

Modellierung und Optimierung mit OPL

7 OPL-Schnittstellen zu anderen Anwendungen

Andreas Popp



Dieser Foliensatz ist lizenziert unter einer Creative Commons
Namensnennung - Weitergabe unter gleichen Bedingungen 4.0
International Lizenz.

7.1 Optimierungsablauf in OPL

7.2 Datenquellen

Excel-Tabellen

Excel-Tabellen

Datenbanken

7.3 OPL in Programmabläufen

Kommandozeile und Datenquellen

Kommandozeile und Datenquellen

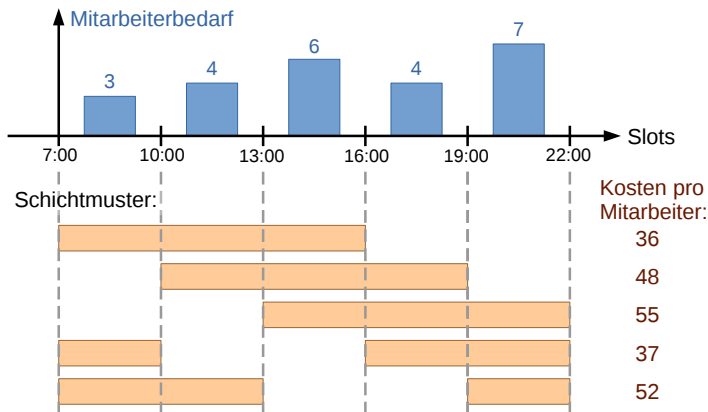
Die Concert-API am Beispiel Java

Die Concert-API am Beispiel Java

ILOG Script

ILOG Script

Beispiel: Vindoo Support



(vgl. Pinedo: Planning and Scheduling in Manufacturing and Services)

Modell: Cyclic-Staffing-Problem

Indexmengen:

T Menge der Zeitslots

S Menge der Schichtmuster

Parameter:

c_s Kosten für Schichtmuster $s \in S$

d_t Bedarf in Zeitslot $t \in T$

a_{ts} Verfügbarkeit der Mitarbeiter aus
Schichtmuster $s \in S$ in Zeitslot $t \in T$

Entscheidungsvariablen:

x_s Eingesetzte Mitarbeiter in Schichtmuster $s \in S$

Modellbeschreibung:

$$\begin{aligned} \min \quad & \sum_{s \in S} c_s \cdot x_s \\ \text{s.t.} \quad & \sum_{s \in S} a_{ts} \cdot x_s \geq d_t \quad \forall t \in T \quad (\text{I}) \\ & x_s \in \mathbb{Z}_0^+ \quad \forall s \in S \end{aligned}$$

7.1 Optimierungsablauf in OPL

7.1 Optimierungsablauf in OPL

Excel-Tabellen

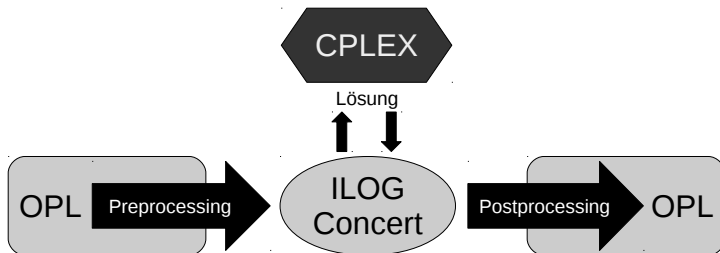
Datenbanken

Kommandozeile und Datenquellen

Die Concert-API am Beispiel Java

ILOG Script

Optimierungsablauf in OPL



7.1 Optimierungs- ablauf in OPL

7.2 Datenquellen

Excel-Tabellen

Datenbanken

7.3 OPL in Programmabläufen

Kommandozeile und
Datenquellen

Die Concert-API am
Beispiel Java

ILOG Script

7.2 Datenquellen

Aufbau einer SheetConnection

Der Objekt-Datentyp `SheetConnection` stellt Verbindungen zu Excel-Tabellen dar (nur unter MS Windows).

Erstellen einer SheetConnection

```
SheetConnection Name("Pfad der Tabelle");
```

Mögliche Datentypen für Einlesen und Schreiben

- ▶ nulldimensionale Datentypen, d.h. einfache Variablen. Auslesen einer einzelnen Zelle.
- ▶ eindimensionale Datentypen, d.h. Sets und einfache Arrays. Auslesen einer Zeile oder Spalte.
- ▶ zweidimensionale Datentypen, d.h. zweidimensionale Arrays. Auslesen einer Zellenmatrix.

Lesen und Schreiben mit absoluter Zelladressierung

Lesen mit absoluter Zelladressierung

Variablenname from *SheetRead(SheetConnection-Name, "Tabellenname!Startzelle:Endzelle")*

Schreiben mit absoluter Zelladressierung

Variablenname to `SheetWrite(SheetConnection-Name, "Tabllenne!Startzelle:Endzelle")`

In Beispiel „Vindoo Support“

Excel-Tabelle für Beispiel „Vindo Support“

	A	B	C	D	E	F	G	H
1	t ↓/s →	1	2	3	4	5	d	Obj
2	1	1	0	0	1	1	3	
3	2	1	1	0	0	1	4	
4	3	1	1	1	0	0	6	
5	4	0	1	1	1	0	4	
6	5	0	0	1	1	1	7	
7	c	36	48	55	37	52		
8	x							

Auszüge aus der Datendatei

```
// SheetConnection  
SheetConnection sheet("CyclicStaffingProblem.xls");
```

```
// Indexpmengen  
T from SheetRead(sheet, "Data!B1:F1");
```

```
//Parameter  
d from SheetRead(sheet, "Data!G2:G6");
```

```
//Entscheidungsvariablen  
x to SheetWrite(sheet, "Data!B8:F8");
```

Lesen und Schreiben mit Namensbereichen

MS Excel ermöglicht es einzelnen Zellbereichen Namen zu geben.

Lesen mit Namensbereichen

```

Variablename from SheetRead(SheetConnection-Name,
"Bereichsname")

```

Schreiben mit Namensbereichen

Variablename to `SheetWrite(SheetConnection-Name, "Bereichsname")`

Excel-Tabellen

Kommandozeile und Datenquellen

Die Concert-API am Beispiel Java

ILOG Script

In Beispiel „Vindoo Support“

Excel-Tabelle für Beispiel „Vindo Support“

	A	B	C	D	E	F	G	H
1	t ↓/s →	1	2	3	4	5	d	Obj
2	1	1	0	0	1	1	3	
3	2	1	1	0	0	1	4	
4	3	1	1	1	0	0	6	
5	4	0	1	1	1	0	4	
6	5	0	0	1	1	1	7	
7	c	36	48	55	37	52		
8	x							

Der gelbe Bereiche heiße „ParamA“

Auszüge aus der Datendatei

```
// SheetConnection  
SheetConnection sheet("CyclicStaffingProblem.xls");
```

```
//Parameter  
a from SheetRead(sheet, "ParamA");
```

Aufbau einer DBConnection

Der Objekt-Datentyp `DBConnection` stellt Verbindungen zu einer Datenbank dar.

Erstellen einer DBConnection

```
DBConnection Name("Schnittstelle",
"Verbindungs-String");
```

Unterstützte Datenbank-Schnittstellen

- ▶ DB2
- ▶ Oracle (Version 10 und 11)
- ▶ OLE DB (MS SQL Server)
- ▶ ODBC (u.a. für MS Access)

Beispiel-Datenbank: CyclicStaffingProblem

s	t	a
1	1	1
1	2	1
1	3	1
1	4	0
1	5	0
2	1	0
2	2	1
2	3	1
2	4	1
2	5	0
3	1	0
3	2	0
3	3	1
3	4	1
3	5	1
4	1	1
4	2	0
4	3	0
4	4	1
4	5	1
5	1	1
5	2	1
5	3	0
5	4	0
5	5	1

(a) Tabelle A

ind	d
1	2
2	4
3	6
4	4
5	7

(b) Tabelle T

ind	c	x
1	36	
2	48	
3	55	
4	37	
5	52	

(c) Tabelle S

Einlesen von Array-Daten

Die Beispiel-Datenbank CyclicStaffingProblem sei über eine ODBC-Schnittstelle verbunden. Der User „user“ mit dem Passwort „password“ hat die nötige Zugangsberechtigung.

Auszüge aus der Datendatei

```
// DBConnection
DBConnection
db("odbc", "CyclicStaffingProblem/user/password");
```

```
// Indexmengen
T from DBRead (db, "SELECT ind from T");
```

```
// Parameter
c from DBRead(db, "SELECT ind,c from S");
a from DBRead(db, "SELECT t,s,a from A");
```

Einlesen und Schreiben von Tupel-Daten

Auszüge aus der Modelldatei

```
//Tuple
tuple shift{
    int ind;
    float c;
}
tuple result{<
    int x;
    int ind;
}
```

```
//Postprocessing
{result} r = {<x[s],s.ind>|s in S};
```

Auszüge aus der Datendatei

```
//Indexmengen
S from DBRead(db, "SELECT ind,c from S");
```

```
//Entscheidungsvariablen
r to DBUpdate(db, "UPDATE S SET x=? WHERE ind=?");
```


7.3 OPL in Programmabläufen

Beispiel: Cyclic-Staffing-Problem

Algorithmische Lösung für das Cyclic-Staffing-Problem

1. P_1 sei die LP-Relaxation des Cyclic-Staffing-Problems. Löse P_1 , wenn Optimallösung ganzzahlig Ende, sonst weiter zu 2.
2. Füge P_1 die Nebenbedingung

$$\sum_{s \in S} x_s = \left| \sum_{s \in S} x_s^* \right|$$

hinzu und erhalte P_2 . Hat P_2 eine Lösung Ende, sonst weiter zu 3.

3. Füge P_1 die Nebenbedingung

$$\sum_{s \in S} x_s = \left[\sum_{s \in S} x_s^* \right]$$

hinzu und erhalte P_3 . Optimallösung von P_3 ist Optimallösung des Cyclic-Staffing-Problems.

Einbindung von OPL/CPLEX in Programmabläufen

Schnittstellen

- ▶ Kommandozeilenanwendung, insbesondere `oplrun`
- ▶ ILOG Concert-API
- ▶ CPLEX Callable Library
- ▶ CPLEX-Spezialschnittstellen
- ▶ ILOG Script

Einbindung von OPL/CPLEX in Programmabläufen

Schnittstellen

- ▶ Kommandozeilenanwendung, insbesondere `oplrun`
- ▶ ILOG Concert-API
- ▶ CPLEX Callable Library
- ▶ CPLEX-Spezialschnittstellen
- ▶ ILOG Script

Auswahl von Vorgängen für Anwendungsbeispiele

- ▶ automatisierte Erstellung und Lösung von Modellinstanzen
- ▶ Auslesen von Entscheidungsvariablen nach der Lösung
- ▶ Einspielen automatisch generierter Daten in eine Modellinstanz

Vorbereitungen im Beispiel des Cyclic-Staffing-Problems

Folgende Nebenbedingung werden hinzugefügt:

Haben beiden Schrankenparameter den gleichen Wert so erhält man effektiv:

Vor- und Nachteile der Kommandozeilenanwendung

Vorteile

- ▶ mit allen Programmiersprachen kombinierbar, die Kommandozeilenanwendungen ausführen können
- ▶ sehr vielseitig
- ▶ gut geeignet für akademische Forschung und kommerzielle Prototypen

Vor- und Nachteile der Kommandozeilenanwendung

Vorteile

- ▶ mit allen Programmiersprachen kombinierbar, die Kommandozeilenanwendungen ausführen können
- ▶ sehr vielseitig
- ▶ gut geeignet für akademische Forschung und kommerzielle Prototypen

Nachteile

- ▶ müssen entweder mit gegebenen Schnittstellen für Datenquellen kombiniert werden oder Datenein- und Ausgabe muss selbst programmiert werden
- ▶ empfindlich gegenüber Fehler in der Konfiguration der Plattform
- ▶ schlecht geeignet für Produkivsysteme

Quelltextausschnitt: Befehlsaufruf

```
'Löse P1
Shell ("oplrn" & modPath & " " & datPath)
```

Quelltextausschnitt: Workaround für leeren Lösungsraum

```
1 'Ergebnis ist nicht ganzzahlig,  
2 'bereite Schritt 2 vor  
3 dataSheet.Range("Obj").Value = "n"
```


ilog.opl Klassen, die Schnittstellen zu OPL bereitstellen

Excel-Tabellen
Datenbanken

Kommandozeile und Datenquellen

Die Concert-API am Beispiel Java

Beispiel: Cyclic-Staffing-Problem

```
IloOplFactory oplF = new IloOplFactory();  
  
IloOplModelSource modelSource =  
    oplF.createOplModelSource("CyclicStaffingProblem.mod");  
  
IloOplErrorHandler err = oplF.createOplErrorHandler();  
IloOplModelDefinition def = oplF.createOplModelDefinition(  
    modelSource, oplF.createOplSettings(err));  
  
IloCplex cplex = oplF.createCplex();  
  
IloOplModel opl = oplF.createOplModel(def, cplex);  
  
IloOplDataSource dataSource =  
    oplF.createOplDataSource("CyclicStaffingProblem.dat");  
opl.addDataSource(dataSource);  
  
opl.generate();  
  
opl.getCplex().solve();  
  
opl.printSolution(System.out);
```

7.1 Optimierungs-
ablauf in OPL

7.2 Datenquellen

Excel-Tabellen

Datenbanken

7.3 OPL in
Programmabläufen

Kommandozeile und
Datenquellen

Die Concert-API am
Beispiel Java

ILOG Script

Zugriff auf Modellelemente

- ▶ Zugriff auf mittels Klasse `IloOplElement` und Methode `getElement(String s)`.
- ▶ Datentypenübersetzung mittels `as`-Methoden

OPL-Datentyp	Concert-Datentyp-Kkeyword	Java-Datentyp
<code>int</code>	<code>Int</code>	<code>int</code>
<code>float</code>	<code>Num</code>	<code>double</code>
<code>string</code>	<code>Symbol</code>	<code>String</code>
<code>Set</code>	<code>Set</code>	
<code>Array</code>	<code>Map</code>	
<code>tuple</code>	<code>Tuple</code>	
<code>range</code>	<code>Range</code>	
<code>dvar</code>	<code>Var</code>	
<code>dexpr</code>	<code>Expr</code>	

Eigene Datenquellen definieren

- ▶ abstrakte Klasse `IloCustomOplDataSource`
- ▶ Datenübergabe durch Funktion `customRead`

Beispiel: Cyclic-Staffing-Problem

```
@Override
public void customRead() {
    //Datahandler-Objekt initialisieren
    IloOplDataHandler handler = getDataHandler();

    //ub übergeben
    handler.startElement("ub");
    handler.addIntItem(this.ub);
    handler.endElement();

    //lb übergeben
    handler.startElement("lb");
    handler.addIntItem(this.lb);
    handler.endElement();
}
```

Excel-Tabellen

Kommandozeile und Datenquellen

Die Concert-API am Beispiel Java

ILOG Script