sie die neue Ausführungskonfiguration mit der rechten Maustaste aus und klicken Sie auf *Ausführen*.
Im *Problembrowser*, welcher standardmäßig unten links steht, wird das Ergebnis angezeigt.

3.2 Die Daten-Datei

Für größere lineare Programme ist es empfehlenswert, das Modell und die Eingabedaten sauber in zwei Dateien zu trennen. Das hat außerdem den Vorteil, dass dasselbe lineare Programm mit unterschiedlichen Daten bzw. Instanzen gelöst werden kann.

3.2.1 Anlegen einer Daten-Datei

Eine Daten-Datei hat die Endung `.dat` und kann alle Konstanten und Eingabe-Elemente enthalten, die in der Beschreibung eines linearen Programms vorkommen.

Über die Konsole Erstellen Sie im gleichen Ordner, in dem auch Ihre Modell-Datei liegt, eine Daten Datei. Beispielweise mit

```
touch beispielDaten.dat
```

Über die IDE Fügen Sie diese einem Projekt mit Datei → Neu → Daten hinzu, und vergeben Sie eine passenden Dateinamen.

3.2.2 Syntax von Eingabedaten

Wir benutzen das Beispiel (1) von der Modell-Datei, bedienen uns aber nun an der Standardform des linearen Programms, um unsere Daten-Datei übersichtlicher zu gestalten. Zur Erinnerung: wir suchen eine zulässige Lösung $x \in \mathbb{R}^n$ für das lineare Programm der Form

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b, \\ & x \geq 0 \end{aligned} \tag{2}$$

mit geeigneter Matrix $A \in \mathbb{R}^{m \times n}$ und Vektoren $b \in \mathbb{R}^m$ und $c \in \mathbb{R}^n$. Das Beispiel (1) kann demnach äquivalent mit

$$A = \begin{pmatrix} 4 & 11 \\ 1 & 1 \\ 0 & 1 \end{pmatrix},\ b = \begin{pmatrix} 880 \\ 150 \\ 60 \end{pmatrix},\ c = \begin{pmatrix} 30 \\ 50 \end{pmatrix}$$

aufgeschrieben werden.

Arrays Variablen können in OPL als *Arrays* eingegeben werden, welche wie in anderen Programmiersprachen auch eine feste Länge besitzen. Beim späteren Einlesen von Daten ins Modell müssen wir diese Länge kennen. Um dies zu vereinfachen, können wir die erwartete Länge in einer Konstanten speichern.
Hierfür verwenden wir Arrays, die jeweils alle Konstante Eingabedaten für unser Programm beschreiben. Diese lassen sich in erwarteter Weise schachteln, um Matrizen darzustellen. Um unseren Einlesevorgang zu erleichtern, definieren wir uns zwei Konstanten: einmal für die Anzahl an Variablen (`anzahlVars`) und einmal für die Anzahl an Nebenbedingungen (`anzahlCons`). Das geht wie folgt:

```
anzahlVars = 2;
anzahlCons = 3;
```

Außerdem fügen wir noch drei Arrays für unsere Matrix A und Vektoren b und c hinzu. Die vollständige Daten-Datei sieht dann so aus:

Daten

```
// Anzahl an Variablen
anzahlVars = 2;
//Anzahl an Nebenbedingungen
anzahlCons = 3;

//Daten
A = [[4, 11], [1, 1], [0, 1]];
b = [880, 150, 60];
c = [30, 50];
```

Erstellen und speichern Sie sich eine entsprechende Daten-Datei und fügen Sie diese Ihrer Ausführungskonfiguration hinzu.

3.2.3 Laden einer Daten-Datei

Um das Modell nun zusammen mit der Daten-Datei auszuführen, müssen die Variablen aus der Daten-Datei geladen werden. Die Syntax für einfache einzelne Variablen lautet:

```
<typ> <name> = ...;
```

In den Daten wurde für unsere Variablen kein Typ spezifiziert, dies geschieht im Modell. Sie müssen also selbstständig darauf achten, dass die Typen der Werte, die Sie laden wollen, auch tatsächlich den vorliegenden Werten der Datei entsprechen. Um den Variablen ihre richtigen Daten zuzuweisen, werden die Namen der Variablen in beiden Dateien verglichen. Unsere beiden Hilfsvariablen laden wir also in der Modell-Datei mit dem Code:

```
int anzahlVars = ...;
int anzahlCons = ...;
```

Um Arrays zu laden ergänzt sich die obige Syntax um ein Paar eckige Klammern je Dimension des Arrays. Anders als in den meisten anderen Programmiersprachen beginnen Arrays in OPL bei Index 1. Um die Indexbereiche von Arrays zu definieren, benutzt OPL Ranges (Intervalle) der Form `1..n` - in unserem Fall also `1..anzahlVars` und `1..anzahlCons`. In unserem Beispiel für die drei Arrays A, b und c sieht das also wie folgt aus:

```
int A[1..anzahlCons][1..anzahlVars] = ...;
int b[1..anzahlVars] = ...;
int c[1..anzahlCons] = ...;
```

Damit lässt sich nun das alte Modell unter Verwendung dieser Variablen wie folgt umschreiben. Legen Sie hierfür wieder eine Modell-Datei an:

Modell

```
//Konstante aus der Daten-Datei
int anzahlVars = ...;
int anzahlCons = ...;

//Arrays aus der Daten-Datei
int A[1..anzahlCons][1..anzahlVars] = ...;
int b[1..anzahlVars] = ...;
int c[1..anzahlCons] = ...;

//Optimierungsvariable
dvar float+ x[1..anzahlVars];
```

Achten Sie darauf, dass in ihren Daten keine Variablen definiert sind, die nicht auch vom Modell gelesen werden wollen. CPLEX geht davon aus, dass alle Daten relevant sind und wird eine Fehlermeldung ausgeben, wenn Teile der Daten nicht auf Variablen des Modells zugewiesen werden können.

Summen Um die Nebenbedingungen aus (2) aufzuschreiben, brauchen wir noch eine Summenfunktion. Die bekannte Schreibweise für Summen

$$\sum_{i=1}^{n} a_n$$

lässt sich mit Hilfe von Ranges oder Mengen (siehe 3.4) sehr einfach in OPL darstellen:

```
sum (i in 1..n) a[i];
```

Damit lassen sich die Nebenbedingungen aus (2) wie folgt in OPL definieren:

```
subject to {
        sum(i in 1..anzahlVars) (A[1][i] * x[i]) <= b[1];
        sum(i in 1..anzahlVars) (A[2][i] * x[i]) <= b[2];
```

```
        sum(i in 1..anzahlVars) (A[3][i] * x[i]) <= b[3];
}
```

3.2.4 Ausführen des Programms

Durch das Anlegen mehrerer Daten Dateien lassen sich einfach zahlreiche Läufe des gleichen Modells jedoch mit unterschiedlichen Eingaben bewerkstelligen.

Über die Konsole Analog zum Ausführen eines Modells ohne Daten-Datei können Sie mittels

```
oplrun mein-modell.mod meine-daten.dat
```

Ihr Modell mit spezifischen Daten ausführen.

Über die IDE CPLEX durchsucht selbstständig die Ausführungskonfiguration nach Daten-Dateien. Hierfür müssen Sie also keine weiteren Anweisungen befolgen.

3.3 Zusammenfassung

Ihr lineares Programm besteht nun aus zwei Dateien: einer `.dat` Datei mit Ihren Konstanten und Eingabedaten, und einer `.mod` Datei mit Ihrem Modell.

Daten

```
// Anzahl an Variablen
anzahlVars = 2;
//Anzahl an Nebenbedingungen
anzahlCons = 3;

//Daten
A = [[4, 11], [1, 1], [0, 1]];
b = [880, 150, 60];
c = [30, 50];
```

Modell

```
//Konstante aus der Daten-Datei
int anzahlVars = ...;
int anzahlCons = ...;

//Arrays aus der Daten-Datei
int A[1..anzahlCons][1..anzahlVars] = ...;
int b[1..anzahlVars] = ...;
int c[1..anzahlCons] = ...;

//Optimierungsvariable
dvar float+ x[1..anzahlVars];

//Zielfunktion
maximize sum(i in 1..anzahlVars) c[i] * x[i];

//Nebenbedingungen
subject to {
        sum(i in 1..anzahlVars) (A[1][i] * x[i]) <= b[1];
        sum(i in 1..anzahlVars) (A[2][i] * x[i]) <= b[2];
        sum(i in 1..anzahlVars) (A[3][i] * x[i]) <= b[3];
}
```

Vor allem für größere Programme kann hilft diese Aufteilung, um eine übersichtliche und gut lesbaren Dateistruktur zu führen.

3.4 Weitere Datentypen und -strukturen

Ähnlich zu C und Java können verschiedenste Funktionen benutzt werden, um mathematische Programme in OPL zu schreiben. In diesem Abschnitt ist eine kleine Auswahl beschrieben, mehr Informationen gibt es in der offiziellen Dokumentation von IBM unter https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/opl_langref.pdf. Dieses Handbuch deckt deutlich mehr Inhalte und Funktionen ab, als in dieser Vorlesung benötigt werden, eignet sich jedoch, um einzelne Funktionen und Feinheiten von CPLEX nachzuschlagen

3.4.1 Vergleichsoperationen und logische Operatoren

Die typischen arithmetischen, vergleichenden und logischen Operatoren sind identisch mit denen in anderen Sprachen wie Java, also z. B.:

x gleich y	`x == y`
x ungleich y	`x != y`
x kleiner y	`x < y`
x größergleich y	`x >= y`
nicht x	`!x`
x oder y	`x || y`
x und y	`x && y`
wenn x, dann y, sonst z	`x ? y : z`

3.4.2 Funktionen

Für einfache arithmetische Operationen bietet OPL unter anderem folgende Optionen:

- `min`
- `max`
- `count`
- `abs`
- `element`

3.4.3 Strings

Neben `int` und `float` für Ganz- und Gleitkommazahlen bietet OPL noch `string` an. Diese verwendet man einfach wie die anderen beiden Typen:

```
string vorlesung = "Graphen und diskrete Optimierung";
```

3.4.4 Range

Eine Reihe von Zahlen kann in OPL mithilfe des Schlüsselwortes `range` angegeben werden. So kann die Reihe $1, \ldots,$ `anzahlVars` zum Beispiel als

```
range Vars = 1..anzahlVars;
```

definiert werden. So können auch Entscheidungsvariablen in (mehrdimensionalen) Arrays angelegt werden:

```
dvar <Typ> <name>[<Range>];
```

Für unser einfaches Beispiel lässt sich folgender Code verwenden:

```
range vars = 1..anzahlVars;
dvar float+ x[vars];
```

3.4.5 Allquantoren

Um mehrere ähnliche Nebenbedingungen in einer Gruppe zusammenzufassen, lassen sich logische Bedingungen im Modell mithilfe von Allquantoren abrufen. Die Nebenbedingungen in unserem Beispiel können so zusammengefasst werden:

$$A_{j,1}x_1 + A_{j,2}x_2 \leq b_j \qquad \forall j \in \{1, 2, 3\}$$

Die Syntax um Quantifizierung in OPL zu erreichen lautet wie folgt:

```
forall(<varnamen> in <Range>){
        erzeuge Nebenbedingungen;
}
```

Diese Syntax erinnert an eine *foreach*-Schleife, wie sie aus anderen Programmiersprachen bekannt ist. Die Laufweite des `forall` wird mithilfe eines `Range`-Objekts definiert, welche auch schon für die Arrays benutzt wurden. So lassen sich also elegant mittels eines einzigen `forall` aus den gespeicherten Daten alle Nebenbedingungen erzeugen:

```
forall(j in 1..anzahlCons){
        A[j][1] * x1 + A[j][2] * x2 <= b[j];
}
```

Dies lässt sich natürlich auch mit einer Summe kombinieren, um folgende Formulierung zu erhalten:

```
forall (j in 1..anzahlCons) {
        sum(i in 1..anzahlVars) (A[j][i] * x[i]) <= b[j];
}
```

welches der folgenden mathematischen Schreibweise entspricht:

$$\sum_{i=1}^{\texttt{anzahlVars}} A_{j,i} \cdot x_i \leq b_j \qquad \forall j \in \{1, \ldots, \texttt{anzahlCons}\}$$

3.4.6 Boolsche Entscheidungsvariablen

Entscheidungsvariablen in OPL sind ihrer Natur nach stets einfache numerische Werte.
Boolsche Werte stehen nur als Entscheidungsvariablen (`dvar`) zur Verfügung. Diese werden mit dem Schlüsselwort `boolean` initialisiert und sind äquivalent zu den Ganzzahlen $\{0, 1\}$ – entsprechend lassen sich auch Rechenoperationen mit booleans durchführen. Dabei entspricht wie erwartet 0 dem Wert false und 1 dem Wert true.
Die Syntax ist:

```
dvar boolean wahrheitswert;
```

3.4.7 Tuple

Tupel sind definierbare Strukturen um logisch zusammenhängende Daten verschiedenen Typs zu speichern – ähnlich wie `structs` in C. Sie lassen sich benennen und können beliebige andere Datentypen und -strukturen beinhalten. Definiert werden Tupel im Modell, speichern lassen sich ihre Werte in Daten-Dateien. Unser einfaches lineares Programm lässt sich unter Verwendung eines Tupels für die Constraint-Konstanten wie folgt umschreiben:

Modell

```
//Variablen
dvar float+ x1;
dvar float+ x2;
int anzahlVars = ...;
int c[1..anzahlVars] = ...;
```

```
//Tuple der Nebenbedingungen
tuple ConstraintWerte {
        int x1Wert;
        int x2Wert;
        int bWert;
}

ConstraintWerte erstes = ...;
ConstraintWerte zweites = ...;
ConstraintWerte drittes = ...;

//Zielfunktion
maximize c[1]*x1 + c[2]*x2;

//Nebenbedingungen
subject to {
        erstes.x1Wert*x1 + erstes.x2Wert*x2 <= erstes.bWert;
        zweites.x1Wert*x1 + zweites.x2Wert*x2 <= zweites.bWert;
        drittes.x1Wert*x1 + drittes.x2Wert*x2 <= drittes.bWert;
}
```

Daten

```
anzahlVars = 2;
v = [30, 50];
erstes = <4, 11, 880>;
zweites = <1, 1, 150>;
drittes = <0, 1, 60>;
```

Beachten Sie, wie mit dem . -Operator auf die Inhalte der Tupel zugegriffen wird und dass sich Tupel durch Spitzklammern < , > initialisieren lassen. Beim Initialisieren werden die angebenden Werte der Reihe nach auf die Elemente des Tupels zugewiesen, achten Sie also auf Reihenfolge und Typen.

3.4.8 Mengen (Sets)

In OPL ist eine Menge eine nicht-indizierte Sammlung von Elementen ohne Duplikate. Standardmäßig sind in OPL Mengen geordnet nach Reihenfolge des Hinzufügens und alle Operationen, die OPL zum Zugriff auf Mengen bereitstellt, werden diese Ordnung nicht ändern. Um eine Menge zu erzeugen werden geschweifte Klammern { , } verwendet.

```
{int} primzahlen = {2,3,5,7,11};
```

Es ist nicht möglich eine Menge aus anderen Datenstrukturen zu bilden, etwa Arrays oder weiteren Mengen. Was jedoch möglich ist, ist eine Menge aus Tupeln anzulegen. Analog zu verschachtelten Arrays lässt sich der obige Code für Tupel unter Verwendung von Quantifizierung umschreiben. Im folgenden Code verwenden wir zusätzlich eine variable Anzahl an Entscheidungsvariablen und eine Summation für die Zielfunktion.

Modell

```
int anzahlVars = ...;
dvar float+ x[1..anzahlVars];
int c[1..anzahlVars] = ...;

tuple ConstraintWerte {
        int x1Wert;
        int x2Wert;
        int bWert;
```

```
}

{ConstraintWerte} c = ...;

//Zielfunktion
maximize sum (i in 1..anzahlVars) c[i] * x[i];

//Nebenbedingungen
subject to {
        forall (werte in c) {
                werte.x1Wert*x[1] + werte.x2Wert*x[2] <= werte.bWert;
        }
}
```

Daten

```
anzahlVars = 2;
c = [30, 50];
Ab = { <4, 11, 880>, <1, 1, 150>, <0, 1, 60> };
```

Mengen haben eine variable Größe, sodass weder beim Einlesen noch beim Durchlaufen einer Menge deren Länge bekannt sein muss (anders als bei Arrays). Beachten Sie, dass Sie einfach mittels `foreach (werte in c)` durch alle Tupel iterieren können. Um auf das i-te Element einer Menge zugreifen zu können, können sie auch die `item(<Menge>,<index>)` -Funktion verwenden. Im obigen Beispiel liefert der Aufruf `item(Ab,2).bWert` die Zahl 60, da Mengen anders als Arrays ab 0 gezählt werden.
Summen über Mengen lassen sich wie folgt einfacher in OPL umsetzen:

```
sum (element in set) element
```

3.4.9 Ausgabe

Die Werte von Entscheidungsvariablen können von OPL auch ausgegeben werden. Dies geschieht im Post-Processing Block, welcher mit dem Befehl `execute` nach der Modelldefinition aufgerufen wird. Die Funktion `writeln` wird benutzt, um Werte in der Konsole, gefolgt von einer neuen Zeile, zu drucken. So können Sie zum Beispiel die Werte der Optimierungsvariablen x_1 und x_2 wie folgt ausgeben:

```
execute {
        writeln("x1:", x1);
        writeln("x2:", x2);
}
```

Für eine vordefinierte Anzahl an Variablen `anzahlVars`:

```
execute {
        for (var i = 1; i <= anzahlVars; i++) {
                writeln("x", i, ":", x[i]);
        }
```