Technische Universität Chemnitz
Prof. Dr. C. Helmberg, A. Lau

Chemnitz, 19.10.2009

Optimierung für Nichtmathematiker
Übung 2

Einführung in die Modellierungssprache AMPL

1. Wir betrachten zunächst das Mozartproblem aus der Vorlesung:

$$\begin{array}{ll} \max & 3 \cdot x_K + 2 \cdot x_T \\ \text{s.t.} & 2 \cdot x_K + 1 \cdot x_T \leq 10 \\ & 1 \cdot x_K + 1 \cdot x_T \leq 6 \\ & 1 \cdot x_K + 2 \cdot x_T \leq 9 \\ & x_K \geq 0, x_T \geq 0 \end{array}$$

Einige AMPL-Befehle am Beispiel:

```
var xK;
var xT;

maximize Profit: 3*xK + 2*xT;

subject to Marzipan: 2*xK + 1*xT <= 10;
subject to Nougat: 1*xK + 1*xT <= 6;
subject to Edelherb: 1*xK + 2*xT <= 9;

subject to Schranke_xK: 0 <= xK;
subject to Schranke_xT: 0 <= xT;
```

- Dateien, die Modelle enthalten, werden als *.mod abgespeichert.
- Alle Zeilen müssen mit einem Semikolon enden.
- Alle vergebenen Namen dürfen maximal einmal vorkommen.
- Kommentare mit `#`: `# Das ist ein Kommentar.`
- Vergleiche: `=, <=, >=`
- Definition von Variablen: `var variablenname;`
- Zielfunktion: `minimize/maximize zielfunktionsname: ... ;`
- Nebenbedingung: `subject to nebenbedingungsname: ... ;`
- Aufruf/Lösen mit AMPL in der Kommandozeile:

```
ampl
model modelname.mod;
solve;
```

- Aufruf/Lösen mit glpsol in der Kommandozeile:

```
./glpsol -m modelname.mod
```

Die obige Darstellungsweise wird leicht unübersichtlich. Außerdem ist es aufwändig, wenn man das Modell z. B. um weitere Produktvarianten erweitern will. Hierbei wäre es günstiger, wenn man die Zielfunktion und die Nebenbedingungen einmal allgemein aufschreibt und zusätzlich die konkreten Werte über Dateien einlesen kann. AMPL bietet diese Möglichkeit an:

- Modelldatei: mozart2.mod

```
set P; # Produkte

param a1 { i in P }; # fuer Marzipan
param a2 { i in P }; # fuer Nougat
param a3 { i in P }; # fuer Edelherb
param c { i in P }; # Zielfunktion
param b { i in 1..3 }; # rechte Seiten

var x { i in P }; # Variablen

maximize Profit: sum { i in P } c[i] * x[i];

subject to Marzipan: sum { i in P } a1[i] * x[i] <= b[1];
subject to Nougat: sum { i in P } a2[i] * x[i] <= b[2];
subject to Edelherb: sum { i in P } a3[i] * x[i] <= b[3];

subject to Schranken { i in P }: 0 <= x[i];
```

- Datendatei: mozart2.dat

```
set P := Kugel Taler;
param: a1 a2 a3 c:=
Kugel 2 1 1 3
Taler 1 1 2 2;
param: b:=
1 10
2 6
3 9 ;
```

Neue Befehle:

- Definition von (Index-)Mengen: `set setname;`
- Parameter: `param paramname { i in 1..number };`
  oder `param paramname { i in setname };`
- auch Definition von mehreren Variablen über Indexmengen möglich
- Zugriff auf Elemente: `x[1]` entspricht x_1
- Summe: `sum { i in setname } ... ;`

Lösen der Probleme:

- Aufruf/Lösen mit AMPL in der Kommandozeile:

```
ampl
model modelname.mod;
data dataname.dat;
solve;
```

- Aufruf/Lösen mit glpsol in der Kommandozeile:

```
./glpsol -m modelname.mod -d dataname.dat
```

Aufgabe: Ändern Sie die Datei mozart2.dat so ab, dass es neben den Kugeln und Talern auch noch Stangen gibt, die einen Gewinn von 6 bringen und für die jeweils drei Einheiten Marzipan und je eine Einheit Nougat und Edelherb benötigt werden.

2. **Rucksackproblem:** Gegeben sei eine Menge an n Objekten. Von diesen Objekten kennen wir das Gewicht g_i und den Wert w_i. Leider können wir mit unserem Rucksack nicht alle Objekte transportieren (Rucksackkapazität k). Ziel ist es nun, einen möglichst hohen Gewinn zu erzielen ohne den Rucksack zu überladen. (Hinweis: Jedes Objekt kommt genau einmal vor.)

   Daraus ergibt sich folgende Optimierungsaufgabe:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} w_i \cdot x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} g_i \cdot x_i \leq k \\ & x_i \in \{0,1\}, \forall i = 1, \ldots, n \end{aligned}$$

   Gegeben ist folgende Datendatei:

```
set Objekte := Vase Uhr Muenze Lampe Geld
               Briefmarke Tassenservice Fernseher Topf Kanne;
param Wert:=
Vase 30 Uhr 20 Muenze 40 Lampe 80 Geld 150 Briefmarke 300
Tassenservice 100 Fernseher 200 Topf 70 Kanne 60;
param Gewicht :=
Vase 5 Uhr 3 Muenze 3 Lampe 30 Geld 5 Briefmarke 3
Tassenservice 40 Fernseher 50 Topf 10 Kanne 15;
param Rucksack := 60;
```

   Aufgabe: Stellen Sie mit AMPL ein Modell auf, welches die Begriffe der Datendatei nutzt. Hinweis: Ganzzahlige Variablen erhalten Sie mit `var varname integer;` (siehe unten).

3. **Normminimierung**
   Gegeben seien n Steckdosen mit ihren Positionen $p_i = (x_i, y_i), i = 1, \ldots, n$. Gesucht ist die Position des Verteilerkastens $p = (x, y)$, mit dem jede Steckdose direkt verbunden werden soll, so dass die Gesamtkabellänge minimal wird. Bei der Verlegung der Kabel ist zu beachten, dass die Kabel zwar parallel, aber nur senkrecht und waagerecht verlaufen dürfen.

   Daraus ergibt sich folgendes Optimierungsmodell:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} \|p_i - p\|_1 \\ \text{s.t.} \quad & p \in \mathbb{R}^2 \end{aligned}$$

   Als lineares Programm aufgeschrieben:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} (s_i + t_i) \\ \text{s.t.} \quad & x - x_i \leq s_i, \forall i = 1, \ldots, n \\ & y - y_i \leq t_i, \forall i = 1, \ldots, n \\ & -x + x_i \leq s_i, \forall i = 1, \ldots, n \\ & -y + y_i \leq t_i, \forall i = 1, \ldots, n \\ & s_i \geq 0, t_i \geq 0, \forall i \in 1, \ldots, n \\ & x, y \in \mathbb{R} \end{aligned}$$

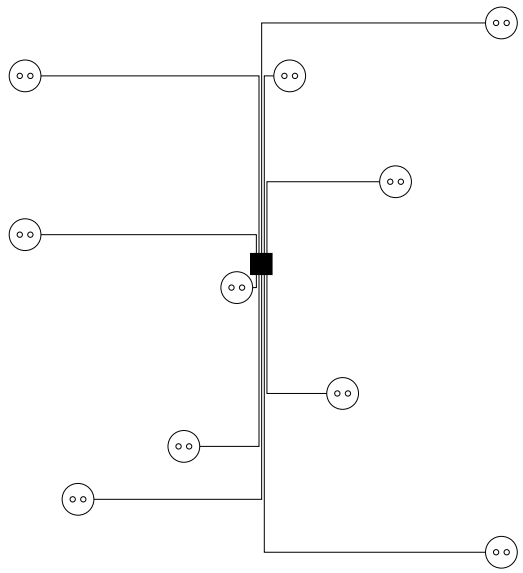
Daraus ergibt sich:

```
param n;
param Punkte{i in 1..n, j in 1..2};

var x;
var y;
var s { i in 1..n };
var t { i in 1..n };

minimize Kosten: sum{ i in 1..n } (s[i] + t[i]);

subject to Schranken_s {i in 1..n }: 0 <= s[i];
subject to Schranken_t {i in 1..n }: 0 <= t[i];
```

Aufgabe: Ergänzen Sie die fehlenden Nebenbedingungen. Nutzen Sie zum Testen die Datendatei `strom.dat`.

```
param n:= 10;
param Punkte: 1 2:=
1 1 9
2 5 5
3 6 9
4 4 2
5 10 0
6 1 6
7 2 1
8 8 7
9 7 3
10 10 10;
```

Darstellung der Optimallösung:



4. Verwendung des NEOS-Solvers: `http://neos.mcs.anl.gov/neos/solvers/index.html`
   Transportproblem: Gegeben sind m Lager, wobei jedes k_i Waren enthält, und n Supermärkte, die s_j Waren benötigen. Durch die Transporte zwischen Lager und Supermarkt entstehen Kosten pro Einheit tranportiertes Gut, welche auch von Start und Ziel abhängen. Gesucht sind die optimalen Transporte, so dass alle Bedarfe gedeckt werden und kein Lager mehr liefert, als es Kapazität hat.

   Modell `transport.mod`:

```
set Lager;
set Supermarkt;

param Kosten{ i in Lager, j in Supermarkt };
param Lager_kap{ i in Lager };
param Supermarktbedarf{ j in Supermarkt };

var x { i in Lager, j in Supermarkt } integer;

minimize Gesamtkosten: sum { i in Lager, j in Supermarkt } Kosten[i,j]*x[i,j];

subject to Lagerkapazitaet { i in Lager }:
        sum{ j in Supermarkt } x[i,j] <= Lager_kap[i];
subject to Bedarf_decken { j in Supermarkt }:
        sum{ i in Lager } x[i,j] = Supermarktbedarf[j];

subject to Schranken { i in Lager, j in Supermarkt }: 0 <= x[i,j];
```

   Beispieldaten `transport.dat`:

```
set Lager := L1 L2 L3;
param Lager_kap:= L1 300 L2 300 L3 200;
set Supermarkt := S1 S2 S3 S4;
param Supermarktbedarf := S1 100 S2 200 S3 250 S4 150;
param Kosten: S1 S2 S3 S4 :=
L1 100 150 50 200
L2 80 200 130 160
L3 120 160 70 100;
```

   Neue Befehle:

   - ganzzahlige Variablen: `var x integer`
   - Es kann auch mehrdimensionale Indexmengen geben, z.B. `var x { i in X, j in Y }`, Zugriff auf Element $x_{i,j}$: `x[i,j]`

   Lösen Sie das Transportproblem mithilfe des NEOS-Solvers – verwenden Sie MINTO. Nutzen Sie dazu die folgende Kommando-Datei `transport.cmd`:

```
solve;
display x, Gesamtkosten;
```

   Was passiert mit der Lösung, wenn man die Ganzzahligkeit der Variablen nicht fordert? Stimmt das auch in Aufgabe 2?