

Modellierung und Optimierung mit OPL

4 Optimierung von Graphenproblemen

Andreas Popp



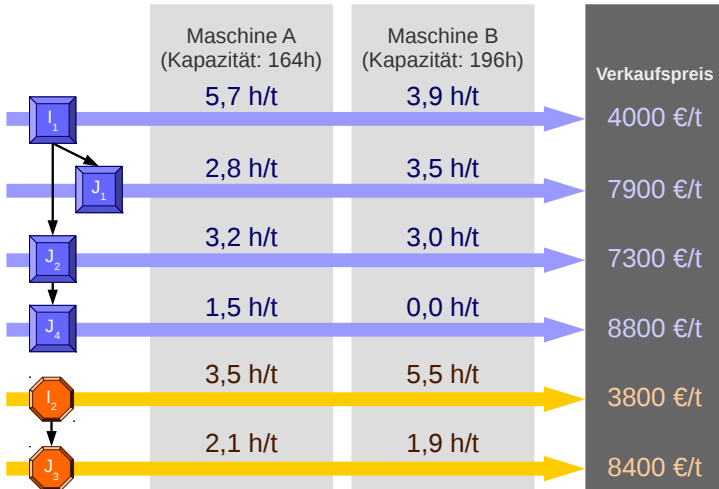
4.2 Abbilden von Graphen in OPL

4.3 OPL: Selbstdefinierte Tupel als Datenstruktur

4.4 OPL: Operatoren mit Bedingungen

4.1 Kurzeinführung in die Graphentheorie

Beispiel: Lewig Adelburg



4.1 Kurzeinführung
in die
Graphentheorie

4.2 Abbilden von
Graphen in OPL

4.3 OPL:
Selbstdefinierte
Tupel als
Datenstruktur

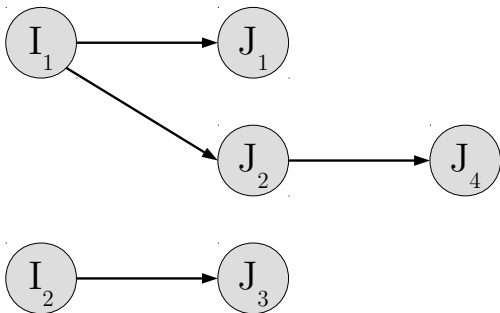
4.4 OPL:
Operatoren mit
Bedingungen

Graphenbegriff: Komponenten

4 Optimierung von Graphenproblemen

CC-BY-SA
A. Popp

4.1 Kurzeinführung in die Graphentheorie

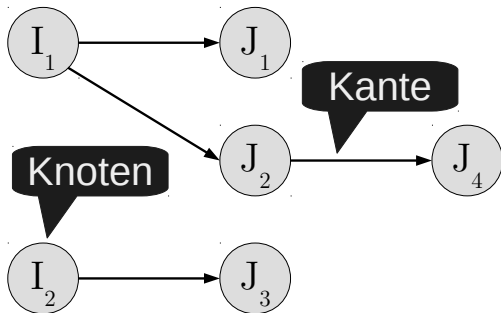


Graphenbegriff: Komponenten

4 Optimierung von Graphenproblemen

CC-BY-SA
A. Popp

4.1 Kurzeinführung in die Graphentheorie



- 6/25

4.2 Abbilden von Graphen in OPL

Indexmengen:

l Menge der Produkte

R Menge der Ressourcen

Parameter:

p_i Preis von Produkt $i \in I$

c_r Kapazität von Ressource $r \in R$

v_{ri} Kapazitätsverbrauch von Produkt $i \in I$ auf Ressource $r \in R$

E Menge der Kanten im Reihenfolgegraph

Entscheidungsvariablen:

x_i Produktionsmenge von Produkt $i \in I$

Modellbeschreibung:

$$\max \sum_{i \in I} p_i \cdot x_i$$

$$\text{s.t.} \quad \sum_{i \in I} v_{ri} \cdot x_i \leq c_i \quad \forall r \in R \quad (\text{I})$$

$$x_i \geq \sum_{(i,j) \in E} x_j \quad \forall i \in I \quad (\text{II})$$

$$x_i \geq 0 \quad \forall i \in I$$

4.2 Abbilden von Graphen in OPL

Anwendungsbeispiel für Graphen

$$x_i \geq \sum_{(i,j) \in E} x_j \quad \forall i \in I$$

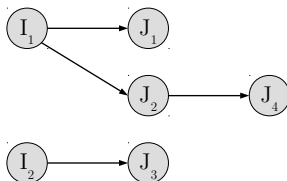
Frage: Wie kann der Graph in einem Optimierungsmodell abgebildet werden?

Definition: Adjazenzmatrix

Die Adjazenzmatrix eines Graphen $G = (V, E)$ mit $V = \{V_1, \dots, V_N\}$ ist eine quadratische $N \times N$ -Matrix (a_{ij}) , für die gilt:

$$a_{ij} = \begin{cases} 1 & \text{Kante } V_i \rightarrow V_j \text{ existiert} \\ 0 & \text{sonst} \end{cases} \quad (1)$$

Adjazenzmatrix im Beispiel



↓ Übersetzung in Adjazenzmatrix ↓

$$\begin{array}{c} I_1 \\ I_2 \\ J_1 \\ J_2 \\ J_3 \\ J_4 \end{array} \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Anwendung von Adjazenzmatrizen in Optimierungsproblemen

```
{string} I = ...;  
int a [I,I] = [  
    [0, 0, 1, 1, 0, 0],  
    [0, 0, 0, 0, 1, 0],  
    [0, 0, 0, 0, 0, 0],  
    [0, 0, 0, 0, 0, 1],  
    [0, 0, 0, 0, 0, 0],  
    [0, 0, 0, 0, 0, 0],  
];
```

$$x_i \geq \sum_{(i,j) \in E} x_j \quad \forall i \in I$$

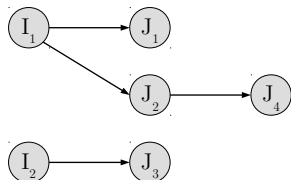
↓ OPL ↓

```
forall(i in I)
  x[i] >= sum (j in I)(a[i,j]*x[j]);
```

Definition: Adjazenzliste

Die Adjazenzliste eines Knoten $v \in V$ eines Graphen $G = (V, E)$ ist eine Menge $A_v \subseteq V$, welche alle Nachfolger von v beinhaltet.

Adjazenzlisten im Beispiel



$$\begin{array}{ll} A_{I_1} = \{J_1, J_2\} & A_{I_2} = \{J_3\} \\ A_{J_1} = \{\} & A_{J_2} = \{J_4\} \\ A_{J_3} = \{\} & A_{J_4} = \{\} \end{array}$$

4.2 Abbilden von Graphen in OPL

Anwendung von Adjazenzlisten in Optimierungsproblemen

```
{string} I = ...;  
{string} A[I] = [  
    {"J1", "J2"},  
    {"J3"},  
    {},  
    {"J4"},  
    {},  
    {}  
];
```

$$x_i \geq \sum_{(i,j) \in E} x_j \quad \forall i \in I$$

↓ OPL ↓

```
forall (i in I)  
    x[i] >= sum(j in A[i])(x[j]);
```

4.1 Kurzeinführung
in die
Graphentheorie

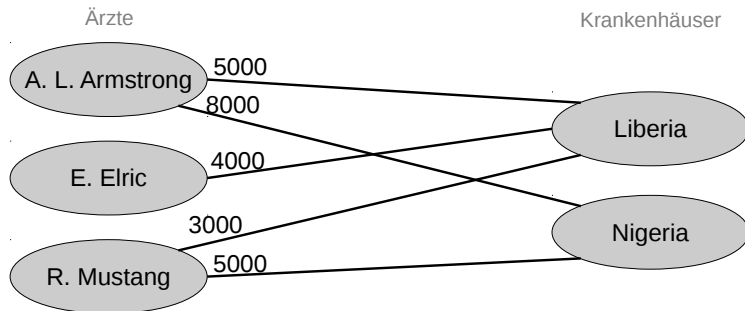
4.2 Abbilden von
Graphen in OPL

4.3 OPL:
Selbstdefinierte
Tupel als
Datenstruktur

4.4 OPL:
Operatoren mit
Bedingungen

4.3 OPL: Selbstdefinierte Tupel als Datenstruktur

Beispiel: Relieve Ärzte



Beispiel: Relieve Ärzte

4 Optimierung von
Graphenproblemen

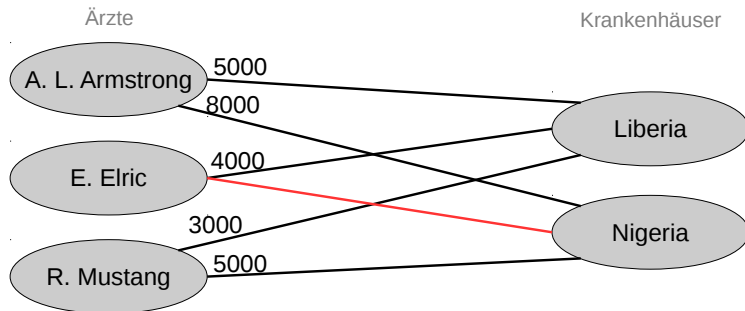
CC-BY-SA
A. Popp

4.1 Kurzeinführung
in die
Graphentheorie

4.2 Abbilden von
Graphen in OPL

4.3 OPL:
Selbstdefinierte
Tupel als
Datenstruktur

4.4 OPL:
Operatoren mit
Bedingungen



Modell: Zuordnungsproblem

Indexmengen:

R Menge der Ressourcen

T Menge der Aufgaben

Parameter:

E Menge der Kanten im Zuordnungsgraphen

c_{rt} Kosten für die Auswahl der Kante $(r, t) \in E$

Entscheidungsvariablen:

x_{rt} Binärvariable, die angibt ob die Kante $(r, t) \in E$ ausgewählt wurde

Modellbeschreibung:

$$\begin{aligned} \min \quad & \sum_{(r,t) \in E} c_{rt} \cdot x_{rt} \\ \text{s.t.} \quad & \sum_{(r,t) \in E} x_{rt} = 1 \quad \forall t \in T \quad (\text{I}) \\ & \sum_{(r,t) \in E} x_{rt} \leq 1 \quad \forall r \in R \quad (\text{II}) \\ & x_{rt} \in \{0, 1\} \quad \forall (r, t) \in E \end{aligned}$$

- 18/25 ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Tupel sind selbstdefinierte Datenstrukturen, die aus Elementen anderer Datenstrukturen bestehen.

Definition eines neuen Tupel-Datentyps

```
tuple Name_der_Tupel-Datenstruktur {
    Datentyp_des_1._Elements Name_des_1._Elements;
    Datentyp_des_2._Elements Name_des_2._Elements;
    ...
}
```

Beispiel: Kanten als Tupel-Datentyp

```
{string} V = {"A", "B", "C"};
tupel edge {
    string start;
    string end;
};
```

4.3 OPL: Selbstdefinierte Tupel als Datenstruktur

20/25 ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Anwendung von Tupel-Datentypen (Variante 1)

4 Optimierung von Graphenproblemen

CC-BY-SA
A. Popp

Knoten und Kanten seien wie oben definiert.

Anwendungsbeispiel

$$\sum_{(r,t) \in E} x_{rt} = 1 \quad \forall t \in T$$

↓ OPL ↓

```
forall(t in T)
  sum(<r,t
```

4.3 OPL: Selbstdefinierte Tupel als Datenstruktur

4.4 OPL: Operatoren mit Bedingungen

4.4 OPL: Operatoren mit Bedingungen

```
forall(Laufindex in Indexmenge : Bedingung)
```

23/25 ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Literale für Wahrheitswerte

true, false

Vergleichsoperatoren für Wahrheitswerte

math. Schreibweise	=	≠	≤	<	≥	>
OPL-Syntax	==	!=	<=	<	>=	>

Logische Verknüpfungen für Wahrheitswerte

math. Schreibweise	\neg	\wedge	\vee	$\underline{\vee}$
OPL-Syntax	!	&&		!=

4.4 OPL: Operatoren mit Bedingungen

Anwendung von Tupel-Datentypen (Variante 2)

4 Optimierung von
Graphenproblemen

CC-BY-SA
A. Popp

Knoten und Kanten seien wie oben definiert.

Anwendungsbeispiel

$$\sum_{(r,t) \in E} x_{rt} = 1 \quad \forall t \in T$$

↓ OPL ↓

```
forall(t in T)
  sum(e in E : e.task == t)(x[e]) == 1;
```

4.1 Kurzeinführung
in die
Graphentheorie

4.2 Abbilden von
Graphen in OPL

4.3 OPL:
Selbstdefinierte
Tupel als
Datenstruktur

4.4 OPL:
Operatoren mit
Bedingungen