

# Optimisation Problems Collection

SOICT - HUST (IT3052E)

Manh Duc, Tran Vu

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                         | <b>3</b>  |
| <b>2</b> | <b>Capacitated Vehicle Routing Problem</b>  | <b>4</b>  |
| 2.1      | Problem Statement . . . . .                 | 4         |
| 2.2      | I/O Format . . . . .                        | 4         |
| 2.3      | Solution Approach . . . . .                 | 4         |
| 2.3.1    | Constraints . . . . .                       | 4         |
| 2.3.2    | Objective function . . . . .                | 5         |
| <b>3</b> | <b>Balanced Courses Assisgnment Problem</b> | <b>6</b>  |
| 3.1      | Problem Statement . . . . .                 | 6         |
| 3.2      | I/O Format . . . . .                        | 6         |
| 3.3      | Solution Approach . . . . .                 | 7         |
| 3.3.1    | Constraints . . . . .                       | 7         |
| 3.3.2    | Objective function . . . . .                | 8         |
| <b>4</b> | <b>N-Queens Problem</b>                     | <b>9</b>  |
| 4.1      | Problem Statement . . . . .                 | 9         |
| 4.2      | I/O Format . . . . .                        | 9         |
| 4.3      | Solution Approach . . . . .                 | 9         |
| 4.3.1    | Constraints . . . . .                       | 9         |
| 4.3.2    | Objective function . . . . .                | 9         |
| <b>5</b> | <b>Traveling Salesman Problem</b>           | <b>10</b> |
| 5.1      | Problem Statement . . . . .                 | 10        |
| 5.2      | I/O Format . . . . .                        | 10        |
| 5.3      | Solution Approach . . . . .                 | 10        |
| 5.3.1    | Constraints . . . . .                       | 10        |
| 5.3.2    | Objective function . . . . .                | 11        |
| <b>6</b> | <b>CBUS Problem</b>                         | <b>12</b> |
| 6.1      | Problem Statement . . . . .                 | 12        |
| 6.2      | I/O Format . . . . .                        | 12        |
| 6.3      | Solution Approach . . . . .                 | 12        |
| 6.3.1    | Variables . . . . .                         | 12        |
| 6.3.2    | Constraints . . . . .                       | 13        |
| 6.3.3    | Objective function . . . . .                | 14        |

# **1 Introduction**

This is a collection of solvers developed during the optimisation course at SOICT - HUST (IT3052E). Note that these implementations are drawn from various external resources due to the complexity of constraint programming.

## 2 Capacitated Vehicle Routing Problem

### 2.1 Problem Statement

A fleet of  $K$  identical trucks having capacity  $Q$  need to be scheduled to deliver pepsi packages from a central depot 0 to clients  $1, 2, \dots, n$ . Each client  $i$  requests  $d[i]$  packages. The distance from location  $i$  to location  $j$  is  $c[i, j]$ ,  $0 \leq i, j \leq n$ . A delivery solution is a set of routes: each truck is associated with a route, starting from depot, visiting some clients and returning to the depot for delivering requested pepsi packages.

**Constraints:**

- Each client must be visited exactly once
- Total package load per truck cannot exceed capacity  $Q$

**Objective:** Minimize total travel distance

### 2.2 I/O Format

Input:

- Line 1:  $n, K, Q$  ( $2 \leq n \leq 12, 1 \leq K \leq 5, 1 \leq Q \leq 50$ )
- Line 2:  $d[1], \dots, d[n]$  ( $1 \leq d[i] \leq 10$ )
- Line  $i+3$  ( $i=0, \dots, n$ ): the  $i$ th row of the distance matrix  $c$  ( $1 \leq c[i, j] \leq 30$ )

Output: Minimal total travel distance.

E.g.

**Input:**

```
4 2 15
7 7 11 2
0 12 12 11 14
14 0 11 14 14
14 10 0 11 12
10 14 12 0 13
10 13 14 11 0
```

**Output:**

```
70
```

### 2.3 Solution Approach

#### 2.3.1 Constraints

Using CP, we need to formulate the constraints. For this problem, the constraints are:

Each client is visited exactly once

```

1 for j in range(1, n + 1):
2     model.Add(sum(x[i,j,k] for i in range(n + 1) for k in
                    range(K) if i != j) == 1)

```

Total number of packages requested by clients cannot exceed its capacity.

```

1 for k in range(K):
2     model.Add(sum(demands[j] * sum(x[i,j,k] for i in range(n
    + 1) if i != j) for j in range(1, n + 1)) <= Q)

```

For each node, the sent package must be exhaustive. In other words, the number of incoming packages must be equal to the outgoing packages.

```

1 for k in range(K):
2     for j in range(n + 1):
3         model.Add(sum(x[i,j,k] for i in range(n + 1) if i !=
                        j) == sum(x[j,i,k] for i in range(n + 1) if i !=
                        j))

```

No subtour (circular routes that aren't the solution)

```

1     u = {}
2     for i in range(n + 1):
3         for k in range(K):
4             u[i,k] = model.NewIntVar(0, n, f'u_{i}_{k}')
5
6     for i in range(1, n + 1):
7         for j in range(1, n + 1):
8             for k in range(K):
9                 if i != j:
10                    model.Add(u[i,k] - u[j,k] + n * x[i,j,k]
                                <= n - 1)

```

### 2.3.2 Objective function

The objective of this problem is to minimise the total distance

```

1 total_dist = sum(dist[i][j] * x[i,j,k] for i in range(n + 1)
    for j in range(n + 1) for k in range(K) if i != j)
2 model.Minimize(total_dist)

```

## 3 Balanced Courses Assignment Problem

### 3.1 Problem Statement

At the beginning of the semester, the head of a computer science department D have to assign courses to teachers in a balanced way. The department D has  $m$  teachers  $T=1,2,\dots,m$  and  $n$  courses  $C=1,2,\dots,n$ . Each teacher  $t \in T$  has a preference list which is a list of courses he/she can teach depending on his/her specialization. We have known a list of pairs of conflicting two courses that cannot be assigned to the same teacher as these courses have been already scheduled in the same slot of the timetable. The load of a teacher is the number of courses assigned to her/him. How to assign  $n$  courses to  $m$  teacher such that each course assigned to a teacher is in his/her preference list, no two conflicting courses are assigned to the same teacher, and the maximal load is minimal.

### 3.2 I/O Format

Input:

- Line 1: contains two integer  $m$  and  $n$  ( $1 \leq m \leq 10, 1 \leq n \leq 30$ )
- Line  $i+1$ : contains an positive integer  $k$  and  $k$  positive integers indicating the courses that teacher  $i$  can teach ( $\forall i = 1, \dots, m$ )
- Line  $m+2$ : contains an integer  $k$
- Line  $i+m+2$ : contains two integer  $i$  and  $j$  indicating two conflicting courses ( $\forall i = 1, \dots, k$ )

Output: The output contains a unique number which is the maximal load of the teachers in the solution found and the value -1 if not solution found.

E.g.

**Input:**

```
4 12
5 1 3 5 10 12
5 9 3 4 8 12
6 1 2 3 4 9 7
7 1 2 3 5 6 10 11
25
1 2
1 3
1 5
2 4
2 5
2 6
3 5
3 7
3 10
4 6
4 9
5 6
5 7
5 8
6 8
6 9
7 8
7 10
7 11
8 9
8 11
8 12
9 12
10 11
11 12
```

**Output:**

3

### 3.3 Solution Approach

#### 3.3.1 Constraints

The problem can be solved using CP. The constraints are:

Each course must be assigned to a teacher who has it in his/her preference list.

```

1 for course in range(1, n+1):
2     allowed_teachers = [t for t in range(1, m+1) if course
3         in preferences[t]]
4     for teacher in range(1, m+1):
5         if teacher not in allowed_teachers:
6             model.Add(course_teacher[course] != teacher)

```

Two conflicting courses cannot be assigned to the same teacher.

```

1 for c1, c2 in conflicting
2     model.Add(course_teacher[c1] != course_teacher[c2])

```

The main part of the problem is to assign each course to a teacher. This can be done using a boolean variable. We can define them as boolean constraints where if a teach is selected, the course is assigned to him/her. In other words, if the course is assigned to a teacher, the boolean variable is true.

```

1 for teacher in range(1, m+1):
2     teaches_vars = []
3     for course in range(1, n+1):
4         teaches = model.NewBoolVar(f'teaches_{teacher}_{course}')
5         model.Add(course_teacher[course] == teacher).
6             OnlyEnforceIf(teaches)
7         model.Add(course_teacher[course] != teacher).
8             OnlyEnforceIf(teaches.Not())
9         teaches_vars.append(teaches)
10    model.Add(teacher_load[teacher] == sum(teaches_vars))
11    model.Add(max_load >= teacher_load[teacher])

```

### 3.3.2 Objective function

The objective of this problem is to minimise the maximal load of the teachers.

```

1 model.Minimize(max_load)

```



## 4 N-Queens Problem

### 4.1 Problem Statement

Find a solution to place  $n$  queens on a chess board such that no two queens attack each other. A solution is represented by a sequence of variables:  $x[1], x[2], \dots, x[n]$  in which  $x[i]$  is the row of the queen on column  $i$  ( $i = 1, \dots, n$ )

### 4.2 I/O Format

Input:

- Line 1: contains a positive integer  $n$  ( $10 \leq n \leq 10000$ )

Output:

- Line 1: write  $n$
- Line 2: write  $x[1], x[2], \dots, x[n]$  (after each value is a SPACE character)

E.g.

**Input:**

10

**Output:**

10  
1 3 6 8 10 5 9 2 4 7

### 4.3 Solution Approach

#### 4.3.1 Constraints

The problem can be solved using CP. The constraints are: Each row must have exactly one queen.

```
1 model.AddAllDifferent(x)
```

No two queens can be on the same diagonal.

```
1 for i in range(n):  
2     for j in range(i + 1, n):  
3         model.Add(x[i] - x[j] != i - j) # Upward diagonal  
4         model.Add(x[i] - x[j] != j - i) # Downward diagonal
```

#### 4.3.2 Objective function

The objective of this problem is to find a solution.

```
1 solver = cp_model.CpSolver()  
2 status = solver.Solve(model)
```

## 5 Traveling Salesman Problem

### 5.1 Problem Statement

There are  $n$  cities  $1, 2, \dots, n$ . The travel distance from city  $i$  to city  $j$  is  $c(i, j)$ , for  $i, j = 1, 2, \dots, n$ . A person departs from city 1, visits each city  $2, 3, \dots, n$  exactly once and comes back to city 1. Find the itinerary for that person so that the total travel distance is minimal. A solution is represented by a sequence of  $n$  variables  $x[1], x[2], \dots, x[n]$  in which the tour is:  $x[1] \rightarrow x[2] \rightarrow \dots \rightarrow x[n] \rightarrow x[1]$ .

### 5.2 I/O Format

Input:

- Line 1: a positive integer  $n$  ( $1 \leq n \leq 200$ )
- Line  $i+1$  ( $i = 1, \dots, n$ ): contains the  $i$ th row of the distance matrix  $x$  (elements are separated by a SPACE character)

Output:

- Line 1: write the value  $n$
- Line 2: write  $x[1], x[2], \dots, x[n]$  (after each element, there is a SPACE character)

E.g.

**Input:**

```
4
0 1 1 9
1 0 9 3
1 9 0 2
9 3 2 0
```

**Output:**

```
4
1 3 4 2
```

### 5.3 Solution Approach

The problem can be solved using CP. We thus need to formulate the constraints as follows:

#### 5.3.1 Constraints

Each city must be visited exactly once. To do this, we can use the circuit constraint.

- **arcs**: a list of tuples  $(i, j, lit)$  where  $lit$  is a boolean variable indicating whether the person goes from city  $i$  to city  $j$ .

```

1  # Create variables for successor of each city
2  succ = [model.NewIntVar(0, n-1, f'succ_{i}') for i in range(
      n)]
3
4  # Create circuit
5  for i in range(n):
6      for j in range(n):
7          if i != j:
8              lit = model.NewBoolVar(f'arc_{i}_{j}')
9              model.Add(succ[i] == j).OnlyEnforceIf(lit)
10             arcs.append((i, j, lit))
11 model.AddCircuit(arcs)

```

### 5.3.2 Objective function

The objective of this problem is to minimise the total distance.

- `distance_terms`: a list of variables representing the distance between each pair of cities.

```

1  # Calculate total distance
2  total_distance = model.NewIntVar(0, sum(max(row) for row in
      dist), 'total_distance')
3
4  for i in range(n):
5      dist_var = model.NewIntVar(0, max(max(row) for row in
      dist), f'dist_{i}')
6      distance_terms.append(dist_var)
7      model.AddElement(succ[i], dist[i], dist_var)
8
9  model.Add(total_distance == sum(distance_terms))
10 model.Minimize(total_distance)

```

## 6 CBUS Problem

### 6.1 Problem Statement

There are  $n$  passengers  $1, 2, \dots, n$ . The passenger  $i$  want to travel from point  $i$  to point  $i + n$  ( $i = 1, 2, \dots, n$ ). There is a bus located at point 0 and has  $k$  places for transporting the passengers (it means at any time, there are at most  $k$  passengers on the bus). You are given the distance matrix  $c$  in which  $c(i, j)$  is the traveling distance from point  $i$  to point  $j$  ( $i, j = 0, 1, \dots, 2n$ ). Compute the shortest route for the bus, serving  $n$  passengers and coming back to point 0.

### 6.2 I/O Format

Input:

- Line 1 contains  $n$  and  $k$  ( $1 \leq n \leq 11, 1 \leq k \leq 10$ )
- Line  $i+1$  ( $i = 1, 2, \dots, 2n + 1$ ) contains the  $(i - 1)$ th line of the matrix  $c$  (rows and columns are indexed from  $0, 1, 2, \dots, 2n$ ).

Output:

- Unique line containing the length of the shortest route.

E.g.

**Input:**

```
3 2
0 8 5 1 10 5 9
9 0 5 6 6 2 8
2 2 0 3 8 7 2
5 3 4 0 3 2 7
9 6 8 7 0 9 10
3 8 10 6 5 0 2
3 4 4 5 2 2 0
```

**Output:**

```
25
```

### 6.3 Solution Approach

The problem can be solved using CP.

#### 6.3.1 Variables

- **x**: Boolean list of variables representing visit from  $i$  to  $j$ .
- **visit**: The order of visit.
- **load**: a list of variables representing the number of passengers on the bus.

```

1 x = {} # Binary variable for edges
2 for i in range(2*n + 1):
3     for j in range(2*n + 1):
4         if i != j:
5             x[i,j] = model.NewBoolVar(f'x_{i}_{j}')
6
7 visit = [model.NewIntVar(0, 2*n, f'visit_{i}')] for i in
           range(2*n + 1)]
8 load = [model.NewIntVar(0, k, f'load_{i}')] for i in range(2*
           n + 1)]

```

### 6.3.2 Constraints

Circuit constraints

- The bus must start at point 0 and return to point 0.
- Each point must be visited exactly once.
- No subtour.
- Pickup before dropoff.

```

1 for i in range(2*n + 1):
2     # Outbound constraint: From each point, there must be
3     # exactly one outgoing edge.
4     model.Add(sum(x[i,j] for j in range(2*n + 1) if j != i)
5               == 1)
6
7     # Inbound constraint: To each point, there must be
8     # exactly one incoming edge.
9     model.Add(sum(x[j,i] for j in range(2*n + 1) if j != i)
10              == 1)
11
12 # MTZ subtour elimination
13 for i in range(2*n + 1):
14     for j in range(2*n + 1):
15         if i != j and i != 0 and j != 0:
16             model.Add(visit[i] - visit[j] + (2*n + 1) * x[i,
17                       j] <= 2*n)
18
19 # Pickup before delivery
20 model.Add(visit[0] == 0) # Start at depot
21 for i in range(1, n+1):
22     model.Add(visit[i] < visit[i+n]) # Pickup i before
23     # delivery i+n

```

Load constraints

- The number of passengers on the bus must be less than or equal to k.

```

1 # Load tracking
2 model.Add(load[0] == 0) # Start empty
3 for i in range(1, n+1):
4     # After pickup i
5     for j in range(2*n + 1):
6         if j != i:
7             model.Add(load[j] >= load[i] + 1).OnlyEnforceIf(
8                 x[i,j])
9     # After delivery i+n
10    for j in range(2*n + 1):
11        if j != i+n:
12            model.Add(load[j] >= load[i+n] - 1).
13                OnlyEnforceIf(x[i+n,j])

```

For each passenger  $i$ , when picking up  $i$ , the load must be greater than initial state. On the other hand, when dropping off  $i$ , the load must be less than the initial state.

### 6.3.3 Objective function

The objective of this problem is to minimise the total distance. We do this by multiplying the distance matrix with the binary variables.

```

1 total_distance = model.NewIntVar(0, sum(max(row) for row in
2   c), 'total_distance')
3 model.Add(total_distance == sum(c[i, j] * x[i, j] for i in
4   range(2*n + 1) for j in range(2*n + 1) if i != j))
5 model.Minimize(total_distance)

```