tightcenter

panikzettel.htwr-aachen.de

Operations Research 1 Panikzettel

Philipp Schröer, Younes Müller

Version 2 — 25.11.2020

Contents

1 Introduction

This Panikzettel is about the lecture Operations Research 1 by Prof. Lübbecke held in the winter semester 2019/20.

This Panikzettel is Open Source. We appreciate comments and suggestions at https://git.rwth-aachen.de/philipp.schroer/panikzettel.

In this Panikzettel, we introduce new notation for better readability: $\overline{n} := \{1, \dots, n\}$ for some n.

2 Mixed-Integer Linear Programs

A *mixed-integer linear program (MILP, MIP)* consists of an objective (here, a linear function to maximize), and linear inequalities over a bunch of variables.

Minimization and inequalities in other directions can be implemented by simple transformations (which we won't cover here).

Note that we use a slightly different notation in this Panikzettel than in the definition on the right. Here, variables along with a domain are listed first. Technically speaking, the domain is also a constraint!

Definition: Mixed-Integer Linear Program

A *mixed-integer linear program (MILP)* has the following form, where $n,\ q \in \mathbb{N}$:

$$\begin{aligned} \max \quad & c^T \\ \text{s.t.} \quad & Ax \leq b \\ & x \quad \in \mathbb{Z}_+^n \times \mathbb{Q}_+^q \end{aligned}$$

If $n = 0$, we have a *linear program* (LP), if $q = 0$, we have a *(pure) integer linear program* (ILP, IP).

Logical operators can be easily implemented in LPs. Let binary variable $x = 1$ iff statement X is true or $x = 0$ iff X is false.

Logical operators

- $X = Y \wedge Z$: $x \leq y$, $y \leq z$, $x \geq y + z - 1$
- $X = Y \vee Z$: $x \geq y$, $x \geq z$, $x \leq y + z$
- $X = \neg Y$: $x = 1 - y$

Consequences

- $X \implies Y$: $x \leq y$
- $\neg Y \implies \neg X$: $x \leq y$
- $X \iff Y$: $x = y$

3 Modeling with Integer Linear Programs

3.1 Minimum Cost (Bipartite) Matching Problem

This is also known as an *assignment problem*. The goal is to assign n workers to m machines with an assignment cost c_{ij}, minimizing the cost such that each machine is operated by a worker.

$$\text{Variables: } x_{ij} \in \{0, 1\} \quad \forall i \in \overline{n},\ j \in \overline{m} \qquad (x_{ij} = 1 \text{ iff worker } i \text{ operates } j)$$

$$\min \sum_{\substack{i \in \overline{n}, \\ j \in \overline{m}}} c_{ij} \cdot x_{ij}$$

$$\text{where } \sum_{j \in \overline{m}} x_{ij} \leq 1 \quad \forall i \in \overline{n} \qquad \text{(each worker to at most one machine)}$$

$$\sum_{i \in \overline{n}} x_{ij} = 1 \quad \forall j \in \overline{m} \qquad \text{(operate every machine)}$$

3.2 Transportation Problem

Let $G = (V, E)$ be a directed graph. Given n supplies a_i, and m demands b_j, and a transport cost c_{ij} for each pair $i \to j$, find a cost-minimal transport that fulfills demands with the given supplies.

$$\text{Variables: } x_{ij} \in \mathbb{N}_0 \quad \forall i \in \overline{n},\ j \in \overline{m} \qquad (\text{ship } x_{ij} \text{ units from } i \text{ to } j)$$

$$\min \sum_{(i,j) \in E} c_{ij} \cdot x_{ij}$$

$$\text{where } \sum_{(i,j) \in E} x_{ij} \geq b_j \quad \forall j \in \overline{m} \qquad \text{(fulfill demands)}$$

$$\sum_{(i,j) \in E} x_{ij} \leq a_i \quad \forall i \in \overline{n} \qquad \text{(supply limits)}$$

3.3 Minimum Cost Network Flows

We want to find a minimum cost flow in a directed graph $G = (V, A)$ with a cost function $c : E \to \mathbb{N}_0$, arc capacities $u : A \to \mathbb{N}_0$ and demands $b : V \to \mathbb{Z}$.

Outgoing edges from i are denoted $\delta^+(i)$, and incoming edges are denoted $\delta_-(i)$.

Definition: Flow

A *flow* is a mapping $f : A \to \mathbb{N}_0$ with

- *Capacity satisfaction:* For all $a \in A$: $f(a) \leq u(a)$,
- *Flow conservation:* For all $v \in V$: $\sum_{a \in \delta^+(v)} f(a) - \sum_{a \in \delta^-(v)} f(a) = b(v)$.

$$
\begin{aligned}
\text{Variables: } & x_{ij} \in \mathbb{N} && \forall (i,j) \in A && \text{(flow of } x_{ij} \text{ from } i \text{ to } j\text{)} \\
\min \quad & \sum_{(i,j)\in A} c_{ij} \cdot x_{ij} \\
\text{where} \quad & x_{ij} \leq u_{ij} && \forall (i,j) \in A && \text{(capacities)} \\
& \sum_{(i,j)\in\delta^+(i)} x_{ij} - \sum_{(j,i)\in\delta^-(i)} x_{ji} = b_i && \forall\, i \in V && \text{(flow conservation)}
\end{aligned}
$$

3.4 Knapsack Problem

In the *0-1 knapsack problem*, we want to to find a profit-maximizing selection of n items of size a_i with profits p_i to pack in a knapsack of capacity b.

$$
\begin{aligned}
\text{Variables: } & x_i \in \{\, 0,\ 1 \,\} && \forall\, i \in \overline{n} && (x_i = 1 \text{ iff } i \text{ is in the knapsack}) \\
\max \quad & \sum_{i\in\overline{n}} p_i \cdot x_i \\
\text{where} \quad & \sum_{i\in\overline{n}} a_i \cdot x_i \leq b && && \text{(capacity)}
\end{aligned}
$$

3.5 Bin Packing Problem

In the (one-dimensional) *bin packing problem*, we want to pack n items of sizes a_i into bins of capacity b, minimising the number of bins. There is an upper bound of m bins.

$$
\begin{aligned}
\text{Variables: } & x_{ij} \in \{\, 0,1 \,\} && \forall\, i \in \overline{n},\ j \in \overline{m} && (x_{ij} = 1 \text{ iff item } i \text{ is in bin } j) \\
& y_j \in \{\, 0,1 \,\} && \forall\, j \in \overline{m} && (y_j = 1 \text{ iff bin j is used}) \\
\min \quad & \sum_{j\in\overline{m}} y_j \\
\text{where} \quad & \sum_{j\in\overline{m}} x_{ij} = 1 && \forall\, i \in \overline{n} && \text{(every item } i \text{ must be packed)} \\
& \sum_{i\in\overline{n}} a_i \cdot x_{ij} \leq b \cdot y_j && \forall\, j \in \overline{m} && \text{(capacity for each bin)}
\end{aligned}
$$

Pattern-based Alternative Model

Use one variable for each feasible *pattern*: One pattern $x \in \{\, 0,1 \,\}^n$ for every combination of items that can be packed into a single bin:

$$
Q = \left\{ x \in \{\, 0,1 \,\}^n \;\middle|\; \sum_{i=1}^{n} a_i \cdot x_i \leq b \right\}.
$$

We minimize the number of used bins by minimising the number of patterns λ_q. To ensure that each item is packed exactly once, we define the set of patterns that contain item i: $Q_i = \{ q \in Q \mid q_i = 1 \}$ for all $i \in \overline{n}$.

$$\begin{aligned}
&\text{Variables: } \lambda_q \in \{ 0, 1 \} \quad \forall q \in Q \quad (\lambda_q = 1 \text{ iff a bin is filled according to pattern } q) \\
&\min \quad \sum_{q \in Q} \lambda_q \\
&\text{where} \quad \sum_{q \in Q_i} \lambda_q = 1 \quad \forall i \in \overline{n} \qquad \text{(each item is packed exactly once)}
\end{aligned}$$

3.6 Location Problems

In the basic *facility location problem*, we have m potential facilities with opening costs f_i, and connection costs c_{ij}. We want to open facilities such that each client is served by exactly one opened facility, minimizing the total cost.

$$\begin{aligned}
&\text{Variables: } x_{ij} \in \{ 0,\ 1 \} && \forall i \in \overline{m}, \forall j \in \overline{n} && (x_{ij} = 1 \text{ iff client } j \text{ is served by facility } i) \\
&\qquad\quad y_i \in \{ 0,\ 1 \} && \forall i \in \overline{m} && (y_i = 1 \text{ iff facility } i \text{ is opened}) \\
&\min \quad \sum_{i \in \overline{m}} f_i \cdot y_i + \sum_{\substack{i \in \overline{m}, \\ j \in \overline{n}}} c_{ij} \cdot x_{ij} \\
&\text{where} \quad \sum_{i \in \overline{m}} x_{ij} = 1 && \forall j \in \overline{n} && \text{(exactly one serving facility per client)} \\
&\qquad\quad x_{ij} \leq y_i && \forall i \in \overline{m}, \forall j \in \overline{n} && \text{(a facility can serve only when opened)}
\end{aligned}$$

There are various variants of the basic problem:

- *demands and capacities*: demands q_j and capacities C_i: $\sum_{j \in \overline{n}} q_j x_{ij} \leq C_i \forall i \in \overline{m}$
- *p-median problem*: open exactly p facilities: $\sum_{i \in \overline{m}} y_i = p$.
- *p-center problem*: minimize largest distances to open facilities: $\min \max_{i,j}\ c_{ij} \cdot x_{ij}$.

3.7 Lot Sizing (sketch)

We decide how much of a product we produce at a time step and how much we store while minimizing the storing cost and setup cost for the machine.
There are T periods, with a demand of b_t in each. Storing the product costs l per unit per time and

setting up the machine to produce goods costs r in a timestep.

$$
\begin{array}{rlll}
\text{Variables:} & x_t \geq 0 & \forall\, t \in \overline{T} & \text{(amount to produce in t)} \\
& y_t \geq 0 & \forall\, t \in \overline{T} & \text{(amount to store in t)} \\
& z_t \in \{\, 0,\, 1 \,\} & \forall t \in \overline{T} & \text{(set up in t?)} \\
\\
\min & \displaystyle\sum_{t \in \overline{T}} l \cdot y_t + \sum_{t \in \overline{T}} r \cdot z_t & & \\
\text{where} & y_{t-1} + x_t - y_t = b_t & \forall\, t \in \overline{T} & \text{(balance material)} \\
& x_t \leq z_t \cdot M & \forall\, t \in \overline{T} & \text{(set up only if producing)}
\end{array}
$$

3.8 Scheduling

In the *parallel machine scheduling* setting, we want to assign n jobs which require times p_j to m identical machines, minimizing the *makespan* (longest completion time).

$$
\begin{array}{rlll}
\text{Variables:} & x_{jk} \in \{\, 0,\, 1 \,\} & \forall\, j \in \overline{n},\, k \in \overline{m} & (x_{jk} = 1 \text{ iff job } j \text{ is assigned to machine } k) \\
& C_{\max} \geq 0 & & \text{(makespan)} \\
\\
\min & C_{\max} & & \\
\text{where} & \displaystyle\sum_{k \in \overline{m}} x_{jk} = 1 & \forall\, j \in \overline{n} & \text{(assign all jobs)} \\
& \displaystyle\sum_{j \in \overline{n}} p_j \cdot x_{jk} \leq C_{\max} & \forall\, k \in \overline{m} & (\text{last completion time defines } C_{\max})
\end{array}
$$

On a **single machine**, we can also solve *scheduling with precedence constraints*. Given n jobs, and a partial order $i \prec j$ on the jobs that defines that i must start before j, we want to minimize the makespan again. Define $E = \{\, \{\, i, j \,\} \mid i \neq j,\ i \not\prec j,\ j \not\prec i \,\}$ and choose M to be "large".

$$
\begin{array}{rlll}
\text{Variables:} & x_{ij} \in \{\, 0,\, 1 \,\} & \forall\, \{\, i, j \,\} \in E & (x_{ij} = 1 \text{ iff job } i \text{ runs before } j) \\
& t_j \geq 0 & \forall\, j \in \overline{n} & \text{(start times)} \\
& C_{\max} \geq 0 & & \text{(makespan)} \\
\\
\min & C_{\max} & & \\
\text{where} & t_i + p_i \leq t_j & \forall\, i \prec j & \text{(start after previous has completed)} \\
& t_i + p_i \leq t_j + (1 - x_{ij}) \cdot M & \forall\, \{\, i, j \,\} \in E & (1) \\
& t_j + p_j \leq t_i + x_{ij} \cdot M & \forall\, \{\, i, j \,\} \in E & (2) \\
& t_j + p_j \leq C_{\max} & \forall\, j \in \overline{n} &
\end{array}
$$

Equations (1) and (2) express that $x_{ij} = 1$ iff i precedes j. Formally, we have $x_{ij} = 1 \Rightarrow t_i + p_i \leq t_j$ and $x_{ij} = 0 \Rightarrow t_j + p_j \leq t_i$. These two expressions are linearised using the "big M" constant.

Extensions:

- *Release dates* r_j can be added by constraints $t_j \geq r_j \quad \forall\, j \in \overline{n}$.
- *Deadlines* d_j can be enforced by $t_j + p_j \leq d_j \quad \forall\, j \in \overline{n}$.

The "big M" can be avoided by discretising the time horizon, creating lots of new variables. So let the time horizon $t = 1, \ldots, T$.

$$
\begin{array}{rll}
\text{Variables:} & x_{jt} \in \{0,1\} & \forall\, j \in \overline{n},\ t \in \overline{T} \qquad (x_{jt} = 1 \text{ iff job } j \text{ starts at time } t) \\
& C_{\max} & \text{(makespan)} \\
\min & C_{\max} & \\
\text{where} & \displaystyle\sum_{t=1}^{T-p_j+1} x_{jt} = 1 & \forall\, j \in \overline{n} \qquad \text{(each job must start)} \\
& \displaystyle\sum_{t=1}^{T-p_i+1} (t+p_i) \cdot x_{it} \le \sum_{t=1}^{T-p_j+1} t_j \cdot x_{jt} & \forall\, i,\ j \in \overline{n},\ i \prec j \qquad \text{(precedence)} \\
& \displaystyle 1 - x_{jt} \ge \sum_{\tau=t}^{t+p_j-1} x_{i\tau} & \forall\ \begin{array}{l} i \in \overline{n} \setminus \{j\},\ j \in \overline{n} \\ t \in \overline{T - p_j + 1} \end{array} \quad \left(\begin{array}{c} \text{job } j \text{ starts at } t \Rightarrow \\ \text{no other } i \text{ can start before } t + p_j \end{array} \right) \\
& \displaystyle\sum_{t=1}^{T-p_j+1} (t+p_j) \cdot x_{jt} \le C_{\max} & \forall\, j \in \overline{n} \qquad \text{(completion time)}
\end{array}
$$

In the above, the expression $\sum_{t=1}^{T-p_i+1} t \cdot x_{it}$ equals the starting time of job i. The expression $\sum_{t=1}^{T-p_i+1} (t+p_i) \cdot x_{it}$ equals the end time of job i.

3.9 Minimum Spanning Tree

A *minimum spanning tree* of a graph $G = (V, E)$ with edge weights $c_e \ge 0$ is a tree in G that connects all nodes.

$$
\begin{array}{rll}
\text{Variables:} & x_e \in \{0,1\} & \forall\, e \in E \qquad (x_e = 1 \text{ iff edge } e \text{ is in the MST}) \\
\min & \displaystyle\sum_{e \in E} c_e \cdot x_e & \\
\text{where} & \displaystyle\sum_{e \in E} x_e = |V| - 1 & \text{(spanning tree)} \\
& \displaystyle\sum_{e \in \delta(X)} x_e \ge 1 & \forall\, \emptyset \subsetneq X \subsetneq V \qquad \text{(connectivity)}
\end{array}
$$

3.10 Traveling Salesperson Problem

In the *traveling salesperson problem (TSP)*, we want to find a minimum cost tour in a graph $G = (V, E)$ with edge costs c_{ij}. A *tour* is a cycle that visits each node exactly once.

$$
\begin{aligned}
\text{Variables: } & x_e \in \{0, 1\} && \forall e \in E && (x_{ij} = 1 \text{ iff } i \to j \text{ is in the tour}) \\
\min \quad & \sum_{e \in E} c_e \cdot x_e \\
\text{where} \quad & \sum_{e \in \delta(i)} x_e = 2 && \forall i \in V && \text{(connectivity)} \\
& \sum_{e \in \delta(S)} x_e \geq 2 && \forall S \subsetneq V,\ |S| \geq 3 && \text{(subtour elimination constraints)}
\end{aligned}
$$

Alternative Subtour Elimination Constraint

The following formulation has only linearly many constraints. We formulate the TSP for a directed graph $G = (V, A)$.

$$
\begin{aligned}
\text{Variables: } & x_e \in \{0, 1\} && \forall e \in A && (x_{ij} = 1 \text{ iff } i \to j \text{ is in the tour}) \\
& u_i \geq 0 && \forall i \in V && \text{(index of i in tour)} \\
\min \quad & \sum_{e \in A} c_e \cdot x_e \\
\text{where} \quad & \sum_{e \in \delta^-(i)} x_e = 1 && \forall e \in A && \text{(in-degree constraint)} \\
& \sum_{e \in \delta^+(i)} x_e = 1 && \forall e \in A && \text{(out-degree constraint)} \\
& u_i + 1 \leq u_j + (1 - x_e) \cdot M && \forall i \in V && \text{(subtour elimination)}
\end{aligned}
$$

The subtour elimination constraint uses the big-M method. It can be adapted to feature time windows.

3.11 Vehicle Routing Problem

In the *vehicle routing problem (VRP)*, we want to serve n customers with K identical vehicles on a directed graph $G = (V, E)$. There is a driving cost for each edge of c_{ij}. We want to find a tour for each vehicle starting and ending at a depot d, so that all customers are served (by one vehicle) at

minimum cost. We assume that not all vehicles have to be used.

$$\begin{aligned}
\text{Variables: } & x_{ij}^k \in \{0,1\} && \forall k \in \overline{K},\ (i,j) \in E && (x_{ij}^k = 1 \text{ iff vehicle } k \text{ goes from } i \text{ to } j)\\
& z_i^k \in \{0,1\} && \forall k \in \overline{K},\ i \in V && (z_i^k = 1 \text{ iff vehicle } k \text{ visits } i)\\
\min & \sum_{(i,j)\in E} c_{ij}^k \cdot x_{ij}^k\\
\text{where } & \sum_{(i,j)\in\delta^+(i)} x_{ij}^k = \sum_{(j,i)\in\delta^-(i)} x_{ji}^k \geq z_i^k && \forall k \in \overline{K},\ i \in V && \text{(flow conservation)}\\
& \sum_{e\in\delta^-(d)} x_e^k = 1 && && \text{(every vehicle visits depot)}\\
& \sum_{k\in\overline{K}} z_i^k = 1 && \forall i \in V && \text{(every customer must be visited)}\\
& \sum_{(i,j)\in\delta^+(S)} x_{ij}^k = \sum_{(j,i)\in\delta^-(S)} x_{ji}^k \geq z_i^k && \forall k \in \overline{K},\ S \subsetneq V \setminus d,\ i \in S && \text{(SEC)}
\end{aligned}$$

3.11.1 Capacity - CVRP

In the *capacitated vehicle routing problem (CVRP)*, we also have demands $q_i \geq 0$ and a vehicle capacity Q. We just need one additional constraint:

$$\text{where} \quad \sum_{i\in V} z_i^k \cdot q_i \leq Q \quad \forall k \in \overline{K} \quad \text{(vehicle capacities)}$$

3.12 Pickup and Delivery

With *CVRP with pickups and deliveries*, we extend the problem domain to $q_i \in \mathbb{R}$. Instead of the simple constraint above, we need a "big M" constraint and new variables y_i^k.

$$\begin{aligned}
\text{Variables: } & 0 \leq y_i^k \leq Q && \forall i \in V,\ k \in \overline{K} && \text{(vehicle load after leaving } i\text{)}\\
\text{where } & y_i^k + q_i \leq y_j^k + (1 - x_{ij}^k) \cdot M && \forall (i,j) \in E,\ k \in \overline{K} && \text{(vehicle load)}
\end{aligned}$$

(The "big M" constraint is equivalent to $x_{ij}^k = 1 \implies y_i^k + q_j \leq y_j^k$.)

3.12.1 Time Windows - VRPTW

The last variant we cover is the *vehicle routing problem with time windows (VRPTW)*. It is like VRP, but now we require delivery in a time window $[a_i, b_i]$ at customer i. The vehicles need t_{ij} to traverse $i \Rightarrow j$. Add variables $w_i^k \geq 0 \quad \forall i \in V,\ k \in \overline{K}$ for the time vehicle k starts at customer i. We want to express

$$x_{ij}^k = 1 \implies w_i^k + t_{ij} \leq w_j^k \qquad \forall (i,j) \in E,\ k \in \overline{K}.$$

This can be realized using a "big M" constraint. It also acts as SEC, so the original SEC can be removed. We add the following constraints:

$$\text{Variables: } w_i^k \geq 0 \qquad \forall\, i \in E, k \in \overline{K} \qquad \text{(vehicle k start time at customer i)}$$

$$\text{where } w_i^k + t_{ij} \leq w_j^k + (1 - x_{ij}^k) \cdot M \quad \forall (i,j) \in E,\ k \in \overline{K} \quad \left(\begin{array}{l}\text{link } w_i^k \text{ and } x_{ij}^k, \\ \text{vehicle k needs } t_{ij} \text{ to go from i to j}\end{array}\right)$$

$$a_i \leq w_i^k \leq b_i \qquad \forall\, i \in V \qquad \text{(time window is met)}$$

Sidenote: The constraint is similar to the alternative SEC in TSP (**??**).

3.13 Set Covering, Set Partitioning, Set Packing

The class of set problems includes *set covering*, *set partitioning*, and *set packing*. In every case, we have a "universe" $\mathcal{U} = \{ e_1, \ldots, e_n \}$, a set of subsets $\mathcal{S} \subseteq \mathcal{P}(\mathcal{U})$, and a cost $c(S)$ for each subset $S \in \mathcal{S}$. All three LPs are similar, and differ only in the objective and inequation operator.

Set Cover	min	$\geq$
Set Partition	min	$=$
Set Packing	max	$\leq$

Set Cover We want to cover all elements with a subset of $\mathcal{S}$.

$$\text{Variables: } x_S \in \{ 0,1 \} \quad \forall\, S \in \mathcal{S} \qquad (x_S = 1 \text{ iff } S \text{ is selected})$$

$$\min \quad \sum_{S \in \mathcal{S}} c(S) \cdot x_s$$

$$\text{where} \quad \sum_{\substack{S \in \mathcal{S}, \\ e_i \in S}} x_S \geq 1 \quad \forall\, i \in \overline{n} \qquad \text{(must cover all elements)}$$

Set Partition Choose a partition $\subseteq \mathcal{S}$, i.e. each element is in exactly one set of the partition.

$$\text{Variables: } x_S \in \{ 0,1 \} \quad \forall\, S \in \mathcal{S} \qquad (x_S = 1 \text{ iff } S \text{ is in the partition})$$

$$\min \quad \sum_{S \in \mathcal{S}} c(S) \cdot x_s$$

$$\text{where} \quad \sum_{\substack{S \in \mathcal{S}, \\ e_i \in S}} x_S = 1 \quad \forall\, i \in \overline{n} \qquad \text{(each element is in exactly one selected set)}$$

Set Packing Select cost-maximal sets, but each element can only be selected at most once.

$$\text{Variables: } x_S \in \{ 0,1 \} \quad \forall\, S \in \mathcal{S} \qquad (x_S = 1 \text{ iff } S \text{ is selected})$$

$$\max \quad \sum_{S \in \mathcal{S}} c(S) \cdot x_s$$

$$\text{where} \quad \sum_{\substack{S \in \mathcal{S}, \\ e_i \in S}} x_S \leq 1 \quad \forall\, i \in \overline{n} \qquad \text{(each element is selected at most once)}$$

The three set problems are problems that reoccur frequently. The alternative bin-packing model (**??**) for example is a packing problem, where the the universe U consists of the items and the set of subsets $S \subseteq P(U)$ is the set of possible one-bin-packings.

4 Modeling with Non-Linear Integer Programs

Previously, we only presented linear integer programs (although often with exponential size). Now we show some techniques to deal with nonlinearity.

Product of two binary variables A product of two binary variables $z = x \cdot y$ where $x, y, z \in \{0, 1\}$ can be linearised as follows:

$$
\begin{aligned}
z &\leq x && x = 0 \Rightarrow z = 0 \\
z &\leq y && y = 0 \Rightarrow z = 0 \\
y &\geq x + y - 1 && x = 1 \wedge y = 1 \Rightarrow z = 1
\end{aligned}
$$

Product of binary and bounded variable Consider $z = x \cdot y$ where $x \in \{0, 1\}$ and $l \leq y \leq u$.

$$
\begin{aligned}
z &\leq u \cdot x \\
z &\geq l \cdot x \\
z &\leq y - l \cdot (1 - x) \\
z &\geq y - u \cdot (1 - x)
\end{aligned}
$$

5 Relaxations of Strength and Models

Let's start with general problem $X \subseteq \mathbb{Z}^n_+$, i.e. the task is to find minimal/maximal points in X. A *formulation* for X is a polyhedron that includes all of X (but possibly more points). Because the polyhedra can be arbitrarily precise, there are infinitely many formulations for a problem.

Definition: Formulation

A polyhedron $P \subseteq \mathbb{Q}^n_+$ is a *formulation* for a problem $X \subseteq \mathbb{Z}^n_+$ if $X = P \cap \mathbb{Z}^n_+$.

Analogously for mixed integer sets $X \subseteq \mathbb{Z}^n_+ \times \mathbb{Q}^q_+$.

However, we want the polyhedron to be as precise as possible. We call a formulation *stronger* than another formulation if it is strictly smaller.

The strongest possible formulation is called *ideal* if it is exactly the convex hull of the problem.

Definition: Strength of Formulations

Let P_1, P_2 be formulations for X.
P_1 is *stronger* than P_2 when $P_1 \subsetneq P_2$.

A formulation P is *ideal* for X when $P = \text{conv}(X)$.

By removing integrality constraints on an integer linear program, we can find lower bounds for optimal solutions (for min problems). We call this *relaxation*, because the problem without integrality constraints is less strong than the integer problem.

Theorem: Relaxation

Let $z^* = \{ c^t x \mid Ax \geq b,\ x \in \mathbb{Z}^n_+ \}$.

The *relaxation* $\underline{z} = \{ c^t x \mid Ax \geq b,\ x \geq 0 \}$ is a lower bound on the optimum: $\underline{z} \leq z^*$.

Given an IP/MIP, any integer feasible solution gives an upper bound $\overline{z}$ for the optimal solution z^*. We call $\overline{z}$ *primal bound*.

Together with the result from optimizing over any relaxation of an IP, we get lower and upper bounds for the IP solution.

The gap γ gives us a worst-case quality guarantee in percent on the primal bound $\overline{z}$. $\overline{z}$ is no further away from z^* than γ% (for min problems).

Definition: Relaxation Quality (gap)

Let z^* be the optimal solution for an IP/MIP,

- $\overline{z} \geq z^*$ be the primal bound,
- $\underline{z} \leq z^*$ the dual bound.

We define gap $\gamma := \begin{cases} \frac{\overline{z}-\underline{z}}{|\underline{z}|} & \text{if } \underline{z} \cdot \overline{z} > 0 \\ \infty & \text{otherwise} \end{cases}$

5.1 Cutting Planes

Cutting planes are used to strengthen the LP that results from relaxation of an IP again, so that the lower bound (again, we assume min problems) becomes more precise (that is, higher).

A *valid* inequality for an LP relaxation preserves all integer solutions. And a *cutting plane* is a valid inequality that cuts away an optimal solution x^* in the LP relaxation (that is not a solution for the original IP).

Definition: Cutting plane

Given

- an IP $\min \{ c^t x \mid \overbrace{Ax \geq b,\ x \in \mathbb{Z}^n_+}^{:=X} \}$,
- its LP relaxation $\min \{ c^t x \mid \underbrace{Ax \geq b,\ x \geq 0}_{:=P} \}$.

An inequality $a^t x \geq a_0$ is *valid* for X iff $a^t \overline{x} \geq a_0 \qquad \forall \overline{x} \in X$

A valid $a^t x \geq a_0$ for X is a *cutting plane* iff $a^t x^* < a_0 \qquad \exists x^* \in P$

6 Exact Algorithms

6.1 Branch-and-Bound

Because of math, solving mixed integer programs (MIPs) is NP-hard. We use the *branch-and-bound algorithm* to recursively split problems into sub-problems and then try to find a solution for the MIP using LPs.

We keep a list of all current sub-problems in a list ("open") and select a sub-problem S. We consider the LP relaxation of S. If S is infeasible or worse than the current best solution, we discard and close S. If the solution is integral, we save the solution as our current best solution ("*incumbent*"). Otherwise we split S into sub-problems and put those in the list. When all those subproblems are closed, close S. We do this until the list of subproblems is empty.

In this way, we explore the tree of subproblem solutions step by step. But using the upper bound $\overline{z}$, we can *prune* nodes: If a subproblem has a fractional value that is not better than $\overline{z}_{IP}$, we can ignore it and its children (their values are at least as high). Finally, the LP relaxation values of **open** subproblems with the current $\overline{z}_{IP}$ gives us the gap during the execution.

Theorem: Branch-and-Bound Gap

Let $\overline{z}_{IP}$ be the upper bound of closed node's values, and $\underline{z}_{LP}$ the minimum LP relaxation value of all open subproblems.

$$\text{gap} = \frac{|\overline{z}_{IP} - \underline{z}_{LP}|}{|\underline{z}_{LP}|}.$$

Algorithm: Branch-and-Bound

Input: IP $\min c^T x,\ x \in X$ (bounded and feasible).

Output: Optimal integer solution x^*.

1. Set $\overline{z}_{IP} \leftarrow \infty$, let $x^* \leftarrow$ undefined.
   At any point, $\underline{z}_{LP}$ is the minimal LP relaxation value of all open subproblems.
2. For next new subproblem S: Solve its relaxation.

If infeasible:	**Close** S.	
If $\hat{x}$ is integral and $c^T\hat{x} < \overline{z}_{IP}$:	Set $\overline{z}_{IP} \leftarrow c^T\hat{x}$, $x^* \leftarrow \hat{x}$. **Close** S.	*(better)*
If $\hat{x}$ is fractional and $c^T\hat{x} \geq \overline{z}_{IP}$:	**Close** S.	*(pruning)*
If $\hat{x}$ is fractional and $c^T\hat{x} < \overline{z}_{IP}$:	**Branch** (add new open subproblems). After all child LPs are solved[a], **close** S.	

3. Return x^*.

[a] A parent node is closed if all direct children's LPs are solved (not necessarily closed).

Search Strategy The lecture discusses two strategies to find the next subproblem among the list of open subproblems: *Depth-first search (DFS)*, and *best-first search (BestFS)*. DFS selects the deepest sub-problem in the search tree first. This strategy is fast in finding feasible solutions, that however may be of poor quality. BestFS selects the sub-problem with the smallest local dual bound first. This gives a small search tree, but requires a lot of memory.

Branching Rule We can use several different *branching rules* to generate the subproblems.

- **Selecting fractional variables**: For $\overline{x}_j \notin \mathbb{Z}_+$, we add either $x_j \leq \lfloor \overline{x}_j \rfloor$ or $x_j \geq \lceil \overline{x}_j \rceil$. If there are several fractional variables, we compute a *score* of some kind and select the best.
- With **most infeasible branching**, we select the variable with the fractional part closest to 0.5. Formally, we select $\operatorname{argmin}_j \min \{ \overline{x}_j - \lfloor \overline{x}_j \rfloor,\ \lceil \overline{x}_j \rceil - \overline{x}_j \}$. Although popular, it's poorly performing.
- Another poorly performing alternative that is not so popular is **least infeasible branching** where the the least fractional variable is selected.
- Then we have the method of **pseudo costs** where a success history of variables is tracked and the success history is multiplied with the "fractionality", i.e. the closeness of the fractional part to 0.5.

6.2 Solving LPs With Exponentially Many Constraints

TSP example: Even the LP relaxation has too many constraints to be solved fast. To speed up the

solving of the LP relaxation, first relax (ignore) all subtour elimination constraints (SEC). Compute the solution of the resulting LP and search the solution for subtours, which violate SECs. This is done by finding minimum cuts in the support graph.

The *support graph* is the graph which has an arc for every decision variable with its value as weight. Finding a *minimum cut* with value v in the support graph means finding a set of vertices which are the most disconnected from the rest of the graph. If $v < 2$, a SEC is violated; thus add the violated SEC to the LP and repeat. If $v \geq 2$, no SEC is violated, thus the solution is optimal.

$$\sum_{e \in \delta(S)} x_e \geq 2 \qquad \forall\, S \subsetneq V, |S| \geq 3 \qquad \text{(subtour elimination constraints of TSP)}$$

We can also use valid inequalities (**??**) to strengthen LP relaxations. This is called *cutting plane method*. Within branch-and-bound, this is called *branch-and-cut*.

6.3 Column Generation

Column generation is a method to address models where there are many more variables than constraints. This occurs in e.g. our bin packing encoding (**??**). In practice most of the variables, that are defined in an integer program are zero in the end. So the ide of column generation is to solve the Lp with a small subset of variables, while assuming the others to be zero. Then the set of variables, that the problem is solved for, is enlarged iteratively. More on this in OR3. When the LP relaxation of each node branch-and-bound is solved by column generation, we call this *branch-and-price*.

6.4 0-1 Knapsack Problem with Bellman's Equation

We can also solve optimisation problems *without linear programming*. A completely misnamed technique called *dynamic programming* uses tables to solve a problem by memoisation. The *Bellman-Ford algorithm* essentially uses dynamic programming to solve the shortest-path problem.

Recall the 0-1 Knapsack problem (**??**): We have n items of sizes a_i, with profits p_i and a capacity of the knapsack of b. We want to find a profit maximizing subset of the items that respects the capacity.

Bellman's equation calculates the maximum profit using only the first j items and a capacity c:

$$j = 1: \qquad f(1,c) = \begin{cases} 0 & \text{for } c = 0, \ldots, a_1 - 1, \\ p_1 & \text{for } c = a_1, \ldots, b. \end{cases}$$

$$j \geq 2: \qquad f(j,c) = \begin{cases} \overbrace{f(j-1,c)}^{j \text{ does not fit}} & \text{for } c = 0, \ldots, a_j - 1, \\ \max(\underbrace{f(j-1,c)}_{\text{do not select } j}, \underbrace{f(j-1,c-a_j)+p_j}_{\text{select } j}) & \text{for } c = a_j, \ldots, b. \end{cases}$$

Thus the optimal profit value is given by $f(n, b)$. Consider the table on the right: We want to compute the field on the bottom right ($f(n, b)$) that contains the final profit value.

	$c = 0$	$c = 1$	...
$j = 1$	0	0	p_1
$j = 2$	0	0	
...	0		

1. Fill in the first line: Zeros, from $c = a_1$ enter p_1.
2. For each next line j, calculate for all c the *states* (cells):
   - While $c < a_j$, copy from $\uparrow$ (j does not fit).
   - Then, calculate $\max(\uparrow, \uparrow\!\!\leftarrow + p_j)$.

The chosen items must be computed by tracing back the computation from the final state:

$$\text{if } f(j,c) = \overbrace{f(j-1,c)}^{j \text{ does not fit}} \implies x_j = 0$$
$$\text{if } f(j,c) = \underbrace{f(j-1,c-a_j)+p_j}_{\text{select } j} \implies x_j = 1$$

7 Heuristics

Heuristics are algorithms that try to solve problems without some guarantees: Often, there is no optimality guarantee for computed results, nor a guarantee on the runtime. And then the heuristic may not even find a feasible solution. In practice, many heuristics are quite useful.

Heuristics can be classified into *construction heuristics* which build a solution from scratch and *improvement heuristics* which improve an existing solution. Some heuristics are *problem-specific*, and some are *general heuristics* for many kinds of problems.

Problem-specific heuristics One simple heuristic is the *nearest neighbor heuristic* for TSP where a TSP tour is constructed by always choosing the nearest univisited vertex. More on TSP heuristics can be found in our Logistics Systems Planning 1 Panikzettel.

The nearest neighbor heuristic is an example of a *greedy* heuristic: It always chooses the next best solution, but that means it only ever reaches local optima, and not necessarily a global optimum.

It is a construction heuristic because it builds up a tour from scratch.

General Improvement Heuristics We have a bunch of general heuristics that (try to) improve existing solutions.

Local search looks in the space around one existing feasible solution by trying some small changes to the solution. This is repeated until no better solution can be found: we have reached a local optimum. It's also possible to choose worse solutions sometimes to escape a local optimum and try to find a better one.

Tabu search is another general method.

Simulated annealing is a local search that accepts worsening solutions with a certain probability that is initially very large and decreases with time. The method is named after the physical process called "annealing" where metal is cooled slowly to strengthen it.

Genetic algorithms mimic natural selection. A set of solutions called *population* is generated. Solutions are combined in some way and then only the best are kept. This is repeated for some time to generate good solutions.

Primal Heuristics in Branch-and-Bound Heuristics can also be applied to find integer solutions for integer programs, supplementing branching. For example, rounding fractional values is very cheap, but does not always work. Doing this repeatedly until a solution or infeasibility is found is called *diving*.

Approximation Algorithms An *α-approximation algorithm* runs in polynomial time and always computes solutions that are only α times worse than the optimal solution. One example is the amazing *Christofide's approximation algorithm*. We discuss it and many other approximation algorithms in our Effiziente Algorithmen Panikzettel.