Lehrstuhl für Informatik I
Algorithmen und Komplexität
Universität Würzburg

Würzburg, den 17. April 2026

Prof. Dr. Alexander Wolff
Antonio Lauerbach, M. Sc.

Einführung in PuLP

Algorithmische Graphentheorie (Sommer 2026)

Inhaltsverzeichnis

1 Überblick

Zum Lösen von Linearen Programmen (LPs) werden wir in den Übungen die Python-Bibliothek PuLP[1] verwenden.

In diesem Dokument werden wir eine kurze Einführung in die Installation und Nutzung von PuLP geben. Es ist keine vollständige Dokumentation, sondern soll als Einführung dienen, damit Sie die Übungen bearbeiten können. Weitere Informationen zu PuLP finden Sie in der offiziellen Dokumentation[2].

Wir empfehlen Ihnen, die Übungen auf Ihrem eigenen Computer zu bearbeiten, Sie können aber auch die CIP-Pool-Rechner der Fakultät[3] nutzen. Näheres dazu finden Sie in Abschnitt 3.

[1] `https://coin-or.github.io/pulp/main/includeme.html`
[2] `https://coin-or.github.io/pulp/main/index.html`
[3] `https://www.mathematik-informatik.uni-wuerzburg.de/dienste/computerpool/`

2 Python Setup und Installation

Um mit PuLP zu arbeiten, muss Python auf Ihrem Computer installiert sein. Aktuell ist dazu eine Python-Version von 3.9 oder höher erforderlich. Um Probleme zu vermeiden, empfehlen wir, dass Sie keine neuere Version als 3.14 verwenden.

2.1 Virtual Environment

Falls Sie die CIP-Pool-Rechner unter NixOS (Linux) nutzen, müssen Sie zunächst ein Virtual Environment (venv)[4] anlegen, bevor Sie PuLP installieren können. Falls Sie Ihren eigenen Rechner nutzen, ist dies optional, aber dennoch empfehlenswert, um die Pakete, die Sie für die Übungen installieren, von anderen Python-Projekten zu trennen.

Um ein Virtual Environment zu erstellen, führen Sie folgenden Befehl in Ihrem Terminal aus, wobei Sie `agt-venv` durch einen Pfad Ihrer Wahl ersetzen können:

```
python -m venv agt-venv
```

Dieser Befehl erstellt ein neues Virtual Environment in dem Verzeichnis `agt-venv`.

Aktivieren des Virtual Environments Um das Virtual Environment zu aktivieren, führen Sie unter Linux oder macOS folgenden Befehl aus, wobei Sie wieder `agt-venv` durch den Pfad ersetzen, den Sie zuvor gewählt haben:

```
source agt-venv/bin/activate
```

Unter Windows können Sie das Virtual Environment mit folgendem Befehl aktivieren:

```
agt-venv\Scripts\activate
```

Nach der Aktivierung des Virtual Environments sollte in der Regel der Name des Environments (z. B. `agt-venv`) in Ihrem Terminal-Prompt angezeigt werden.

Um das Virtual Environment nach getaner Arbeit zu verlassen, können Sie einfach den folgenden Befehl eingeben:

```
deactivate
```

[4] `https://docs.python.org/3/tutorial/venv.html`

2.2 PuLP Installation

Um PuLP zu installieren, können Sie den Python-Paketmanager pip verwenden. Führen Sie folgenden Befehl in Ihrem Terminal aus, um PuLP zu installieren:

```
python -m pip install pulp
```

Dieser Befehl installiert die neueste Version von PuLP. Sobald die Installation abgeschlossen ist, können Sie PuLP in Ihren Python-Skripten verwenden.

Gurobi als Solver PuLP kommt direkt mit dem Solver CBC[5], wir empfehlen allerdings Gurobi. Dieser kann gleich zusammen mit PuLP installiert werden, indem Sie den folgenden Befehl ausführen:

```
python -m pip install pulp[gurobi]
```

Alternativ können Sie zunächst PuLP installieren und anschließend Gurobi installieren, indem Sie den folgenden Befehl ausführen:

```
python -m pip install gurobipy
```

Gurobi ist ein kommerzieller Solver, jedoch kann man ohne Lizenz kleine Programme (bis zu 2000 Variablen und Constraints) lösen. Dies ist für die Übungen ausreichend. Als Student können Sie sich aber auch eine kostenlose akademische Lizenz („Named-User License“) holen, mit der Sie größere Programme lösen können. Weitere Informationen dazu finden Sie auf der Gurobi-Website[6].

3 CIP-Pool

Die CIP-Pool-Rechner können vor Ort, in den Räumen A001 und A002 im Infogebäude, im Raum E40 im Rechenzentrum sowie im Raum 01.103 im Bibliotheks- und Seminarzentrum genutzt werden. Beim Booten eines Rechners können Sie zwischen Windows und Linux (NixOS) wählen. Wir empfehlen Linux zu nutzen.

3.1 Zugriff über SSH

Sie können auch per SSH auf die CIP-Pool-Rechner zugreifen. Dazu müssen sie sich zunächst im Netzwerk der Universität befinden.

[5] Dieser Solver scheint auf den CIP-Pool-Rechnern nicht zu funktionieren.

[6] https://www.gurobi.com/academia/academic-program-and-licenses/

Hochschulnetz Falls Sie an der Uni sind und das eduroam WLAN nutzen, befinden Sie sich automatisch im Hochschulnetz. Andernfalls können Sie sich per VPN mit dem Hochschulnetz verbinden. Mehr Details dazu finden Sie auf den Seiten des Rechenzentrums[7].

SSH-Verbindung zum Sprungbrettrechner Wenn Sie im Hochschulnetz sind, müssen Sie sich zunächst mit dem Sprungbrettrechner des CIP-Pools verbinden. Dies funktioniert über folgenden Befehl, wobei Sie `s000000` durch Ihre eigene sNummer ersetzen:

```
ssh s000000@cipgate.informatik.uni-wuerzburg.de
```

Anschließend müssen Sie ihr Passwort eingeben.

SSH-Verbindung zu einem CIP-Pool Rechner Wenn Sie sich mit dem Sprungbrettrechner des Computerpools verbunden haben, wird Ihnen eine Liste von eingeschalteten Rechnern der CIP-Pools (und wie viele Nutzer dort angemeldet sind) angezeigt. Sollte die Liste leer sein, ist gerade kein Rechner eingeschaltet. Sie können selbst einen Rechner von außen 'aufwecken', also einschalten. Hierfür müssen Sie den folgenden Befehl abschicken, wenn Sie mit dem Sprungbrettrechner verbunden sind:

```
wakeAP rechnername
```

Die Rechnernamen sind nach dem Schema `<raumnummer>-<rechnernummer>` benannt. Einen Rechner im CIP-Pool A001 können Sie also z.B. mit folgendem Befehl 'wecken':

```
wakeAP a001-01
```

Wenn Sie einen Befehl zum 'Aufwecken' ausgeführt haben, warten Sie einen Moment bis der Rechner gestartet ist. Anschließend können Sie sich mittels SSH mit dem CIP-Pool-Rechner verbinden, in unserem Beispiel also mit:

```
ssh a001-01
```

Sobald Sie mit einem CIP-Pool Rechner verbunden sind, können Sie loslegen.

Zugriff beenden Ihre SSH-Session können Sie beenden, in dem Sie den Befehl

```
exit
```

eingeben. Wenn Sie sich mit einem CIP-Pool-Rechner verbunden haben, müssen Sie diesen Befehl zweimal ausführen, um zunächst die Verbindung zum CIP-Pool-Rechner und dann die Verbindung zum Sprungbrettrechner zu beenden.

[7] `https://www.rz.uni-wuerzburg.de/dienste/it-sicherheit/vpn/`

4 PuLP Basics

Um PuLP zu nutzen, müssen Sie zunächst die Bibliothek in Ihrem Python-Skript importieren. Dies können Sie mit folgender Zeile tun:

```
from pulp import *
```

Ein lineares Programm besteht aus drei Hauptkomponenten: Variablen, einer Zielfunktion (Objective) und Nebenbedingungen (Constraints).

Variablen Eine Variable x representiert eine unbekannte Größe, die wir bestimmen möchten. Diese Variable kann dabei reell, ganzzahlig oder binär sein. In PuLP können Sie eine Variable mit der Funktion `LpVariable` erstellen. Zum Beispiel können Sie eine nicht-negative reelle Variable x mit folgendem Code erstellen:

```
x = LpVariable("x", lowBound=0, cat=LpContinuous)
```

Eine ganzzahlige Variable y können Sie mit folgendem Code erstellen:

```
y = LpVariable("y", cat=LpInteger)
```

Falls die Variable y nur die Werte 0 und 1 annehmen soll, also binär sein soll, können Sie zusätzlich die Argumente `lowBound` und `upBound` verwenden, um die Variable auf den Bereich $[0, 1]$ (und aufgrund der Ganzzahligkeit somit auf die Werte 0 und 1) zu beschränken:

```
y = LpVariable("y", lowBound=0, upBound=1, cat=LpInteger)
```

Für binäre Variablen gibt es aber auch die Kategorie `LpBinary`, mit der die Variable automatisch auf die Werte 0 und 1 beschränkt wird. Somit könnte eine Binärvariable y auch mit folgendem Code erstellt werden:

```
y = LpVariable("y", cat=LpBinary)
```

Problem & Zielfunktion Jedes lineare Programm hat eine Zielfunktion, die entweder maximiert oder minimiert werden soll. In PuLP können Sie ein lineares Programm mit der Funktion `LpProblem` erstellen:

```
prob = LpProblem("BeispielLP", LpMinimize)
```

Um ein Maximierungsproblem zu erstellen, verwenden Sie stattdessen `LpMaximize`.

Die Zielfunktion können Sie mit dem += Operator hinzufügen. Im folgenden Beispiel fügen wir die Zielfunktion $3x + 2y$ hinzu:

```
prob += 3*x + 2*y
```

Constraints Nebenbedingungen können ebenfalls mit dem += Operator hinzugefügt werden. Zum Beispiel können Sie die Nebenbedingung $x + y \geq 10$ mit folgendem Code hinzufügen:

```
prob += x + y >= 10, "Beispiel Nebenbedingung"
```

Weitere Nebenbedingungen können auf die gleiche Weise hinzugefügt werden. Dabei müssen Sie den Nebenbedingungen nicht unbedingt einen Namen geben, sie könnten also eine Nebenbedingung auch einfach wie folgt hinzufügen:

```
prob += 2*x + y >= 15
```

Wenn Sie den Nebenbedingungen Namen geben, müssen Sie sicherstellen, dass diese Namen eindeutig sind, sonst wirft PuLP einen Fehler.

Lösen Nachdem Sie Ihr lineares Programm definiert haben, können Sie es mit der `solve` Methode lösen. Den Solver können Sie als Argument übergeben. Im Fall von Gurobi, sieht das wie folgt aus:

```
prob.solve(GUROBI())
```

Ergebnisse Nachdem das Problem gelöst wurde, können Sie den Status der Lösung und die Werte der Variablen ausgeben. Der Status gibt an, ob eine optimale Lösung gefunden wurde, das Problem unbeschränkt ist, oder ob es unlösbar ist. Den Status können Sie wie folgt ausgeben:

```
LpStatus[prob.status]
```

Die Werte der Variablen können mit der `varValue` Eigenschaft ausgegeben werden. Zum Beispiel können Sie den Wert der Variable x mit folgendem Code ausgeben:

```
x.varValue
```

Alternativ können Sie auch die Funktion `value` verwenden, um den Wert einer Variable auszugeben:

```
value(x)
```

Den Wert der Zielfunktion können Sie ebenfalls mit der Funktion `value` wie folgt ausgeben:

```
value(prob.objective)
```

4.1 Beispiel

Nun haben Sie alle notwendigen Grundlagen, um ein einfaches lineares Programm zu erstellen und zu lösen. Versuchen Sie, das folgende lineare Programm in PuLP zu formulieren und zu lösen:

$$\begin{aligned} \text{minimize} \quad & 3x + 2y \\ \text{subject to} \quad & x + y \geq 10 \\ & 2x + y \geq 15 \\ & x \geq 0 \\ & y \in \{0, 1\} \end{aligned}$$

Sie sollten eine optimale Lösung mit $x = 9$ und $y = 1$ erhalten, mit einem Zielfunktionswert von 29.

Ihr Python-Skript könnte dabei wie folgt aussehen:

```
from pulp import *

prob = LpProblem("BeispielLP", LpMaximize)

x = LpVariable("x", lowBound=0, cat=LpContinuous)
y = LpVariable("y", cat=LpBinary)

prob += 3*x + 2*y, "Beispiel Zielfunktion"

prob += x + y >= 10, "Nebenbedingung 1"
prob += 2*x + y >= 15, "Nebenbedingung 2"

prob.solve(GUROBI())

print(f"Status: {LpStatus[prob.status]}")
print(f"Zielfunktionswert: {value(prob.objective)}")
print(f"x = {x.varValue}")
print(f"y = {y.varValue}")
```

5 Fortgeschrittene Konzepte

Im Folgenden werden wir einige fortgeschrittene Konzepte von PuLP vorstellen, die dabei helfen komplexere lineare Programme zu formulieren. Dazu betrachten wir das folgende Problem: Stellen Sie sich vor, Sie möchten verreisen und müssen entscheiden,

welche Gegenstände Sie in Ihren Rucksack packen. Sie haben eine Liste von Gegenständen

```
items = ["Portmonee", "Taschentuch", "Handy", "Buch",
 ↪  "Schachbrett", "Zimmerpflanze", "Murmeln"]
```

die Sie mitnehmen könnten, und für jeden Gegenstand wissen Sie, wie viel er wiegt

```
gewicht = {
    "Portmonee": 200,
    "Taschentuch": 50,
    "Handy": 300,
    "Buch": 600,
    "Schachbrett": 800,
    "Zimmerpflanze": 1000,
    "Murmeln": 100
}
```

und welchen Nutzen er hat

```
nutzen = {
    "Portmonee": 20,
    "Taschentuch": 5,
    "Handy": 10,
    "Buch": 7,
    "Schachbrett": 10,
    "Zimmerpflanze": 0,
    "Murmeln": 1
}
```

Ihr Rucksack hat ein Gewichtslimit von 1200. Sie möchten nun entscheiden, welche Gegenstände Sie in Ihren Rucksack packen, um den Gesamtnutzen zu maximieren, ohne das Gewichtslimit zu überschreiten.

Bevor Sie weiterlesen, versuchen Sie dieses Problem zunächst auf Papier als (ganzzahliges) lineares Programm zu formulieren. Überlegen Sie, welche Variablen Sie benötigen, wie die Zielfunktion aussehen soll und welche Nebenbedingungen es gibt.

Indizierte Variablen Eine naheliegende Formulierung des Rucksackproblems erstellt für jeden Gegenstand eine binäre Variable, die angibt ob der Gegenstand in den Rucksack gepackt wird oder nicht. In PuLP kann gleich für eine Liste an Indizes (in diesem Fall die Gegenstände) mit der Funktion `LpVariable.dicts` jeweils eine Variable erstellt werden. In unserem Beispiel sieht dies wie folgt aus:

```
x = LpVariable.dicts("x", items, cat=LpBinary)
```

Auf die Variable für einen bestimmten Gegenstand, z. B. das Handy, kann dann mit `x["Handy"]` zugegriffen werden.

Summen In unserem Problem ist die Zielfunktion die Summe des Nutzens aller Gegenstände, die in den Rucksack gepackt werden. In PuLP kann eine Liste an Elementen mit der Funktion `lpSum` aufsummiert werden. In unserem Beispiel könnte die Zielfunktion wie folgt formuliert werden:

```
prob += lpSum([nutzen[i]*x[i] for i in items]),
 ↪  "Gesamtnutzen"
```

Die Nebenbedingung, dass das Gewicht der gepackten Gegenstände das Gewichtslimit nicht überschreiten darf, könnte wie folgt formuliert werden:

```
prob += lpSum([gewicht[i]*x[i] for i in items]) <= 1200,
 ↪  "Gewichtsbeschränkung"
```

Damit haben Sie nun eine vollständige Formulierung des Rucksackproblems in PuLP erstellt. Wenn Sie das Problem lösen, sollten Sie einen Nutzen von 42 erhalten, wobei die Gegenstände Portmonee, Taschentuch, Handy und Buch eingepackt werden.

Schleifen Wir können das Rucksackproblem auch noch ein wenig erweitern: Stellen Sie sich vor, dass Sie zusätzlich Gruppen von Gegenständen haben[8]

```
gruppen = [["Portmonee","Taschentuch"], ["Handy","Buch"],
 ↪  ["Schachbrett","Zimmerpflanze","Murmeln"]]
```

und dass Sie von jeder Gruppe mindestens einen Gegenstand mitnehmen wollen.

Überlegen Sie zunächst selbst (auf Papier), wie Sie diese zusätzliche Anforderung in Ihr lineares Programm integrieren können, bevor Sie weiterlesen.

Vermutlich werden Sie nicht umhinkommen, für jede Gruppe eine zusätzliche Nebenbedingung hinzuzufügen, die sicherstellt, dass von den Gegenständen in der Gruppe mindestens einer ausgewählt wird. Um für jede Gruppe eine solche Nebenbedingung hinzuzufügen, können Sie Schleifen verwenden. In unserem Beispiel könnte dies wie folgt aussehen:

```
for gruppe in gruppen:
    prob += lpSum([x[i] for i in gruppe]) >= 1
```

Wenn Sie nun das LP mit den zusätzlichen Nebenbedingungen lösen, sollten Sie einen vollständig gefüllten Rucksack mit einem Nutzen von 38 erhalten, wobei die Gegenstände Portmonee, Handy, Buch und Murmeln eingepackt werden.

[8] Die Gruppen in diesem Beispiel sind zugegebenermaßen etwas fragwürdig

Validierung des Codes Um zu überprüfen, dass Ihr Code das Problem korrekt formuliert bietet es sich an die (I)LP-Formulierung in eine LP-Datei zu exportieren. Dies können Sie mit der `writeLP` Methode tun, also in unserem Beispiel mit folgendem Code:

```
prob.writeLP("rucksack.lp")
```

Anhand der LP-Datei können Sie dann händisch überprüfen, ob die Formulierung korrekt ist oder sich beim Programmieren Bugs eingeschlichen haben.

In unserem Beispiel sollte die LP-Datei wie folgt aussehen:

```
\* Rucksackproblem *\
Maximize
Gesamtnutzen: 7 x_Buch + 10 x_Handy + x_Murmeln + 20
 ↪  x_Portmonee
 + 10 x_Schachbrett + 5 x_Taschentuch
Subject To
Gewichtsbeschränkung: 600 x_Buch + 300 x_Handy + 100 x_Murmeln
 + 200 x_Portmonee + 800 x_Schachbrett + 50 x_Taschentuch
 + 1000 x_Zimmerpflanze <= 1200
_C1: x_Portmonee + x_Taschentuch >= 1
_C2: x_Buch + x_Handy >= 1
_C3: x_Murmeln + x_Schachbrett + x_Zimmerpflanze >= 1
Binaries
x_Buch
x_Handy
x_Murmeln
x_Portmonee
x_Schachbrett
x_Taschentuch
x_Zimmerpflanze
End
```

Hier sehen Sie auch, dass PuLP für Nebenbedingungen automatisch Namen generiert, wenn diese nicht explizit angegeben werden.

Allgemeines Programm Durch die Verwendung von indizierten Variablen und Summen ist es dann möglich, ein allgemeines Programm zu schreiben, das für beliebige Gegenstände, Gewichte, Nutzen und Gewichtslimits funktioniert. Eine mögliche Implementierung finden Sie in unter folgendem Link: `https://wuecampus.uni-wuerzburg.de/moodle/mod/resource/view.php?id=3794881`