

PG 534 - Vehicle Routing Einführung

Markus Chimani & Karsten Klein

LS11, TU Dortmund

PG Ziele

- Framework
 - Branch & Cut, Branch & Price,...
 - Auswählbare Zielfunktionen, Nebenbedingungen, Heuristiken, Ungleichungen, Separationsmethoden, etc.
- Experimentelle Studie
 - auch Praxisdaten

Übersicht

- Grundlagen: Kleine Beispiele
 - LP vs. ILP
 - Lösungsmethoden
- Solver / Framework
- Grosses Beispiel
 - TSP: Formulierung, Cuts
 - SCIP: Grundlagen
 - SCIP Code für TSP
- Organisatorisches
 - Accounts, Webserver, etc. → Freitag

Lineare Programmierung

Formulierung eines Optimierungsproblems mit Variablenvektor $x = (x_1, \dots, x_n)$ über:

- Lineare Kostenfunktion mit Kostenvektor $c = (c_1, \dots, c_n)$

$$\min \sum_{i=1..n} c_i x_i \quad (\text{auch max})$$

- Lineare (Un-)Gleichungen als Nebenbedingungen

$$\sum_{i=1..n} a_i x_i \leq b \quad (\text{auch } \geq, =)$$

Spezialformen: $x_i \geq 0$

Vektor x , der alle Bedingungen erfüllt: *Gültige Lösung*

Vektor x^* mit $cx^* \leq cx \forall$ gültigen x : *Optimallösung*

Lineares Programm

- Ausgeschrieben:

$$\min/\max \quad c_1x_1 + \dots + c_nx_n$$

$$\text{sto} \quad a_{11}x_1 + \dots + a_{1n}x_n \leq b_1$$

...

$$a_{m1}x_1 + \dots + a_{mn}x_n \leq b_m$$

x_i unbeschränkte Variablen, oder $x_i \geq 0$

- *Standardform:*

	Ursprünglich	Transformiert
Zielfunktion	$\max cx$	$\min -cx$
Gleichung	$a_jx \leq b_j$	$a_jx + s = b_j, \quad s \geq 0$
Unbeschränkte Variable	x_i	$x_i^+ - x_i^-$ mit $x_i^+, x_i^- \geq 0$

Lineares Programm

- Kompaktschreibweise mit Matrix $A \in \mathbb{R}^{m,n}$,
Vektoren $b \in \mathbb{R}^m$, $c, x \in \mathbb{R}^n$:

$$\min \{c^T x \mid Ax = b, x \geq 0\}$$

Beispiel: Diätproblem

x_i : Verschiedene Nahrungsmittel

b_i : Idealversorgung mit Nährstoffen (z.B. Vitamine,...)

c_i : Kosten eines Nahrungsmittels

Ziel: Günstigste Diät mit ausreichender Versorgung

Nahrungsmittel	Brennwert/kcal	Vitamin C/mg	Preis/€
Brot (500g)	1230	0	2,99
Orangensaft (1l)	560	300	2,50
Bedarf:	≥ 2000	≥ 60	???

$$\min 2,99x_1 + 2,50x_2$$

$$\text{sto } 1230x_1 + 560x_2 \geq 2000$$

$$300x_2 \geq 60$$

$$x_1, x_2 \geq 0$$

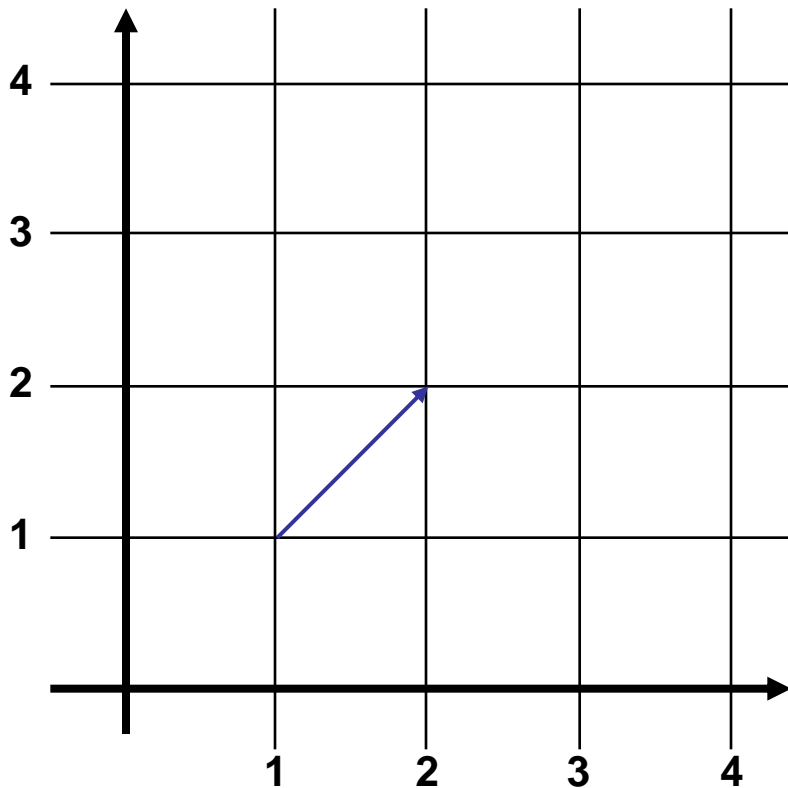
Lineare Programmierung

Komplexität: Lineare Optimierungsprobleme sind in polynomieller Zeit lösbar

- Methoden:
 - Innere Punkte Methode
 - Ellipsoidmethode
 - Simplexmethode (worst-case exponentiell)

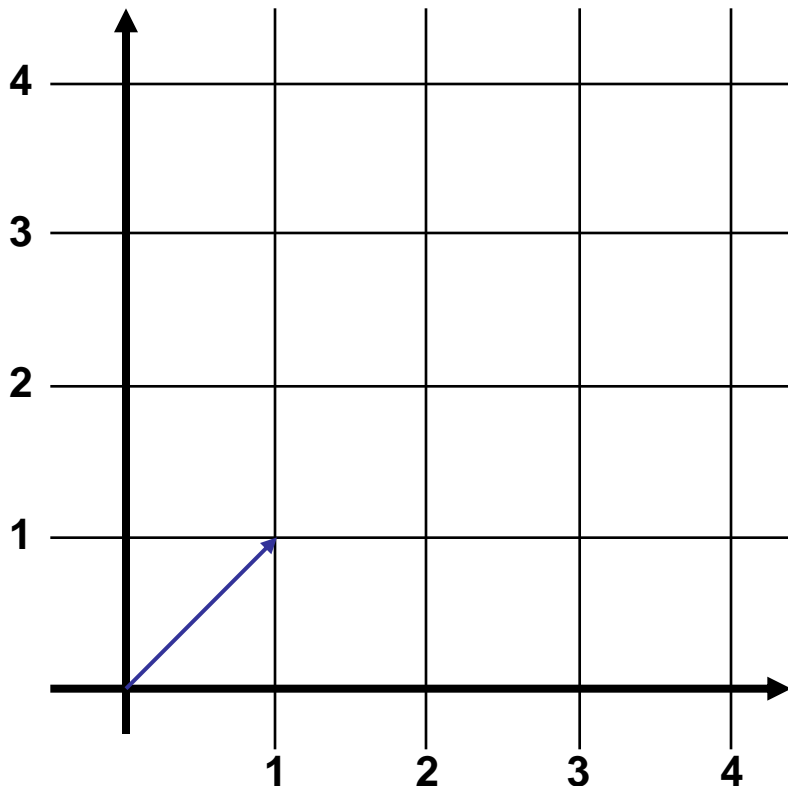
Geometrische Interpretation

- Vektoren: Richtung, Länge
- Kosten $x_1 + x_2 \rightarrow$ Vektor $c = (1,1)$



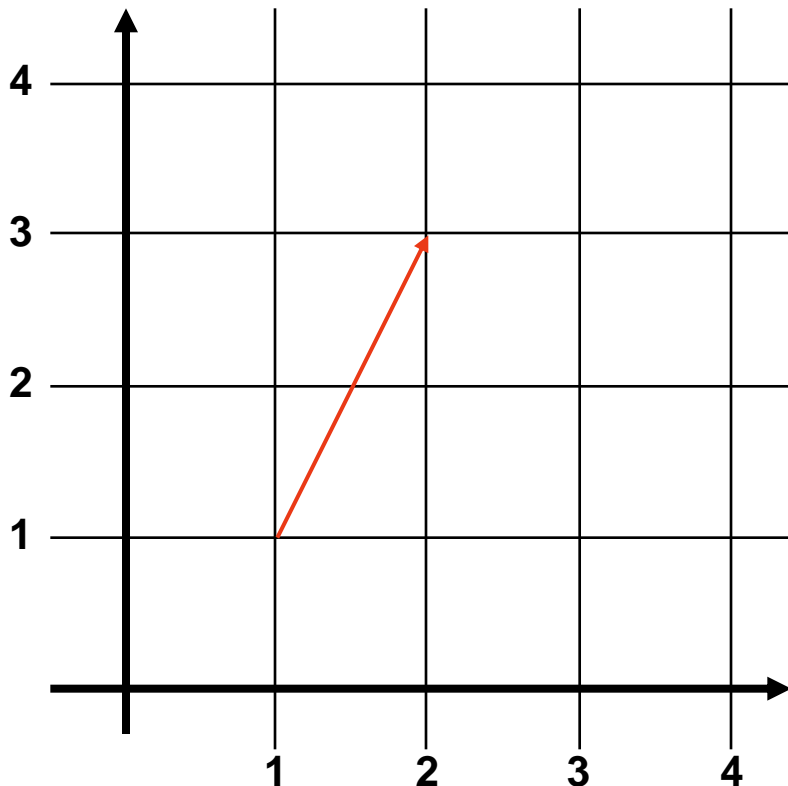
Geometrische Interpretation

- Vektoren: Richtung, Länge
- Kosten $x_1 + x_2 \rightarrow$ Vektor $c = (1,1)$



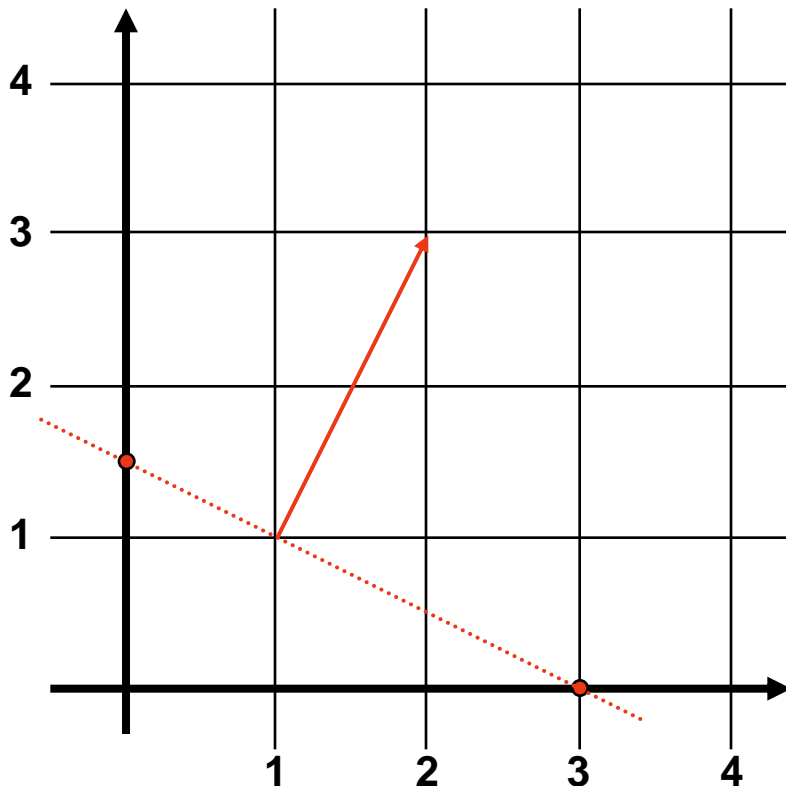
Geometrische Interpretation

- Vektoren: Richtung, Länge
- Kosten $x_1 + x_2 \rightarrow$ Vektor $c = (1,1)$
- Gleichung $x_1 + 2x_2 = 3 \rightarrow$ Vektor $a = (1,2)$



Geometrische Interpretation

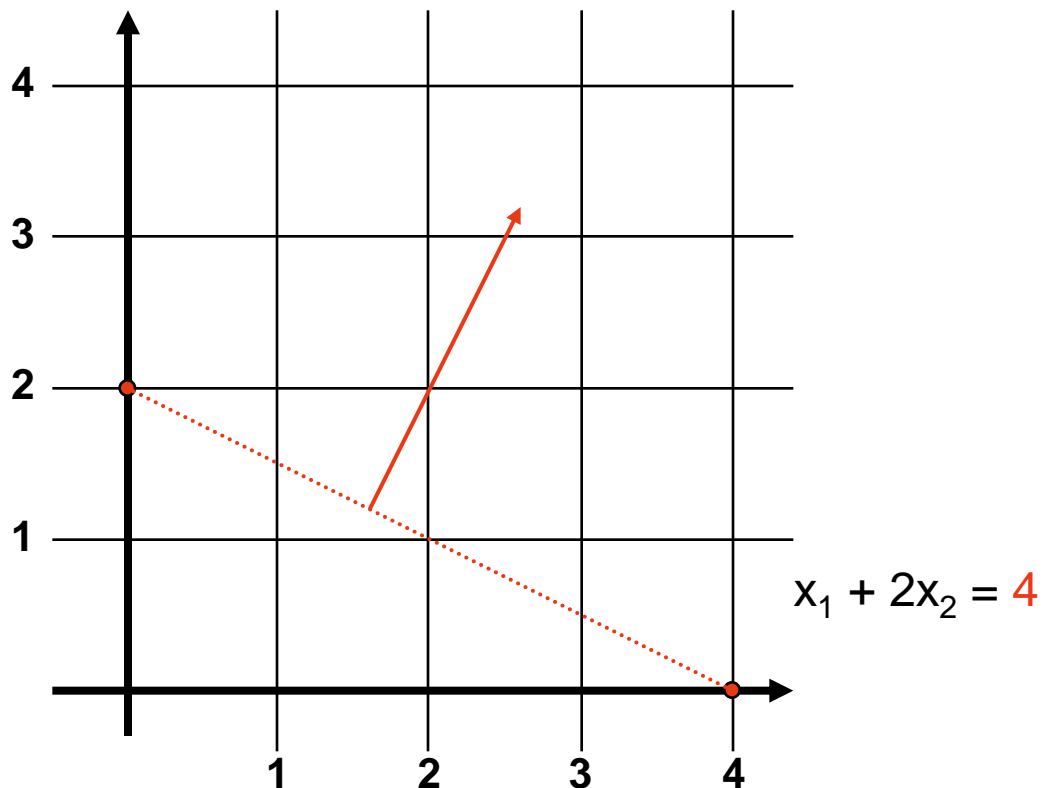
- Vektoren: Richtung, Länge
- Kosten $x_1 + x_2 \rightarrow$ Vektor $c = (1,1)$
- Gleichung $x_1 + 2x_2 = 3 \rightarrow$ Vektor $a = (1,2)$



$$x_2 = \frac{1}{2}(3 - x_1)$$

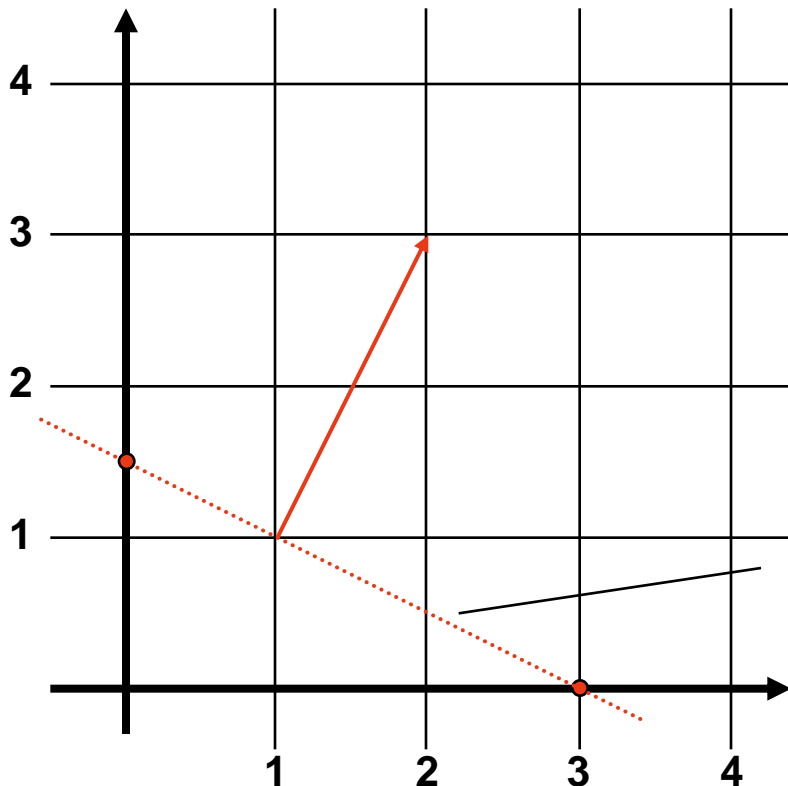
Geometrische Interpretation

- Vektoren: Richtung, Länge
- Kosten $x_1 + x_2 \rightarrow$ Vektor $c = (1,1)$
- Gleichung $x_1 + 2x_2 = 3 \rightarrow$ Vektor $a = (1,2)$



Geometrische Interpretation

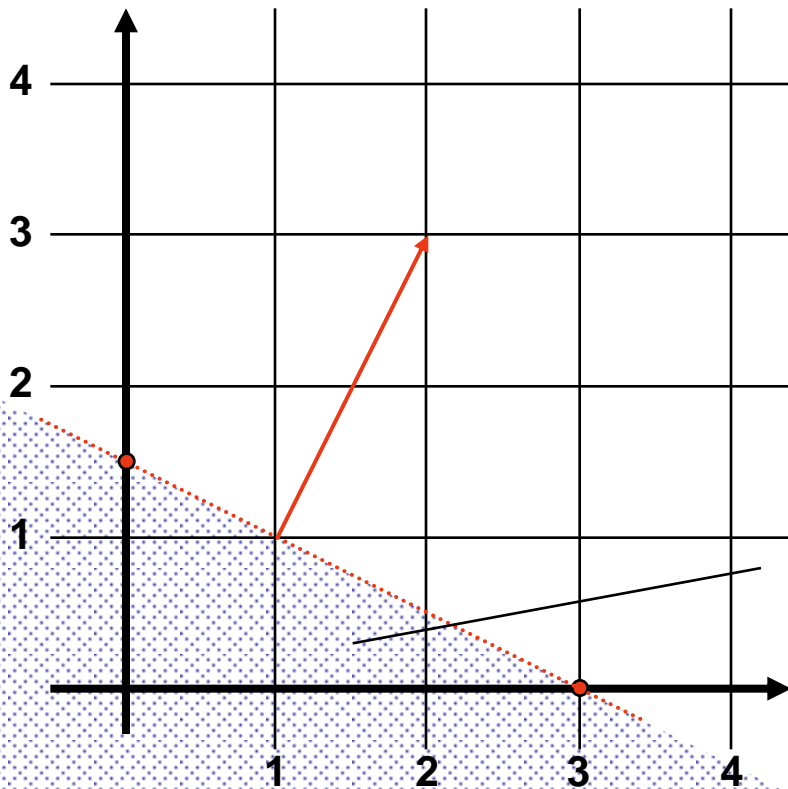
- Vektoren: Richtung, Länge
- Kosten $x_1 + x_2 \rightarrow$ Vektor $c = (1,1)$
- Gleichung $x_1 + 2x_2 = 3 \rightarrow$ Vektor $a = (1,2)$



Hyperebene, allgemein
 $\{x \in \mathbb{R}^n \mid ax = b\}$,
 $n-1$ dimensional

Geometrische Interpretation

- Vektoren: Richtung, Länge
- Kosten $x_1 + x_2 \rightarrow$ Vektor $c = (1,1)$
- Gleichung $x_1 + 2x_2 = 3 \rightarrow$ Vektor $a = (1,2)$

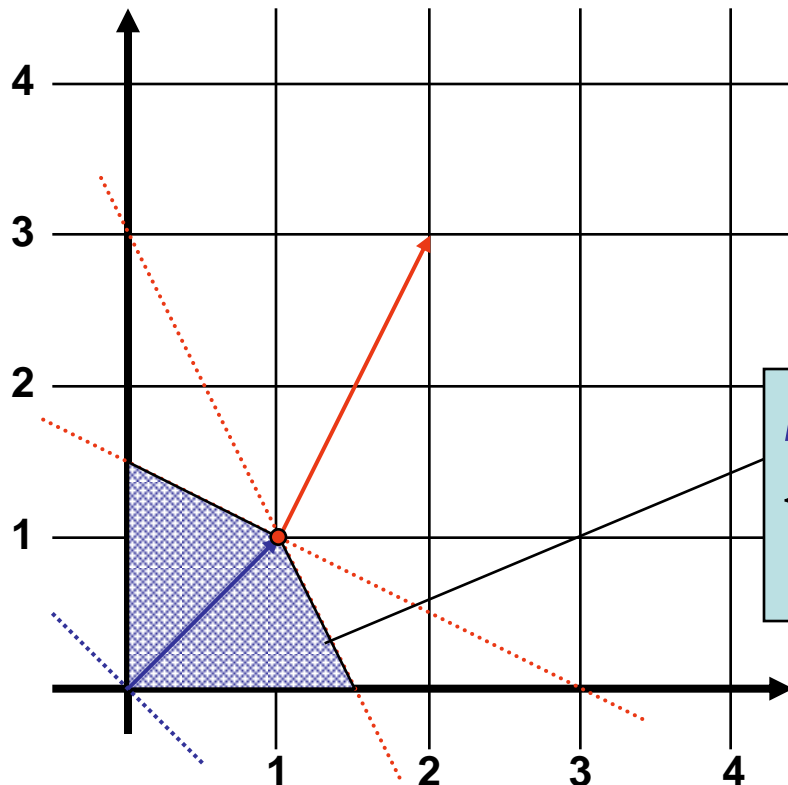


Ungleichung $x_1 + 2x_2 \leq 3$?

Halbraum, allgemein
 $\{x \in \mathbb{R}^n \mid ax \leq b\}$,
n dimensional

Geometrische Interpretation

- Vektoren: Richtung, Länge
- Kosten $x_1 + x_2 \rightarrow$ Vektor $c = (1,1)$
- Gleichung $x_1 + 2x_2 = 3 \rightarrow$ Vektor $a = (1,2)$



Polyeder, allgemein
 $\{x \in \mathbb{R}^n \mid Ax \leq b\}$,

$$\begin{aligned} \max \quad & x_1 + x_2 \\ & x_1 + 2x_2 \leq 3 \\ & 2x_1 + x_2 \leq 3 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Optimallösung (1, 1),
Wert 2

Intermezzo

Interactive CPLEX: LP1

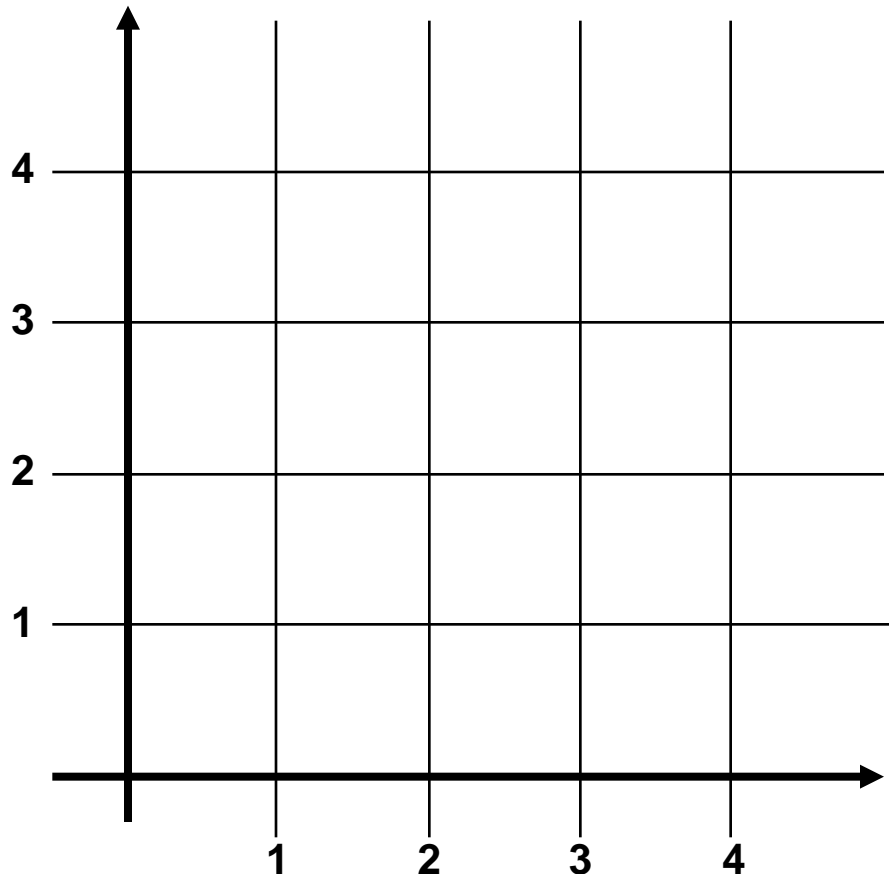
Lineare Programmierung

- Lösung im letzten Beispiel war ganzzahlig!

Was, wenn nicht, aber Ganzzahligkeit erforderlich?

Integer Linear Programming

- ILP: Ganzzahligkeit der Lösung verlangt (diskret)
- Mixed ILP: Falls nur für einige x_i Ganzzahligkeit



$$\max x_1 + x_2$$

$$\text{sto } 14x_1 - 18x_2 \geq -9$$

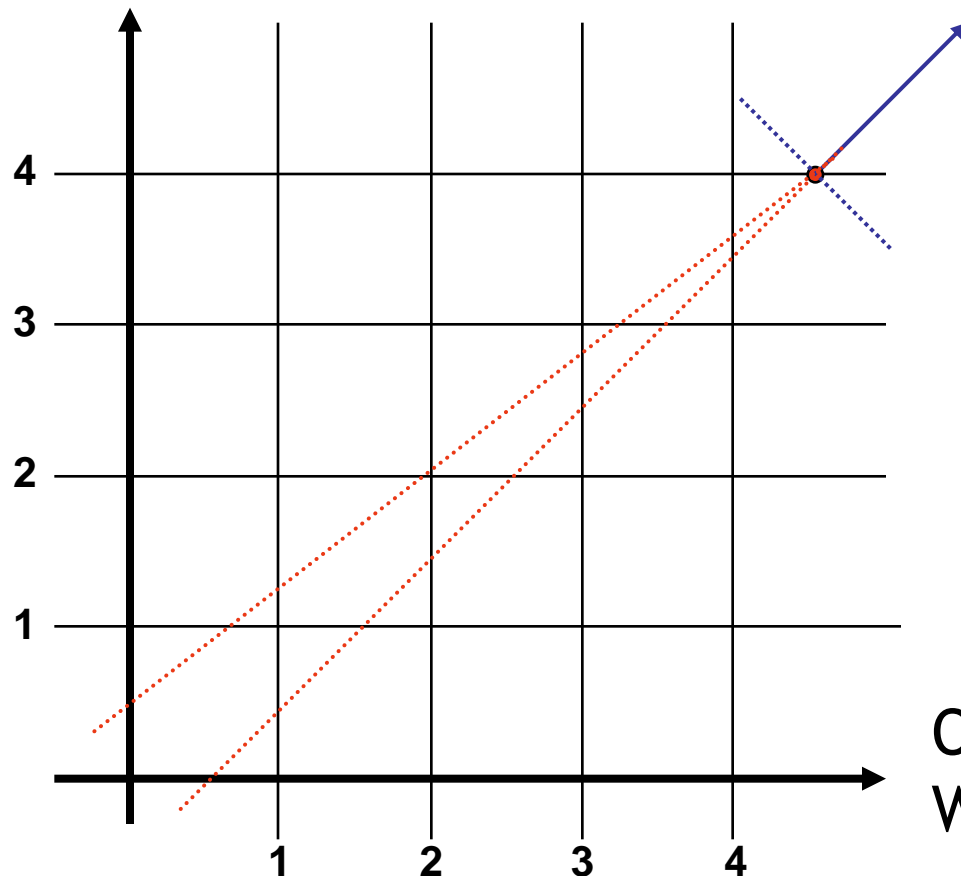
$$2x_1 - 2x_2 \leq 1$$

$$x_1, x_2 \geq 0$$

Interactive **CPLEX**: LP2

Integer Linear Programming

- ILP: Ganzzahligkeit der Lösung verlangt (diskret)
- Mixed ILP: Falls nur für einige x_i Ganzzahligkeit



$$\max x_1 + x_2$$

$$\text{sto } 14x_1 - 18x_2 \geq -9$$

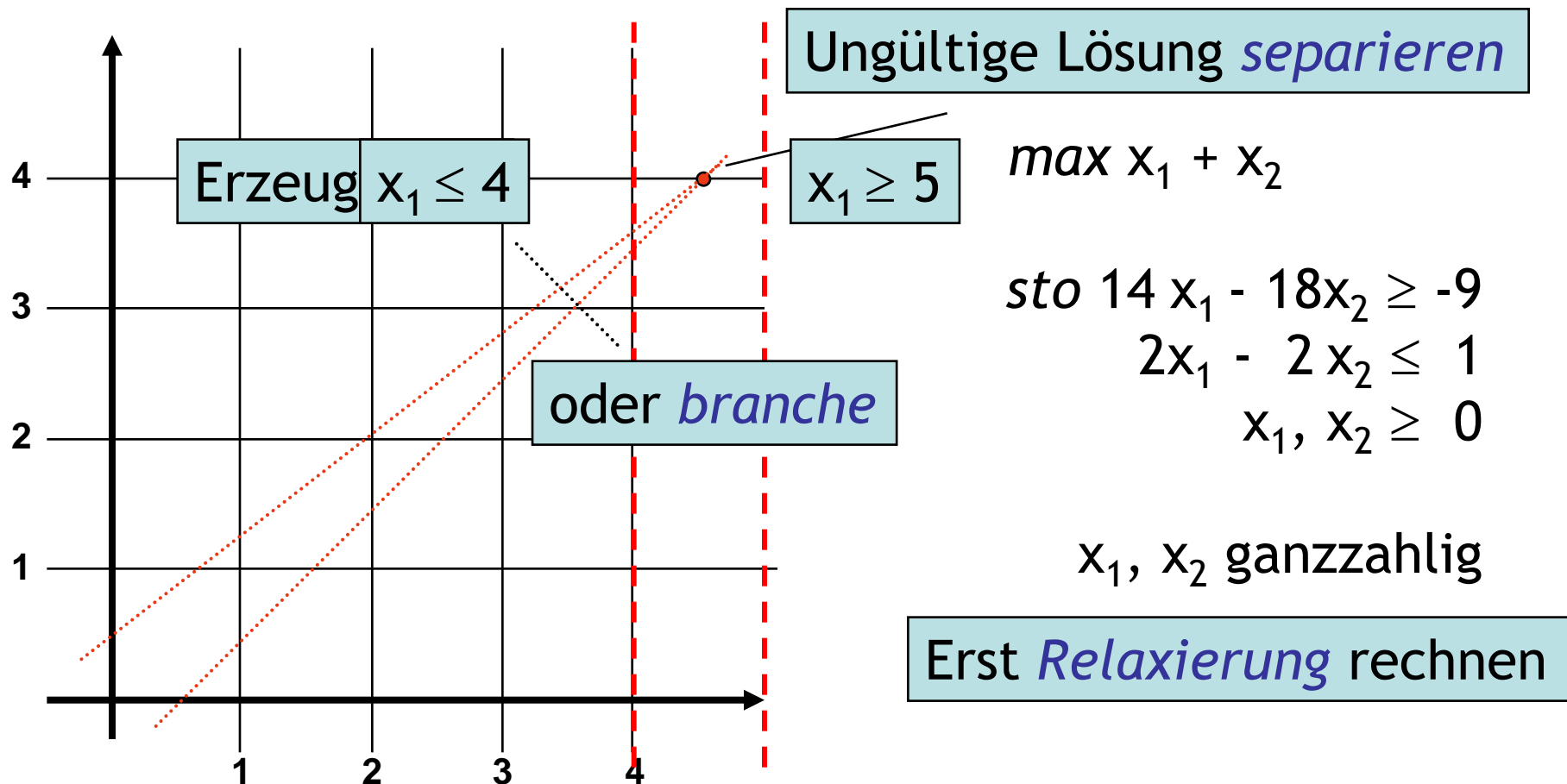
$$2x_1 - 2x_2 \leq 1$$

$$x_1, x_2 \geq 0$$

Optimallösung (4.5, 4),
Wert 8,5

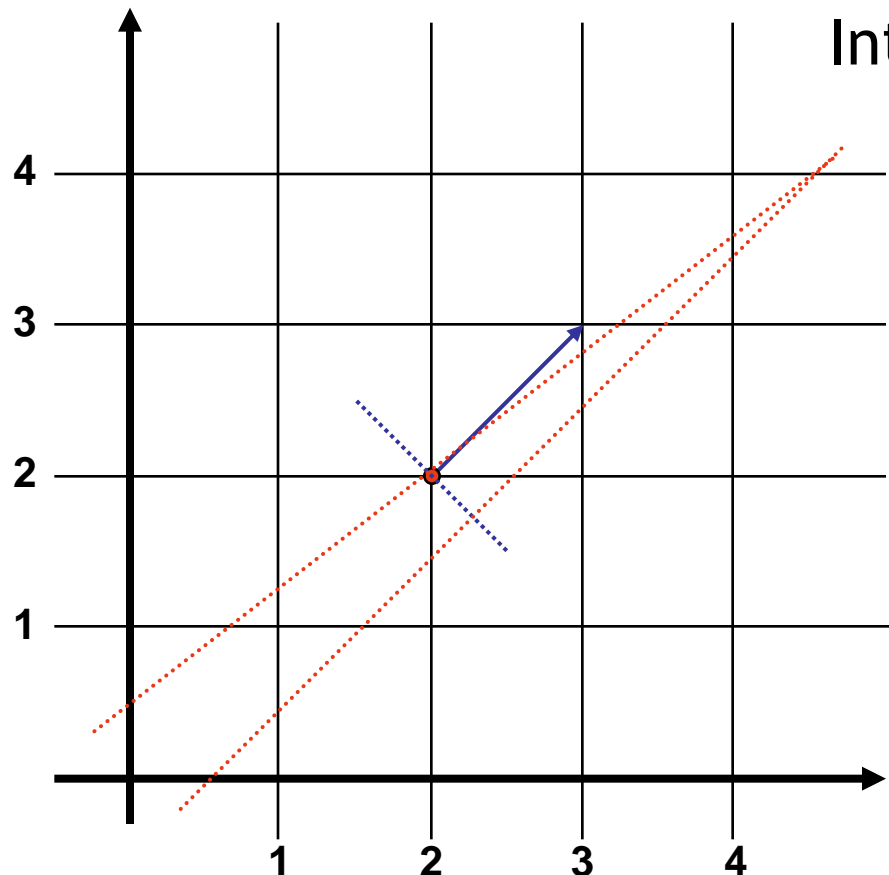
Integer Linear Programming

- ILP: Ganzzahligkeit der Lösung verlangt (diskret)
- Mixed ILP: Falls nur für einige x_i Ganzzahligkeit



Integer Linear Programming

- ILP: Ganzzahligkeit der Lösung verlangt (diskret)
- Mixed ILP: Falls nur für einige x_i Ganzzahligkeit



Interactive **CPLEX: LP2ILP**

$$\max x_1 + x_2$$

$$\text{sto } 14x_1 - 18x_2 \geq -9$$

$$2x_1 - 2x_2 \leq 1$$

$$x_1, x_2 \geq 0$$

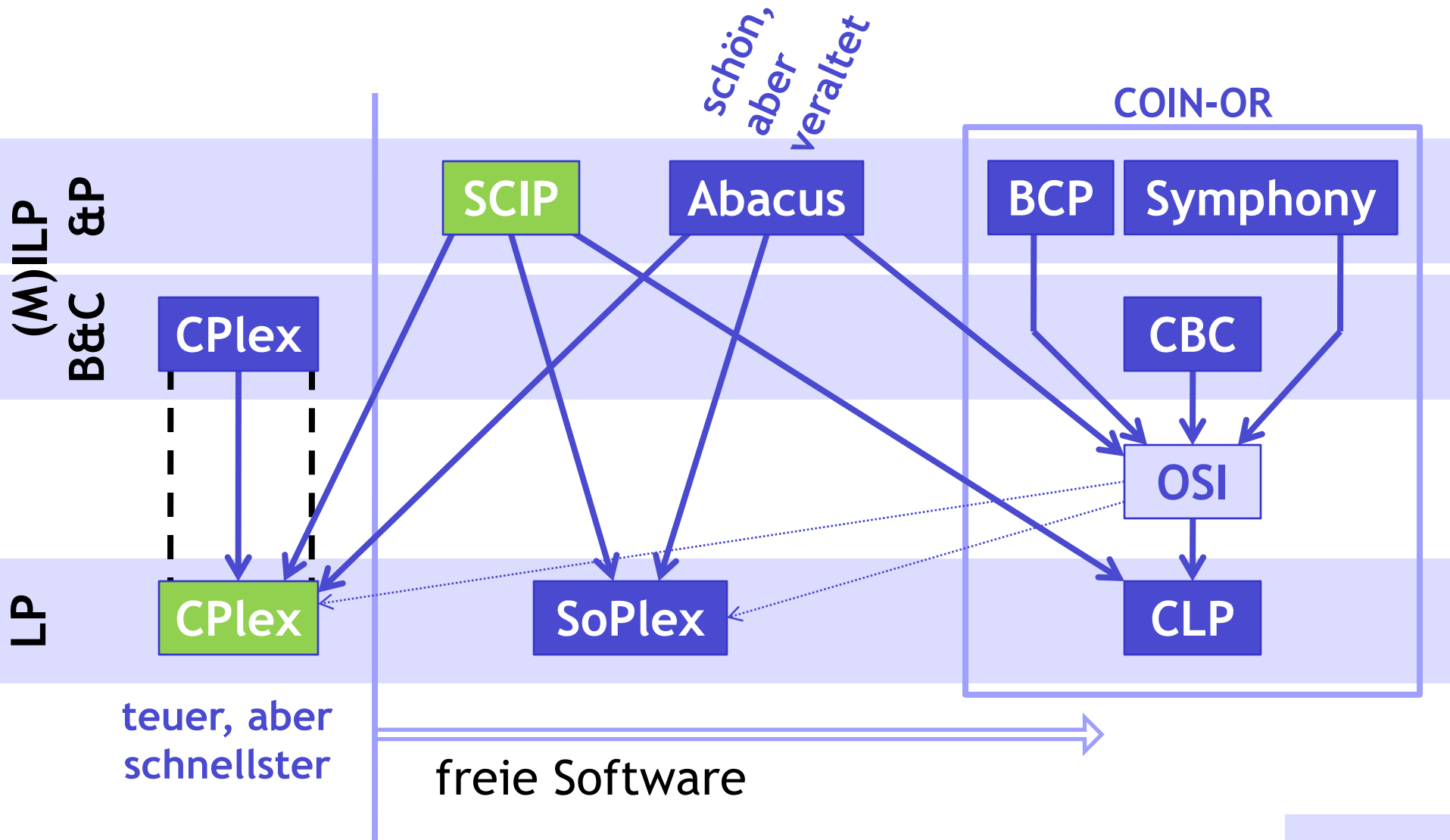
x_1, x_2 ganzzahlig

Optimallösung (2, 2), Wert 4

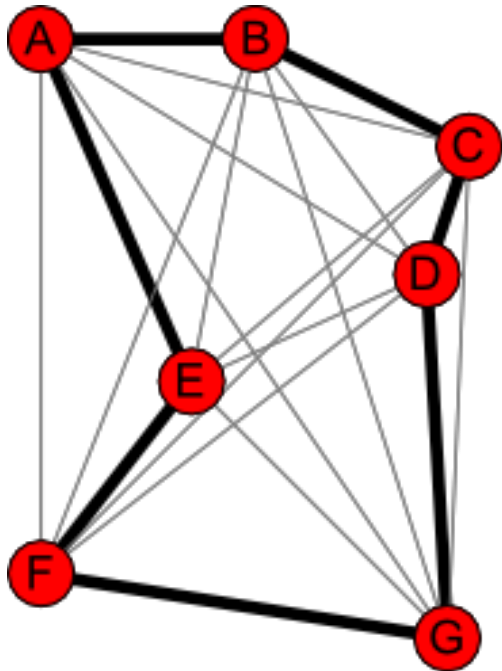
Integer Linear Programming

- ILP ist NP-vollständig, sogar im Spezialfall $x_i \in \{0, 1\}$
- Stärke der Formulierung: Viel wichtiger als bei LP!
- Hier nicht Anzahl Variablen und Nebenbedingungen entscheidend, sondern wie genau Nebenbedingungen die konvexe Hülle der gültigen ganzzahligen Lösungen beschreiben!

(Einige) Solver / Frameworks



TSP: Formulierung (1)



Gegeben: n Städte: $S = \{A, B, C, D, \dots\}$
Distanzen zw. Städten: $d(A, B)$

Gesucht: Kürzeste Rundtour durch alle Städte

Variablen: $x_{v,w} \in \{0, 1\} \quad \forall v, w \in S$

Zielfunkt.: $\min \sum_{v,w \in S} d(v,w) \cdot x_{v,w}$

TSP: Formulierung (2)

Variablen: $x_{v,w} \in \{0,1\} \quad \forall v,w \in S$

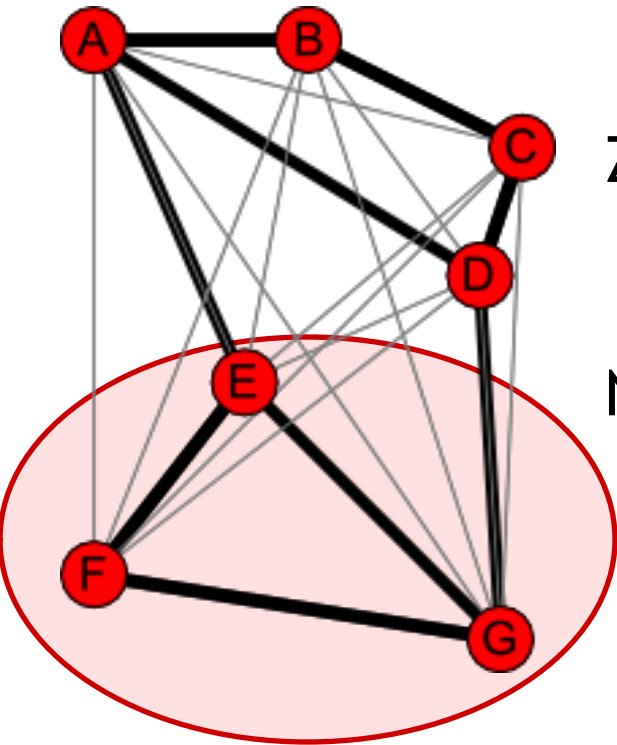
Zielfunkt.: $\min \sum_{v,w \in S} d(v,w) \cdot x_{v,w}$

Nebenbedingungen:

$$\sum_{w \in S} x_{v,w} = 2 \quad \forall v \in S$$

$$\sum_{v \in T} \sum_{w \notin T} x_{v,w} \geq 2 \quad \forall T \subset S$$

exponentiell viele!



TSP: Lösungsmethoden

Berechne Heuristik:
obere Schranke O



Starte mit einfacherem
linearen Programm P

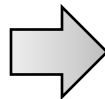


$$\min \sum_{v,w \in S} d(v,w) \cdot x_{v,w}$$

$$0 \leq x_{v,w} \leq 1 \quad \forall v,w \in S$$

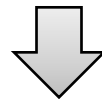
$$\sum_{w \in S} x_{v,w} = 2 \quad \forall v \in S$$

~~$$\sum_{v \in T} \sum_{w \notin T} x_{v,w} \geq 2 \quad \forall T \subset S \quad (*)$$~~



Subproblem

1. Löse $P \rightarrow L$ (polynomiell, CPLEX)
2. Ist L gültig? $O \leftarrow \min(O, L)$!
3. Ist $L \geq O$? Subproblem beenden.
4. Heuristik auf Basis von L
→ ggf. neues O
5. Finde verletzte Ungleichung (*)?
Füge zu P hinzu & gehe zu 1.



Subproblem

$$x_{v,w} = 0$$



Subproblem

$$x_{v,w} = 1$$



Alle Subprobleme beendet?
 O ist Optimum

SCIP: Übersicht

- SCIP = Solving Constraint Integer Programs
- <http://scip.zib.de> , Doxygen-Docs (sehr gut & hilfreich)
- @home: download, compile via (default: gcc, linux)
make LPS=? READLINE=false ZLIB=false ZIMPL=false
spx (SoPlex), clp (CLP)
- @uni: installiert&compiliert (LPS=cpx / CPlex) in
/home/plug/scip-1.1.0
- Examples: TSP, VRP,...
/home/plug/scip-1.1.0/examples/...

TSP mit SCIP: (Einige) Wichtige Dateien

■ Wesentliche Dateien

`mymain.cpp`

- `main()`
- Aufbauen der SCIP-Strukturen
- Registrieren der Plugins
(Reader, Separierung, Heuristiken, Pricer,...)

`myreader.h`

`myreader.cpp`

- Einlesen des Problems
- Erstellen des initialen LPs

`mysubtour.h`

`mysubtour.cpp`

- Separierung der Subtour-Constraints

mymain.cpp

```
#include <iostream>
#include "objscip/objscip.h"
#include "objscip/objscipdefplugins.h"
#include "myreader.h"
#include "mysubtour.h"

SCIP_RETCODE runSCIP(int argc, char** argv) {
    SCIP* scip = NULL;
    SCIP_CALL( SCIPcreate(&scip) );
    SCIP_CALL( SCIPincludeObjReader(scip, new MyReader(scip), TRUE) );
    SCIP_CALL( SCIPincludeObjConshdlr(scip, new MySubtour(), TRUE) );
    SCIP_CALL( SCIPincludeDefaultPlugins(scip) );
    SCIP_CALL( SCIPprocessShellArguments(scip, argc, argv, "parameter.txt" );
    SCIP_CALL( SCIPfree(&scip) );
    return SCIP_OKAY;
}

int main(int argc, char** argv) {
    SCIP_RETCODE retcode = runSCIP(argc, argv);
    if( retcode != SCIP_OKAY )
        SCIPprintError(retcode, stderr);
}
```

```
SCIP_CALL( ???() );
SCIP_RETCODE ret = ???();
if( ret != SCIP_OKAY)
    return ret;
```

```
SCIPprocessShellArguments
```

```
...
SCIPsolve(scip);
...
```

TSP mit SCIP: (Einige) Wichtige Dateien

■ Wesentliche Dateien

`mymain.cpp`



- `main()`
- Aufbauen der SCIP-Strukturen
- Registrieren der Plugins (Reader, Separierung, Heuristiken, Pricer,...)

`myreader.h`

`myreader.cpp`

- Einlesen des Problems
- Erstellen des initialen LPs

`mysubtour.h`

`mysubtour.cpp`

- Separierung der Subtour-Constraints

myreader.h

```
#include "objscip/objscip.h"

class MyReader : public scip::ObjReader {
public:
    MyReader() : scip::ObjReader("myreader", "file reader for TSP files", "tsp") {}
    virtual ~MyReader() {}

    virtual SCIP_RETCODE scip_free( SCIP* scip, SCIP_READER* reader ) {
        return SCIP_OKAY;
    }
    virtual SCIP_RETCODE scip_write( ..., SCIP_RESULT* result ) {
        *result = SCIP_DIDNOTRUN;
        return SCIP_OKAY;
    }
    virtual SCIP_RETCODE scip_read( SCIP* scip, SCIP_READER* reader,
                                   const char* filename, SCIP_RESULT* result );
};
```

myreader.cpp

```
#include "myprobddata.h" // class MyProbData : public scip::ObjProbData { ... };

SCIP_RETCODE MyReader::scip_read( SCIP* scip, SCIP_READER* reader,
                                   const char* filename, SCIP_RESULT* result ) {
    *result = SCIP_DIDNOTRUN;
    MyProbData* graph = loadGraphFromFile(filename); // assume function exists
    if(graph == NULL) return SCIP_READERROR;
    SCIP_CALL( SCIPcreateObjProb(scip, "MyProblemData", graph, TRUE) );

    // add variables
    for( alle kanten edge von graph ) {
        SCIP_VAR* var;
        stringstream name;
        name << "x_" << edge->id;
        SCIP_CALL( SCIPcreateVar(scip, &var, name.str().c_str(), 0, 1, edge->length,
                                SCIP_VARTYPE_BINARY, TRUE, FALSE, NULL, NULL, NULL, NULL) );
        edge->correspondingVar = var;
        SCIP_CALL( SCIPaddVar(scip, var) );
    }

    *** nächste Folie hier: add constraints ***

    *result = SCIP_SUCCESS;
    return SCIP_OKAY;
}
```

0, 1, edge->length

untere Schranke: 0,
obere Schranke: 1,
Koeffizient in Zielfunktion

myreader.cpp

$$\sum_{w \in S} x_{v,w} = 2 \quad \forall v \in S$$

```
// add degree constraints
for( alle knoten node von graph ) {
    SCIP_CONS* cons;
    stringstream name;
    name << "degCon_" << node->id;
    SCIP_CALL( SCIPcreateConsLinear(scip, &cons, name.str().c_str(),
        0, NULL, NULL, 2.0, 2.0,
        TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE) );

    for( alle kanten edge adjazent zu node )
        SCIP_CALL( SCIPaddCoefLinear(scip, cons,
            edge->correspondingVar, 1.0) );

    SCIP_CALL( SCIPaddCons(scip, cons) );
    SCIP_CALL( SCIPreleaseCons(scip, &cons) );
}

// add subtour constraint handler
SCIP_CONS* cons;
SCIP_CONSDATA* consdata; SCIP_CALL( SCIPallocMemory( scip, &consdata) );
consdata->graph = graph;
SCIP_CALL( SCIPcreateCons(scip, cons, name, SCIPfindConshdlr(scip, "subtour"),
    consdata, FALSE, TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE) );
SCIP_CALL( SCIPaddCons(scip, cons) );
SCIP_CALL( SCIPreleaseCons(scip, &cons) );
```

0, NULL, NULL

0 Variablen,
leeres Array für Variablen,
leeres Array für Koeffizienten

2.0, 2.0

linke und rechte Schranke
 $2.0 \leq \text{constraint} \leq 2.0$

TSP mit SCIP: (Einige) Wichtige Dateien

■ Wesentliche Dateien

`mymain.cpp`



- `main()`
- Aufbauen der SCIP-Strukturen
- Registrieren der Plugins (Reader, Separierung, Heuristiken, Pricer,...)

`myreader.h`

`myreader.cpp`



- Einlesen des Problems
- Erstellen des initialen LPs

`mysubtour.h`

`mysubtour.cpp`

- Separierung der Subtour-Constraints

mysubtour.h

```
class MySubtour : public scip::ObjConshdlr {  
public:
```

```
    MySubtour() : scip::ObjConshdlr("subtour", ...) {}  
    virtual ~MySubtour() {}
```

```
    // ist eine lösung gültig?
```

```
    virtual SCIP_RETCODE scip_check(..., SCIP_SOL* sol, ... );
```

```
    virtual SCIP_RETCODE scip_enfolp( ... );
```

```
    virtual SCIP_RETCODE scip_enfops( ... );
```

enfo = enforce

lp = fractional solution
ps = pseudo solution

Für Gültigkeit

```
    // rundungsimplicationen von constraints
```

```
    virtual SCIP_RETCODE scip_lock( ... );
```

```
    // separate
```

```
    virtual SCIP_RETCODE scip_sepalp( ... );
```

```
    virtual SCIP_RETCODE scip_sepasol(..., SCIP_SOL* sol, ... );
```

Für Geschwindigkeit

(*), (**)

Meist jeweils idente bzw. sehr ähnliche Implementierungen

$$\sum_{v \in T} \sum_{w \notin T} x_{v,w} \geq 2 \quad \forall T \subset S$$

```
SCIP_RETCODE ConshdlrSubtour::scip_sepalp(  
    SCIP* scip, ..., SCIP_CONS** cons, ..., SCIP_RESULT* result) {  
    MyProbData* graph = SCIPconsGetData( cons[0] )->graph;  
  
    for( alle kanten edge von graph )  
        edge->capacity = SCIPgetSolVal(scip, NULL, edge->correspondingVar);  
  
    *result = SCIP_DIDNOTFIND;  
    if( mincut(graph) < 2.0 ) { // assume function exists  
        SCIP_ROW* row;  
        SCIP_CALL( SCIPcreateEmptyRow(scip, &row, "sep",  
            2.0, SCIPinfinity(scip), FALSE, FALSE, TRUE) );  
        2.0 ≤ constraint ≤ ∞  
        SCIP_CALL( SCIPcacheRowExtensions(scip, row) );  
        for( alle kanten edge von graph )  
            if( edge->incidentNode1->mark != edge->incidentNode2->mark )  
                SCIP_CALL( SCIPaddVarToRow(scip, row, edge->correspondingVar, 1.0) );  
        SCIP_CALL( SCIPflushRowExtensions(scip, row) );  
        SCIP_CALL( SCIPaddCut(scip, NULL, row, FALSE) );  
        SCIP_CALL( SCIPreleaseRow(scip, &row) );  
        *result = SCIP_SEPARATED;  
    }  
    return SCIP_OKAY;  
}
```

mincut(graph)

- Berechnet minimalen Schnitt mit Kantenkapazitäten (edge->capacity)
- Markiert die Knoten der zwei Partitionen mit 0 bzw. 1 (node->mark)

NULL

explizite Lösung, oder (wenn NULL) derzeitige LP/Pseudo Lösung

Während der Seminarphase:

<http://scip.zib.de>, Doxygen-Doku: HowTo

Sodala... das wars für heute!

Bis Freitag!