Basisinformationen zu AMPL

AMPL (A Mathematical Programming Language) ist eine algebraische Modellierungssprache für mathematische Optimierungsprobleme. AMPL wurde an den Bell Laboratories entwickelt. AMPL bietet die Möglichkeit, Modelle für mathematische Optimierungsprobleme in abstrakter Form aufzustellen sowie die resultierenden Optimierungsprobleme durch Aufruf externer Solver einer Lösung zuzuführen. Eine weitere Stärke von AMPL ist die Trennung von Modell- und Datenteil.

Detaillierte, regelmässig auf den neuesten Stand gebrachte Informationen zu AMPL sind im WWW unter `http://ampl.com` verfügbar. Als Referenz für den Sprachumfang von AMPL bietet sich das folgende Buch an:

> R. Fourer, D.M. Gay und B.W. Kernighan, AMPL: A Modelling Language for Mathematical Programming, 2-nd edition, Duxbury Press, ISBN 0-534-38809-4. Gratis downloadbar unter `http://ampl.com/resources/the-ampl-book/`.

Einige Kurzbeschreibungen zum Sprachumfang sind frei im WWW erhältlich (für Links siehe die WWW-Seite von AMPL.)

AMPL wird in verschiedenen Versionen (kostenpflichtig bzw. gratis) in Vollform bzw. mit Einschränkungen angeboten.

- Kostenpflichtige Versionen: *Business* (für Firmen) und *Research* (für den Einsatz in der akademischen Welt)
- Kostenfreie Versionen:
  - *Webbasiertes Try AMPL Now Interface:* `http://ampl.com/try-ampl/try-ampl-online/` für kleinere Beispiele ohne Notwendigkeit einer Installtation von AMPL.
  - *Freie Demo Version/Student:* ohne zeitliche Einschränkung benutzbar, Problemgröße eingeschränkt ($\leq$ 300 Variable und Restriktionen bzw. $\leq$ 500 im rein linearen Fall). Enthaltene Solver CPLEX, Gurobi, MINOS und SNOPT.
  - *Teaching*: Zeitlich begrenzte Vollversion für eine bestimmte Lehrveranstaltung. Für Optimierung 1 (SS 2015) wurde eine solche Lizenz beantragt. Es gibt Versionen für Windows und Linux (32 und 64 Bit) und Mac. Eine Reihe von Solvern für diverse Typen von Optimierungsproblemen sind hier bereits enthalten und brauchen nicht separat beschafft und installiert werden. Ferner gibt es weitere Solver im public domain bzw. via NEOS (siehe unten). Details in der Lehrveranstaltung.

Darüberhinaus gibt es mittlerweile auch die Möglichkeit AMPL und Solver remote über NEOS zu verwenden (dies erlaubt u.a. auch Solver zu verwenden, die man lokal nicht verfügbar hat). Details siehe dazu `http://ampl.com/try-ampl/run-ampl-on-neos/`.
Via Kestrel, siehe `http://ampl.com/try-ampl/run-ampl-on-neos/#local-ampl` (separat zu installieren) kann ein auf NEOS vorhandener Solver aus einer lokal installierten AMPL Version aufgerufen werden.

AMPL ist standarmäßig als Kommandozeilenversion verfügbar. Mittlerweile gibt es aber auch eine neue integrierte Entwicklungsumgebung (AMPL IDE), siehe `http://ampl.com/products/ide/`. Diese ist bei Bedarf extra zu installieren.

Das Hauptanwendungsgebiet von AMPL ist zwar die Modellierung komplexerer Optimierungsprobleme, aber AMPL kann auch als Schnittstelle zu Solvern verwendet werden, um vorliegende Optimierungsprobleme zu lösen. Das folgende Beispiel soll dies an Hand eines kleinen linearen Programms illustrieren.

Beispiel 1: (Direkte Verwendung, eigentlich untypisch für AMPL)

Kommandofolge (interaktiv)

```
var XB;
var XC;
maximize Profit: 25*XB + 30*XC;
subject to Zeit: (1/200)*XB + (1/140)*XC <= 40;
subject to B_Limit: 0 <= XB <= 6000;
subject to C_Limit: 0 <= XC <= 4000;

solve;
display XB,XC;
quit;
```

Beispiel 2: Eine typischere Anwendung von AMPL

Interaktive Befehlsabfolge:

```
model;
param n; # gegebene Daten
set Perioden := 1..n; Anmerkung: Nur zur Demonstration. Menge im Bsp nicht benötigt
set Produkte;
set Maschinen;
param Bedarf {Produkte} >= 0;
param maxBedarf {p in Produkte} >= Bedarf[p];
param Erloes {Produkte};
param Herstellung {Produkte, Maschinen} >= 0;
param Kapazitaet {Maschinen} >= 0, integer; vordefiniert: nichtnegativ und ganzzahlig
# gesuchte Variablen:
var Produktion {Produkte} >= 0, integer; definiert ganzzahlige Variable
```

```
maximize Gewinn: sum {p in Produkte}
  Produktion[p] * Erloes[p] # Zielfunktion
subject to Kap {m in Maschinen}: # Nebenbedingungen
  sum {p in Produkte} Produktion[p] * Herstellung[p,m]
        <= Kapazitaet[m];
Bedarfsdeckung {p in Produkte}: Bedarf[p] <= Produktion[p] <= maxBedarf[p];
```

Daten:

```
data;
param n := 12;
set Produkte := Zimtsterne Vanillekipferln;
set Maschinen := Mixer Herd;
param:                Bedarf  maxBedarf :=
   Zimtsterne         30.5     200.9
   Vanillekipferln    79        82 ;
param Kapazitaet :=
         Mixer   10
         Herd    20;
param Herstellung: #  zweidimensionale Tabellen
                   Herd  Mixer :=
   Zimtsterne       12.3    4
   Vanillekipferln   9.5    5 ;
```

oder (transponierte Reihenfolge von Zeilen und Spalten:

```
param Herstellung (tr):
     Zimtsterne Vanillekipferln :=
   Herd        12.3    9.5
   Mixer          4      5 ;
```

oder:

```
param Herstellung :=
   Zimtsterne Herd         12.3
   Zimtsterne Mixer         4
   Vanillekipferln Herd     9.5
   Vanillekipferln Mixer    5 ;
```

Fileorientierte Kommandoabfolge

Es ist empfehlenswert das Modell und die Daten nicht interaktiv einzugeben, wie oben vorgeführt, sondern in einer separaten Modelldatei und einer separaten Datendatei zu erfassen.

Im obigen Beispiel könnte man die Datei `prod.mod` erstellen, die mit der Zeile `param n` startet und mit der Zeile vor `data` endet und eine Datei `prod.dat`, die mit der Zeile nach `data` startet. Diese Files können dann mit `model` bzw. `data` geladen werden (siehe untenstehende Befehlsliste).

Ferner kann man auch noch die Kommandos in einer Kommandodatei `prod.run` speichern und diese dann mit `include` bzw. mit `commands aufrufen`. Für das obige Beispiel könnte `prod.run` z.B. wie folgt aussehen

```
reset;
model prod.mod;
data prod.dat;
solve;
display Produktion;
```

Einsatzbereich für AMPL: AMPL erlaubt die Formulierung von linearen und nichtlinearen Zielfunktionen und Restriktionen und erlaubt die Einführung von ganzzahligen Variablen. Wenn es um die Lösung eines aufgestellten Modells geht, muss jedoch ein für das Modell passender Solver vorhanden sein.

Beispiel 3: ein nichtlineares Optimierungsproblem

```
param N:=100;
param L:=1;
param h:=2*L/N;
var x {0..N};
var y {0..N};
minimize pot_energy: sum{j in 1..N} (y[j-1]+y[j])/2
subject to x_left_anchor: x[0]=0;
subject to y_left_anchor: y[0]=0;
subject to x_right_anchor: x[N]=L;
subject to y_right_anchor: y[N]=0;
subject to link_up {j in 1..N} (x[j]-x[j-1])^2+(y[j]-y[j-1])^2<=h^2;
let {j in 0..N} x[j]:=j*L/N;
let {j in 0..N} y[j]:=0;
solve;
```

Einige nützliche Befehle:

`model` *Datei*`.mod;` # Laden der Modelldatei

```
data Datei.dat;     # Laden der Datendatei
include Dateiname;
commands Datei.run;
option solver cplex    # Wählt CPLEX als Solver aus, analog für andere Solver
solve;    # Lösen des Modells  mit dem gewählten Solver oder dem Default)
display Gewinn, {p in Produkte} Produktion[p];    # Ergebnisanzeige
display Kap.dual, Kap.slack;    # Anzeige der dualen Variablen und der Slacks
display Produktion.lb, Produktion.ub;    # Anzeige von unteren und oberen Schranken
display Produktion.rc;    # Anzeige von reduzierten Kosten
display Gewinn > prod.out;    # Ausgabe in neu erzeugte Datei prod.out
display Kap.dual >> prod.out;    # Hinzufügen eines Ergebnisses zur Ergebnisdatei
close prod.out;    # Schliessen der Ergebnisdatei
let Kapazitaet[''Herd''] := 30;    # Datenänderung
reset; reset model; reset data;    # Reset der Files
drop Bedingung;    # Streichen einer Restriktion
restore Bedingung;    # Wiedereinführung einer Restriktion
fix Variable;    # Fixierung einer Variablen auf einen Wert z.B. fix Produktion[''Zimtsterne'']:=14
unfix Variable;    # Aufhebung der Fixierung
```