Department of Mathematics and Computer Science
University of Southern Denmark, Odense

April 29, 2025
Marco Chiarandini

AI505 – Optimization

Sheet 09, Spring 2025

Exercises with the symbol $^+$ are to be done at home before the class. Exercises with the symbol * will be tackled in class. The remaining exercises are left for self training after the exercise class. Some exercises are from the text book and the number is reported. They have the solution at the end of the book.

Exercise 1$^+$ (19.2)

A Boolean satisfiability problem, often abbreviated SAT, requires de- termining whether a Boolean design exists that causes a Boolean-valued objective function to output true. SAT problems were the first to be proven to belong to the difficult class of NP-complete problems.21 This means that SAT is at least as difficult as all other problems whose solutions can be verified in polynomial time. Consider the Boolean objective function:

$$f(\boldsymbol{x}) = x_1 \wedge (x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2)$$

Formulate the problem as an integer linear program. Can any Boolean satisfiability problem be formulated as an integer linear program? Solve the problem with scipy.

Exercise 2*

Consider $\max\{\boldsymbol{c}^T\boldsymbol{x} \mid A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \in \mathbb{Z}^n\}$. Sometimes the solution of the linear relaxation is already integral. Can you find a sufficient condition for the matrix A for that to happen?

Exercise 3*

Consider the following problem:

$$\begin{aligned} &\min x_1 + x_2 \\ \text{subject to: } &\frac{2}{3}x_1 + \frac{1}{2}x_2 \geq 1 \\ &x_1, x_2 \geq 0 \\ &x_1, x_2 \in \mathbb{Z} \end{aligned}$$

Derive a Chvatal-Gomory cut.

Exercise 4

Apply the linear programming branch and bound algorithm to the following instance of the 0-1 knapsack problem: values $v = [9, 4, 2, 3, 5, 3]$, weights $w = [7, 8, 4, 5, 9, 4]$ and capacity $W = 20$.

Exercise 5

Consider the problem of selecting students for a swimming medley relay team. In Table 1 we show times for each swimming style of five students.
We need to choose a student for each of the four swimming styles such that the total relay time is minimized. Formulate the problem as a MILP and solve it in Python.

Exercise 6

In this exercise, you have to implement the dynamic programming algorithm for TSP with the help of the code below.
Some typing preliminaries. Cities are points in the 2D plane represented by complex numbers, a builtin type for a two-dimensional data.

Student	backstroke	breaststroke	butterfly	freestyle
A	43.5	47.1	48.4	38.2
B	45.5	42.1	49.6	36.8
C	43.4	39.1	42.1	43.2
D	46.5	44.1	44.5	41.2
E	46.3	47.8	50.4	37.2

Table 1:

```
import functools
import itertools
import pathlib
import random
import time
import math
import re
import matplotlib.pyplot as plt
from collections import Counter, defaultdict, namedtuple
from statistics import mean, median, stdev
from typing import Set, List, Tuple, Iterable, Dict


# Basic concepts
City = complex # e.g.  City(300, 100)
Cities = frozenset # A set of cities
Tour = list # A list of cities visited, in order
TSP = callable # A TSP algorithm is a callable function
Link = Tuple[City, City] # A city-city link

def distance(A: City, B: City) -> float:
    "Distance between two cities"
    return abs(A - B)

def shortest(tours: Iterable[Tour]) -> Tour:
    "The tour with the smallest tour length."
    return min(tours, key=tour_length)

def tour_length(tour: Tour) -> float:
    "The total distances of each link in the tour, including the link from last back to
        first."
    return sum(distance(tour[i], tour[i - 1]) for i in range(len(tour)))

def valid_tour(tour: Tour, cities: Cities) -> bool:
    "Does ‘tour‘ visit every city in ‘cities‘ exactly once?"
    return Counter(tour) == Counter(cities)
```

Then we can randomly generate a problem instance as follows.

```


# generate instance
def random_cities(n, seed=1234, width=9999, height=6666) -> Cities:
    "Make a set of n cities, sampled uniformly from a (width x height) rectangle."
    random.seed((n, seed))
    return Cities(City(random.randrange(width), random.randrange(height))
                  for c in range(n))
```

We can find optimal solutions by exhaustive search, ie, enumeration of all permutations of points.

```
def exhaustive_tsp(cities) -> Tour:
    "Generate all possible tours of the cities and choose the shortest one."
    return shortest(possible_tours(cities))


possible_tours = itertools.permutations
exhaustive_tsp(random_cities(8))
```

Finally, we can visualize the tour.

```
Segment = list # A portion of a tour; it does not loop back to the start.

def plot_tour(tour: Tour, style='bo-', hilite='rs', title=''):
    "Plot every city and link in the tour, and highlight the start city."
    scale = 1 + len(tour) ** 0.5 // 10
    plt.figure(figsize=((3 * scale, 2 * scale)))
    start = tour[0]
    plot_segment([*tour, start], style)
    plot_segment([start], hilite)
    plt.title(title)

def Xs(cities) -> List[float]: "X coordinates"; return [c.real for c in cities]
def Ys(cities) -> List[float]: "Y coordinates"; return [c.imag for c in cities]

def plot_segment(segment: Segment, style='bo:'):
    "Plot every city and link in the segment."
    plt.plot(Xs(segment), Ys(segment), style, linewidth=2/3, markersize=4, clip_on=False)
    plt.axis('scaled'); plt.axis('off')


plot_tour(exhaustive_tsp(random_cities(8)))
```