jupyter problem-set-4 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown Validate

```
    return [ [uniform(0, 5) if i != j else None for j in range(n)] for i

for trial in range(20):
    print(f'Trial # {trial}')
    n = randint(6, 11)
    cost_matrix = create_cost(n)
    constraints = [(1, 3), (4, 2), (n-1, 1), (n-2, 2)]
    tour = tsp_with_extra_constraints(n, cost_matrix, constraints)
    i = 0
    tour_cost = 0
    for j in tour[1:]:
        tour_cost += cost_matrix[i][j]
        i = j
    tour_cost += cost_matrix[i][0]
    print(f'Tour:{tour}')
    print(f'Cost of your tour: {tour_cost}')
    for i in range(n):
        num = sum([1 if j == i else 0 for j in tour])
        assert  num == 1, f'Vertex {i} repeats {num} times in tour'
    for (i, j) in constraints:
        assert tour.index(i) < tour.index(j), f'Tour does not respect co
print('Test Passed (10 points)')
```

```
Trial # 0
Tour:[0, 4, 6, 7, 9, 8, 2, 1, 3, 5]
Cost of your tour: 12.916585371664487
Trial # 1
Tour:[0, 5, 1, 4, 3, 2]
Cost of your tour: 11.120757024777006
Trial # 2
Tour:[0, 6, 7, 4, 1, 3, 2, 5]
Cost of your tour: 10.027905344045518
Trial # 3
```

Grades

Close

Passed • Grade Received: 100%

Submissions

November 19, 2023 6:46 AM PST (Highest Grade)

Problem 1

We saw how to solve TSPs in this module and in particular presented two approaches to encode a TSP as an integer linear program. In this problem, we will ask you to adapt the TSP solution to the related problem of k Travelling Salespeople Problem (k-TSP).

Let G be a complete graph with n vertices that we will label $0, \ldots, n-1$ (keeping Python array indexing in mind). Our costs are specified using a matrix C wherein $C_{i,j}$ is the cost of the edge from vertex i to j for $i \neq j$.

In this problem, we have $k \geq 1$ salespeople who must start from vertex 0 of the graph (presumably the location of the sales office) and together visit every location in the graph, each returning back to vertex 0. Each location must be visited exactly once by some salesperson in the team. Therefore, other than vertex 0 (the start/end vertex of every salesperson's tour), no two salesperson tours have a vertex in common. Notice that for $k = 1$, this is just the regular TSP problem we have studied.

Also, all k salespeople must be employed in the tour. In other words, if we have $k = 3$ then each salesperson must start at 0 and visit a sequence of one or more vertices and come back to 0. No salesperson can be "idle".

Example-1

Consider a graph with 5 nodes and the following cost matrix:

Vertices	0	1	2	3	4
0	–	3	4	3	5
1	1	–	2	4	1
2	2	1	–	5	4
3	1	1	5	–	4
4	2	1	3	5	–

For instance $C_{2,3}$ the cost of edge from vertex 2 to 3 is 5. The – in the diagonal entries simply tells us that we do not care what goes in there since we do not have self-loops in the graph.

The optimal 2-TSP tour for $k = 2$ salespeople is shown below.

- Salesperson # 1: $0 \to 2 \to 1 \to 4 \to 0$.
- Salesperson # 2: $0 \to 3 \to 0$.

The total cost of the edges traversed by the two salespeople equals 12.

For $k = 3$, the optimal 3– TSP tour is as shown below.

- Salesperson # 1: $0 \to 1 \to 4$,
- Salesperson # 2: $0 \to 2$,
- Salesperson # 3: $0 \to 3$.

The total cost is 16.

The objective of this problem is to formulate an ILP using the MTZ approach.

Problem 1A (MTZ approach)

We will use the same ILP setup as in our notes (see the notes on Exact Approaches to TSP that includes the ILP encodings we will use in this problem).

- Decision variables $x_{i,j}$ for $i \neq j$ denoting that the tour traverses the edge from i to j.
- Time stamps $t_1, \ldots, t_{n-1}$. The start/end vertex 0 does not get a time stamp.

Modify the MTZ approach to incorporate the fact that k salespeople are going to traverse the graph.

(A) Degree Constraints

What do the new degree constraints look like? Think about how many edges in the tour will need to enter/leave each vertex? Note that you may have to treat vertex 0 differently from the other vertices of the graph.

Your answer below is not graded. However you are encouraged to write it down and check with the answers to select problems provided at the end.

Student's answer — Score: 0.0 / 0.0 (Top)

YOUR ANSWER HERE

Comments:
No response.

(B) Time Stamp Constraints

Formulate the time stamp constraints for the k-TSP problem. Think about how you would need to change them to eliminate subtour.

Your answer below is not graded. However you are encouraged to write it down and check with the answers to select problems provided at the end.

Student's answer — Score: 0.0 / 0.0 (Top)

YOUR ANSWER HERE

Comments:
No response.

(C) Implement

Complete the implementation of the function `k_tsp_mtz_encoding(n, k, cost_matrix)` below. It follows the same input convention as the code supplied in the notes. The input `n` denotes the size of the graph with vertices labeled `0` ,.., `n-1` , `k` is the number of salespeople, and `cost_matrix` is a list of lists wherein `cost_matrix[i][j]` is the edge cost to go from `i` to `j` for `i != j` . Your code must avoid accessing `cost_matrix[i][i]` to avoid bugs. These entries will be supplied as `None` in the test cases.

Your code must return a list `lst` that has exactly k lists in it, wherein `lst[j]` represents the locations visited by the j^{th} salesperson.

For the example above, for $k = 2$, your code must return

```
[ [0, 2, 1, 4], [0, 3] ]
```

For the example above, for $k = 3$, your code must return

```
[ [0, 1, 4], [0, 2], [0, 3] ]
```

In [1]:

Student's answer (Top)

```
from pulp import *
def k_tsp_mtz_encoding(n, k, cost_matrix):
    # check inputs are OK
    assert 1 <= k < n
    assert len(cost_matrix) == n, f'Cost matrix is n
ot {n}x{n}'
    assert all(len(cj) == n for cj in cost_matrix),
f'Cost matrix is not {n}x{n}'
    prob = LpProblem('kTSP', LpMinimize)
    # finish your implementation here
    # your code must return a list of k-lists [ [0,
i1, ..., il], [0, j1,...,jl],...] wherein
    # the ith entry in our list of lists represents
the
    # tour undertaken by the ith salesperson
    # your code here
    cost_matrix = [[0 if cost is None else cost for
cost in row] for row in cost_matrix]


    binary_vars = [
        [LpVariable(f'x_{i}_{j}', cat='Binary') if i
!= j else None for j in range(n)]
        for i in range(n)
    ]

    # add time stamps for ranges 1 .. n (skip vertex
0 for timestamps)
    time_stamps = [LpVariable(f't_{j}', lowBound=0,
upBound=n, cat='Continuous') for j in range(1, n)]


    # Create and add the objective function
    objective_function = lpSum([lpSum([xij * cj if x
ij is not None else 0 for (xij, cj) in zip(brow, cro
w)]) for (brow, crow) in zip(binary_vars, cost_matri
x)])
    prob += objective_function

    # Set 1: For Node[0], k edges leave and k edges
enter
    prob += lpSum([xij for xij in binary_vars[0] if
xij is not None]) == k
    prob += lpSum([binary_vars[j][0] for j in range
(n) if j != 0]) == k
```

```
    # Set 2: For Node[1:n], 1 edge leaves and 1 edge
enters
    for i in range(1, n):
        prob += lpSum([xij for xij in binary_vars[i]
if xij is not None]) == 1
        prob += lpSum([binary_vars[j][i] for j in ra
nge(n) if j != i]) == 1

    # Add time stamp constraints
    for i in range(1, n):
        for j in range(1, n):
            if i == j:
                continue
            xij = binary_vars[i][j]
            ti = time_stamps[i - 1]
            tj = time_stamps[j - 1]
            prob += tj >= ti + xij - (1 - xij) * (n
+ 1)

    # Solve the problem
    status = prob.solve(PULP_CBC_CMD(msg=False))
    assert status == constants.LpStatusOptimal, f'Un
expected non-optimal status {status}'

    # Extract tours for each salesman
    tours = extract_k_tsp_tours(binary_vars, n, k)
    return tours

def extract_k_tsp_tours(binary_vars, n, k):
    tours = []

    # Initialize a list to track visited cities
    visited = [False] * n

    for _ in range(k):
        tour = []
        current_city = 0  # Start from city 0
        while True:
            tour.append(current_city)
            visited[current_city] = True

            # Find the next unvisited city in the cu
rrent tour
            next_city = None
            for j in range(n):
                if not visited[j] and binary_vars[cu
rrent_city][j].varValue == 1:
                    next_city = j
                    break

            if next_city is not None:
                current_city = next_city
            else:
                break  # Finished the current tour

        tours.append(tour)

        # Find the next unvisited city for the next
tour
        for i in range(n):
            if not visited[i]:
                current_city = i
                break

    return tours
```

In [2]:

Grade cell: `cell-f102dee236a23785` Score: 5.0 / 5.0 (Top)

```
cost_matrix=[ [None,3,4,3,5],
              [1, None, 2,4, 1],
              [2, 1, None, 5, 4],
              [1, 1, 5, None, 4],
              [2, 1, 3, 5, None] ]
n=5
k=2
all_tours = k_tsp_mtz_encoding(n, k, cost_matrix)
print(f'Your code returned tours: {all_tours}')
assert len(all_tours) == k, f'k={k} must yield two t
ours -- your code returns {len(all_tours)} tours ins
tead'

tour_cost = 0
for tour in all_tours:
    assert tour[0] == 0, 'Each salesperson tour must
start from vertex 0'
    i = 0
    for j in tour[1:]:
        tour_cost += cost_matrix[i][j]
        i = j
    tour_cost += cost_matrix[i][0]

print(f'Tour cost obtained by your code: {tour_cos
t}')
assert abs(tour_cost - 12) <= 0.001, f'Expected tour
cost is 12, your code returned {tour_cost}'
for i in range(1, n):
    is_in_tour = [ 1 if i in tour else 0 for tour in
all_tours]
    assert sum(is_in_tour) == 1, f' vertex {i} is in
{sum(is_in_tour)} tours -- this is incorrect'

print('test passed: 3 points')
```

**Congratulations! All test cases in this ce
ll passed.**

In [3]:

Grade cell: `cell-8f9a204f86c84b7f` Score: 3.33 / 3.33 (Top)

```
cost_matrix=[ [None,3,4,3,5],
              [1, None, 2,4, 1],
              [2, 1, None, 5, 4],
              [1, 1, 5, None, 4],
              [2, 1, 3, 5, None] ]
n=5
k=3
all_tours = k_tsp_mtz_encoding(n, k, cost_matrix)
print(f'Your code returned tours: {all_tours}')
assert len(all_tours) == k, f'k={k} must yield two t
ours -- your code returns {len(all_tours)} tours ins
tead'

tour_cost = 0
for tour in all_tours:
    assert tour[0] == 0, 'Each salesperson tour must
start from vertex 0'
    i = 0
    for j in tour[1:]:
        tour_cost += cost_matrix[i][j]
        i = j
    tour_cost += cost_matrix[i][0]

print(f'Tour cost obtained by your code: {tour_cos
t}')
assert abs(tour_cost - 16) <= 0.001, f'Expected tour
cost is 16, your code returned {tour_cost}'
for i in range(1, n):
    is_in_tour = [ 1 if i in tour else 0 for tour in
all_tours]
    assert sum(is_in_tour) == 1, f' vertex {i} is in
{sum(is_in_tour)} tours -- this is incorrect'

print('test passed: 2 points')
```

**Congratulations! All test cases in this ce
ll passed.**

In [4]:

Grade cell: `cell-b8417f007421c582` Score: 5.0 / 5.0 (Top)

```
cost_matrix = [
 [None, 1, 1, 1, 1, 1, 1, 1],
    [0, None, 1, 2, 1, 1, 1, 1],
    [1, 0, None, 1, 2, 2, 2, 1],
    [1, 2, 2, None, 0, 1, 2, 1],
    [1, 1, 1, 1, None, 1, 1, 1],
    [0,  1, 2, 1, 1, None, 1, 1],
    [1, 0,  1, 2, 2, 2,None, 1],
    [1, 2, 2, 0, 1, 2, 1, None],
]
n = 8
k = 2

all_tours = k_tsp_mtz_encoding(n, k, cost_matrix)
print(f'Your code returned tours: {all_tours}')
assert len(all_tours) == k, f'k={k} must yield two t
ours -- your code returns {len(all_tours)} tours ins
tead'

tour_cost = 0
for tour in all_tours:
    assert tour[0] == 0, 'Each salesperson tour must
start from vertex 0'
    i = 0
    for j in tour[1:]:
        tour_cost += cost_matrix[i][j]
        i = j
    tour_cost += cost_matrix[i][0]

print(f'Tour cost obtained by your code: {tour_cos
t}')
assert abs(tour_cost - 4) <= 0.001, f'Expected tour
cost is 4, your code returned {tour_cost}'
for i in range(1, n):
    is_in_tour = [ 1 if i in tour else 0 for tour in
all_tours]
    assert sum(is_in_tour) == 1, f' vertex {i} is in
{sum(is_in_tour)} tours -- this is incorrect'

print('test passed: 3 points')
```

**Congratulations! All test cases in this ce
ll passed.**

In [5]:

Grade cell: `cell-03146af7d96a27d5` Score: 3.33 / 3.33 (Top)

```
cost_matrix = [
 [None, 1, 1, 1, 1, 1, 1, 1],
    [0, None, 1, 2, 1, 1, 1, 1],
    [1, 0, None, 1, 2, 2, 2, 1],
    [1, 2, 2, None, 0, 1, 2, 1],
    [1, 1, 1, 1, None, 1, 1, 1],
    [0,  1, 2, 1, 1, None, 1, 1],
    [1, 0,  1, 2, 2, 2,None, 1],
    [1, 2, 2, 0, 1, 2, 1, None],
]
n = 8
k = 4
```

```
all_tours = k_tsp_mtz_encoding(n, k, cost_matrix)
print(f'Your code returned tours: {all_tours}')
assert len(all_tours) == k, f'k={k} must yield two t
ours -- your code returns {len(all_tours)} tours ins
tead'

tour_cost = 0
for tour in all_tours:
    assert tour[0] == 0, 'Each salesperson tour must
start from vertex 0'
    i = 0
    for j in tour[1:]:
        tour_cost += cost_matrix[i][j]
        i = j
    tour_cost += cost_matrix[i][0]

print(f'Tour cost obtained by your code: {tour_cos
t}')
assert abs(tour_cost - 6) <= 0.001, f'Expected tour
cost is 6, your code returned {tour_cost}'
for i in range(1, n):
    is_in_tour = [ 1 if i in tour else 0 for tour in
all_tours]
    assert sum(is_in_tour) == 1, f' vertex {i} is in
{sum(is_in_tour)} tours -- this is incorrect'

print('test passed: 2 points')
```

```
Congratulations! All test cases in this ce
ll passed.
```

In [6]:

Grade cell: `cell-f903e8e7ad9e3b86` Score: 25.0 / 25.0 (Top)

```
from random import uniform, randint

def create_cost(n):
    return [ [uniform(0, 5) if i != j else None for
j in range(n)] for i in range(n)]

for trial in range(5):
    print(f'Trial # {trial}')
    n = randint(5, 11)
    k = randint(2, n//2)
    print(f' n= {n}, k={k}')
    cost_matrix = create_cost(n)
    print('cost_matrix = ')
    print(cost_matrix)
    all_tours = k_tsp_mtz_encoding(n, k, cost_matri
x)
    print(f'Your code returned tours: {all_tours}')
    assert len(all_tours) == k, f'k={k} must yield t
wo tours -- your code returns {len(all_tours)} tours
instead'

    tour_cost = 0
    for tour in all_tours:
        assert tour[0] == 0, 'Each salesperson tour
must start from vertex 0'
        i = 0
        for j in tour[1:]:
            tour_cost += cost_matrix[i][j]
            i = j
        tour_cost += cost_matrix[i][0]

    print(f'Tour cost obtained by your code: {tour_c
ost}')
    #assert abs(tour_cost - 6) <= 0.001, f'Expected
tour cost is 6, your code returned {tour_cost}'
    for i in range(1, n):
        is_in_tour = [ 1 if i in tour else 0 for tou
r in all_tours]
        assert sum(is_in_tour) == 1, f' vertex {i} i
s in {sum(is_in_tour)} tours -- this is incorrect'
    print('------')
print('test passed: 15 points')
```

```
Congratulations! All test cases in this ce
ll passed.
```

Problem 1 B

Notice that in previous part, it happens that with $k = 4$ salespeople, we actually get a worse cost than using $k = 3$ people. You can try out a few examples to convince yourself as to why this happens.

We wish to modify the problem to allow salespeople to idle. In other words, although we input k salespeople, the tour we construct may involve $1 \leq l \leq k$ salespeople.

Modify the ILP formulation from the previous problem to solve the problem of up to k people rather than exactly k salespeople. Note that we still require that every vertex be visited exactly once by some salesperson.

Complete the implementation of the function `upto_k_tsp_mtz_encoding(n, k, cost_matrix)` below. It follows the same input convention as previous problem but note that we are now computing a tour with at most k salespeople. In other words, not all salespeople need be employed in the tour.

Your code must return a list `lst` that has less than or equal to k lists, where in `lst[i]` represents the locations visited by the i^{th}

wherein `tsl[j]` represents the locations visited by the j^{th} salesperson.

For Example-1 from the previous part above, for $k = 2$ or $k = 3$, your code must return

```
[ [0, 3, 1, 4, 2] ]
```

As it turns out, in this example a single salesperson suffices to yield optimal cost.

In [7]:

Student's answer (Top)

```
from pulp import *
def upto_k_tsp_mtz_encoding(n, k, cost_matrix):
    # check inputs are OK
    assert 1 <= k < n
    assert len(cost_matrix) == n, f'Cost matrix is not {n}x{n}'
    assert all(len(cj) == n for cj in cost_matrix), f'Cost matrix is not {n}x{n}'
    #prob = LpProblem('kTSP', LpMinimize)
    # finish your implementation here
    # your code must return a list of k-lists [ [0, i1, ..., il], [0, j1,...,jl],...] wherein
    # the ith entry in our list of lists represents the
    # tour undertaken by the ith salesperson
    # your code here
    cost_matrix = [[0 if cost is None else cost for cost in row] for row in cost_matrix]

    tours = []
    min_cost = float('inf')
    for l in range(k,0,-1):

        prob = LpProblem('kTSP', LpMinimize)


        binary_vars = [[LpVariable(f'x_{i}_{j}', cat='Binary') if i != j else None for j in range(n)]
        for i in range(n)]

        # add time stamps for ranges 1 .. n (skip vertex 0 for timestamps)
        time_stamps = [LpVariable(f't_{j}', lowBound=0, upBound=n, cat='Continuous') for j in range(1, n)]


        # Create and add the objective function
        objective_function = lpSum([lpSum([xij * cj if xij is not None else 0 for (xij, cj) in zip(brow, crow)]) for (brow, crow) in zip(binary_vars, cost_matrix)])
        prob += objective_function


        prob += lpSum([xij for xij in binary_vars[0] if xij is not None]) == l
        prob += lpSum([binary_vars[j][0] for j in range(n) if j != 0]) == l

        # Set 2: For Node[1:n], 1 edge leaves and 1 edge enters
        for i in range(1, n):
            prob += lpSum([xij for xij in binary_vars[i] if xij is not None]) == 1
            prob += lpSum([binary_vars[j][i] for j in range(n) if j != i]) == 1

        # Add time stamp constraints
        for i in range(1, n):
            for j in range(1, n):
                if i == j:
                    continue
                xij = binary_vars[i][j]
                ti = time_stamps[i - 1]
                tj = time_stamps[j - 1]
                prob += tj >= ti + xij - (1 - xij) * (n + 1)

        # Solve the problem
        status = prob.solve(PULP_CBC_CMD(msg=False))
        assert status == constants.LpStatusOptimal, f'Unexpected non-optimal status {status}'

        current_cost = value(prob.objective)
        if current_cost<min_cost:
            min_cost = current_cost
            # Extract tours for each salesman
            tours = extract_k_tsp_tours(binary_vars, n, l)

    return tours

def extract_k_tsp_tours(binary_vars, n, k):
    tours = []

    # Initialize a list to track visited cities
    visited = [False] * n

    for _ in range(k):
        tour = []
        current_city = 0  # Start from city 0
        while True:
            tour.append(current_city)
            visited[current_city] = True
```

```
            # Find the next unvisited city in the cu
rrent tour
            next_city = None
            for j in range(n):
                if not visited[j] and binary_vars[cu
rrent_city][j].varValue == 1:
                    next_city = j
                    break

            if next_city is not None:
                current_city = next_city
            else:
                break  # Finished the current tour

        tours.append(tour)

        # Find the next unvisited city for the next
tour
        for i in range(n):
            if not visited[i]:
                current_city = i
                break

    return tours
```

In [8]:

Grade cell: `cell-934ddb639bc6c01d` Score: 5.0 / 5.0 (Top)

```
cost_matrix=[ [None,3,4,3,5],
              [1, None, 2,4, 1],
              [2, 1, None, 5, 4],
              [1, 1, 5, None, 4],
              [2, 1, 3, 5, None] ]
n=5
k=3
all_tours = upto_k_tsp_mtz_encoding(n, k, cost_matri
x)
print(f'Your code returned tours: {all_tours}')
assert len(all_tours) <= k, f'<= {k} tours -- your c
ode returns {len(all_tours)} tours instead'

tour_cost = 0
for tour in all_tours:
    assert tour[0] == 0, 'Each salesperson tour must
start from vertex 0'
    i = 0
    for j in tour[1:]:
        tour_cost += cost_matrix[i][j]
        i = j
    tour_cost += cost_matrix[i][0]

assert len(all_tours) == 1, f'In this example, just
one salesperson is needed to optimally visit all ver
tices. Your code returns {len(all_tours)}'
print(f'Tour cost obtained by your code: {tour_cos
t}')
assert abs(tour_cost - 10) <= 0.001, f'Expected tour
cost is 10, your code returned {tour_cost}'
for i in range(1, n):
    is_in_tour = [ 1 if i in tour else 0 for tour in
all_tours]
    assert sum(is_in_tour) == 1, f' vertex {i} is in
{sum(is_in_tour)} tours -- this is incorrect'

print('test passed: 3 points')
```

**Congratulations! All test cases in this ce
ll passed.**

In [9]:

Grade cell: `cell-17fd7b9a31ded1eb` Score: 5.0 / 5.0 (Top)

```
cost_matrix = [
 [None, 1, 1, 1, 1, 1, 1, 1],
    [0, None, 1, 2, 1, 1, 1, 1],
    [1, 0, None, 1, 2, 2, 2, 1],
    [1, 2, 2, None, 0, 1, 2, 1],
    [1, 1, 1, 1, None, 1, 1, 1],
    [0,  1, 2, 1, 1, None, 1, 1],
    [1, 0,  1, 2, 2, 2,None, 1],
    [1, 2, 2, 0, 1, 2, 1, None],
]
n = 8
k = 5

all_tours = upto_k_tsp_mtz_encoding(n, k, cost_matri
x)
print(f'Your code returned tours: {all_tours}')
assert len(all_tours) <= k, f'k={k} must yield two t
ours -- your code returns {len(all_tours)} tours ins
tead'

tour_cost = 0
for tour in all_tours:
    assert tour[0] == 0, 'Each salesperson tour must
start from vertex 0'
    i = 0
    for j in tour[1:]:
        tour_cost += cost_matrix[i][j]
        i = j
    tour_cost += cost_matrix[i][0]

print(f'Tour cost obtained by your code: {tour_cos
t}')
assert abs(tour_cost - 4) <= 0.001, f'Expected tour
```

```
cost is 4, your code returned {tour_cost}'
for i in range(1, n):
    is_in_tour = [ 1 if i in tour else 0 for tour in
all_tours]
    assert sum(is_in_tour) == 1, f' vertex {i} is in
{sum(is_in_tour)} tours -- this is incorrect'

print('test passed: 3 points')
```

```
Congratulations! All test cases in this ce
ll passed.
```

In [10]:

Grade cell: `cell-4910fe35d0e1c603` Score: 6.67 / 6.67 (Top)

```
from random import uniform, randint

def create_cost(n):
    return [ [uniform(0, 5) if i != j else None for
j in range(n)] for i in range(n)]

for trial in range(20):
    print(f'Trial # {trial}')
    n = randint(5, 11)
    k = randint(2, n//2)
    print(f' n= {n}, k={k}')
    cost_matrix = create_cost(n)
    print('cost_matrix = ')
    print(cost_matrix)
    all_tours = upto_k_tsp_mtz_encoding(n, k, cost_m
atrix)
    print(f'Your code returned tours: {all_tours}')
    assert len(all_tours) <= k, f'k={k} must yield t
wo tours -- your code returns {len(all_tours)} tours
instead'

    tour_cost = 0
    for tour in all_tours:
        assert tour[0] == 0, 'Each salesperson tour
must start from vertex 0'
        i = 0
        for j in tour[1:]:
            tour_cost += cost_matrix[i][j]
            i = j
        tour_cost += cost_matrix[i][0]

    print(f'Tour cost obtained by your code: {tour_c
ost}')
    #assert abs(tour_cost - 6) <= 0.001, f'Expected
tour cost is 6, your code returned {tour_cost}'
    for i in range(1, n):
        is_in_tour = [ 1 if i in tour else 0 for tou
r in all_tours]
        assert sum(is_in_tour) == 1, f' vertex {i} i
s in {sum(is_in_tour)} tours -- this is incorrect'
    print('------')
print('test passed: 4 points')
```

```
Congratulations! All test cases in this ce
ll passed.
```

Problem 2 (10 points)

We noted the use of Christofides algorithm for metric TSP. We noted that for non-metric TSPs it does not work. In fact, the shortcutting used in Christofides algorithm can be *arbitrarily* bad for a TSP that is symmetric but fails to be a metric TSP.

In this example, we would like you to frame a symmetric TSP instance ($C_{ij} = C_{ji}$) with 5 vertices wherein the algorithm obtained by "shortcutting" the minimum spanning tree (MST), that would be a 2-factor approximation for metric TSP, yields an answer that can be quite "far off" from the optimal solution.

Enter a **symmetric** cost-matrix for the TSP below as a 5x5 matrix as a list of lists following convention in our notes. such that the optimal answer is at least 10^6 times smaller than that obtained by the TSP-based approximation. We will test your answer by running the TSP with shortcutting algorithm.

Hint: Force the edges $(0,1),(1,2),(2,3)$ and $(3,4)$ to be the minimum spanning tree. But make the weight of the edge form 4 back to 0 very high.

Note: this problem is tricky and requires you to be very familiar with how Christofides algorithm works. It may be wise to attempt the remaining problems first before this one. Do not worry about the diagonal entry of your matrices.

In [11]:

Student's answer (Top)

```
# Write down the cost matrix as a list of lists. Rem
ember, we want an example with $5$ vertices
# cost_matrix = [[..], [...],[...]]
# The cost matrix should represent a TSP instance su
ch that the optimal answer is 10^6 smaller than the
# answer obtained by shortcutting a minimum spanning
tree.
# your code here
cost matrix=[
```

```
    [0, 1, 2, 4, 2e7],
    [1, 0, 1, 3, 6],
    [2, 1, 0, 1, 5],
    [4, 3, 1, 0, 1],
    [2e7, 6, 5, 1, 0]
]
```

In [12]:

Grade cell: `cell-62a3ff5de5ea71ff` Score: 5.0 / 5.0 (Top)

```
# check that the cost matrix is symmetric.
assert len(cost_matrix) == 5, f'Cost matrix must hav
e 5 rows. Yours has {len(cost_matrix)} rows'
assert all(len(cj) == 5 for cj in cost_matrix), f'Ea
ch row of the cost matrix must have 5 entries.'
for i in range(5):
    for j in range(i):
        assert cost_matrix[i][j] == cost_matrix[j]
[i], f'Cost matrix fails to be symmetric at entries
{(i,j)} and {(j,i)}'
print('Structure of your cost matrix looks OK (3 poi
nts).')
```

```
Congratulations! All test cases in this ce
ll passed.
```

Please ensure that you run the two cells below or else, your tests will fail.

In [13]:

```
# MST based tsp approximation
import networkx as nx

# This code implements the simple MST based shortcutt
ing approach that would yield factor of 2
# approximation for metric TSPs.
def minimum_spanning_tree_tsp(n, cost_matrix):
    G = nx.Graph()
    for i in range(n):
        for j in range(i):
            G.add_edge(i, j, weight=cost_matrix[i]
[j])
    T = nx.minimum_spanning_tree(G)
    print(f'MST for your graph has the edges {T.edge
s}')
    mst_cost = 0
    mst_dict = {} # store mst as a dictionary
    for (i,j) in T.edges:
        mst_cost += cost_matrix[i][j]
        if i in mst_dict:
            mst_dict[i].append(j)
        else:
            mst_dict[i] = [j]
        if j in mst_dict:
            mst_dict[j].append(i)
        else:
            mst_dict[j] = [i]
    print(f'MST cost: {mst_cost}')
    print(mst_dict)
    # Let's form a tour with short cutting
    def traverse_mst(tour_so_far, cur_node):
        assert cur_node in mst_dict
        next_nodes = mst_dict[cur_node]
        for j in next_nodes:
            if j in tour_so_far:
                continue
            tour_so_far.append(j)
            traverse_mst(tour_so_far, j)
        return
    tour = [0]
    traverse_mst(tour, 0)
    i = 0
    tour_cost = 0
    for j in tour[1:]:
        tour_cost += cost_matrix[i][j]
        i = j
    tour_cost += cost_matrix[i][0]
    return tour, tour_cost
```

In [14]:

```
# optimal TSP tour taken from our notes using MTZ enc
oding
from pulp import *

def mtz_encoding_tsp(n, cost_matrix):
    assert len(cost_matrix) == n, f'Cost matrix is no
t {n}x{n}'
    assert all(len(cj) == n for cj in cost_matrix),
f'Cost matrix is not {n}x{n}'
    # create our encoding variables
    binary_vars = [ # add a binary variable x_{ij} if
i not = j else simply add None
        [ LpVariable(f'x_{i}_{j}', cat='Binary') if i
!= j else None for j in range(n)]
        for i in range(n) ]
    # add time stamps for ranges 1 .. n (skip vertex
0 for timestamps)
    time_stamps = [LpVariable(f't_{j}', lowBound=0, u
pBound=n, cat='Continuous') for j in range(1, n)]
    # create the problem
    prob = LpProblem('TSP-MTZ', LpMinimize)
    # create add the objective function
    objective_function = lpSum( [ lpSum([xij*cj if xi
j != None else 0 for (xij, cj) in zip(brow, crow) ])
```

```
                                for (brow, crow) in zip(bi
nary_vars, cost_matrix)] )

    prob += objective_function

    # add the degree constraints
    for i in range(n):
        # Exactly one leaving variable
        prob += lpSum([xj for xj in binary_vars[i] if
xj != None]) == 1
        # Exactly one entering
        prob += lpSum([binary_vars[j][i] for j in ran
ge(n) if j != i]) == 1
    # add time stamp constraints
    for i in range(1,n):
        for j in range(1, n):
            if i == j:
                continue
            xij = binary_vars[i][j]
            ti = time_stamps[i-1]
            tj = time_stamps[j -1]
            prob += tj >= ti + xij - (1-xij)*(n+1) #
add the constraint
    # Done: solve the problem
    status = prob.solve(PULP_CBC_CMD(msg=False)) # tu
rn off messages
    assert status == constants.LpStatusOptimal, f'Une
xpected non-optimal status {status}'
    # Extract the tour
    tour = [0]
    tour_cost = 0
    while len(tour) < n:
        i = tour[-1]
        # find all indices j such that x_ij >= 0.999
        sols = [j for (j, xij) in enumerate(binary_va
rs[i]) if xij != None and xij.varValue >= 0.999]
        assert len(sols) == 1, f'{sols}' # there bett
er be just one such vertex or something has gone quit
e wrong
        j = sols[0] # extract the lone solutio
        tour_cost = tour_cost + cost_matrix[i][j] # a
dd to the tour cost
        tour.append(j) # append to the tour
        assert j != 0
    i = tour[-1]
    tour_cost = tour_cost + cost_matrix[i][0]
    return tour, tour_cost
```

In [15]:

Grade cell: `cell-43ec31eff68e08e6` Score: 11.67 / 11.67 (Top)

```
#test that exact answer is 10^6 times smaller than a
pproximate answer.
# compute MST based approximation
tour, tour_cost = minimum_spanning_tree_tsp(5, cost_
matrix)
print(f'MST approximation yields tour is {tour} with
cost {tour_cost}')
# compute exact answer
opt_tour, opt_tour_cost = mtz_encoding_tsp(5, cost_m
atrix)
print(f'Optimal tour is {opt_tour} with cost {opt_to
ur_cost}')
# check that the fraction is 1million times apart.
assert tour_cost/opt_tour_cost >= 1E+06, 'The TSP +
shortcutting tour must be at least 10^6 times costli
er than optimum. In your case, the ratio is {tour_co
st/opt_tour_cost}'
print('Test passed: 7 points')
```

```
Congratulations! All test cases in this ce
ll passed.
```

Problem 3

In this problem, we wish to solve TSP with additional constraints. Suppose we are given a TSP instance in the form of a $n \times n$ matrix C representing a complete graph.

We wish to solve a TSP but with additional constraints specified as a list $[(i_0, j_0), \ldots, (i_k, j_k)]$ wherein each pair (i_l, j_l) in the list specifies that vertex i_l must be visited in the tour before vertex j_l. Assume that the tour starts/ends at vertex 0 and none of the vertices in the constraint list is 0. I.e, $i_l \neq 0, j_l \neq 0$ for all $0 \leq l \leq k$.

Modify one of the ILP encodings we have presented to solve TSP with extra constraints. Implement your solution in the function `tsp_with_extra_constraints(n, cost_matrix, constr_list)` where the extra argument `constr_list` is a list of pairs `[(i0,j0),...., (ik, jk)]` that specify for each pair `(il,jl)` that vertex `il` must be visited before `jl`. Assume that the problem is feasible (no need to handle infeasible instances). Your code should output the optimal tour as a list.

Example

Consider again the graph with 5 nodes and the following cost matrix from problem 1:

Vertices	0	1	2	3	4
0	–	3	4	3	5
1	1	–	2	4	1

2	2	1	–	5	4
3	1	1	5	–	4
4	2	1	3	5	–

The optimal TSP tour will be $[0,3,1,4,2]$ with total cost 10.

Suppose we added the constraints $[(4,3),(1,2)]$ we note that the tour satisfies the constraint $(1,2)$ since it visits vertex 1 before vertex 2 but it unfortunately, $(4,3)$ is violated since vetex 3 is visited before 4 in the tour.

In [16]:

Student's answer (Top)

```
def tsp_with_extra_constraints(n, cost_matrix, const
raints):
    assert len(cost_matrix) == n, f'Cost matrix is n
ot {n}x{n}'
    assert all(len(cj) == n for cj in cost_matrix),
f'Cost matrix is not {n}x{n}'
    assert all( 1 <= i < n and 1 <= j < n and i != j
for (i,j) in constraints)
    # TODO: encode the problem in pulp (a) decision
variables; (b) constraints; (c) objective; (d) solve
and extract
    #       solution. This is going to be very close
to the MTZ encoding that we have presented in our no
tes. You can use
    #       our code as a starting point.
    # your code here
    binary_vars = [ # add a binary variable x_{ij} i
f i not = j else simply add None
        [ LpVariable(f'x_{i}_{j}', cat='Binary') if
i != j else None for j in range(n)]
        for i in range(n) ]
    # add time stamps for ranges 1 .. n (skip vertex
0 for timestamps)
    time_stamps = [LpVariable(f't_{j}', lowBound=0,
upBound=n, cat='Continuous') for j in range(1, n)]
    # create the problem
    prob = LpProblem('TSP-MTZ', LpMinimize)
    # create add the objective function
    objective_function = lpSum( [ lpSum([xij*cj if x
ij != None else 0 for (xij, cj) in zip(brow, crow)
])
                                  for (brow, crow) in zip(b
inary_vars, cost_matrix)] )

    prob += objective_function

    # add the degree constraints
    for i in range(n):
        # Exactly one leaving variable
        prob += lpSum([xj for xj in binary_vars[i] i
f xj != None]) == 1
        # Exactly one entering
        prob += lpSum([binary_vars[j][i] for j in ra
nge(n) if j != i]) == 1
    # add time stamp constraints
    for i in range(1,n):
        for j in range(1, n):
            if i == j:
                continue
            xij = binary_vars[i][j]
            ti = time_stamps[i-1]
            tj = time_stamps[j-1]
            prob += tj >= ti + xij - (1-xij)*(n+1) #
add the constraint
    for i1, j1 in constraints:
        prob += time_stamps[i1-1] <= time_stamps[j1-
1]

    # Done: solve the problem
    status = prob.solve(PULP_CBC_CMD(msg=False)) # t
urn off messages
    assert status == constants.LpStatusOptimal, f'Un
expected non-optimal status {status}'
    # Extract the tour
    tour = [0]
    tour_cost = 0
    while len(tour) < n:
        i = tour[-1]
        # find all indices j such that x_ij >= 0.999
        sols = [j for (j, xij) in enumerate(binary_v
ars[i]) if xij != None and xij.varValue >= 0.999]
        assert len(sols) == 1, f'{sols}' # there bet
ter be just one such vertex or something has gone qu
ite wrong
        j = sols[0] # extract the lone solutio
        tour_cost = tour_cost + cost_matrix[i][j] #
add to the tour cost
        tour.append(j) # append to the tour
        assert j != 0
    i = tour[-1]
    tour_cost = tour_cost + cost_matrix[i][0]
    return tour
```

In [17]:

Grade cell: `cell-6d36c6539b102ad2` Score: 5.0 / 5.0 (Top)

```
cost_matrix=[ [None,3,4,3,5],
              [1, None, 2,4, 1],
              [2, 1, None, 5, 4],
```

```
            [1, 1, 5, None, 4],
            [2, 1, 3, 5, None] ]
n=5
constraints = [(3,4),(1,2)]
tour = tsp_with_extra_constraints(n, cost_matrix, co
nstraints)
i = 0
tour_cost = 0
for j in tour[1:]:
    tour_cost += cost_matrix[i][j]
    i = j
tour_cost += cost_matrix[i][0]
print(f'Tour:{tour}')
print(f'Cost of your tour: {tour_cost}')
assert abs(tour_cost-10) <= 0.001, 'Expected cost wa
s 10'
for i in range(n):
    num = sum([1 if j == i else 0 for j in tour])
    assert  num == 1, f'Vertex {i} repeats {num} tim
es in tour'
for (i, j) in constraints:
    assert tour.index(i) < tour.index(j), f'Tour doe
s not respect constraint {(i,j)}'
print('Test Passed (3 points)')
```

```
Congratulations! All test cases in this ce
ll passed.
```

In [18]:

Grade cell: `cell-6efd9b8c3a4a7996` Score: 3.33 / 3.33 (Top)

```
cost_matrix=[ [None,3,4,3,5],
            [1, None, 2,4, 1],
            [2, 1, None, 5, 4],
            [1, 1, 5, None, 4],
            [2, 1, 3, 5, None] ]
n=5
constraints = [(4,3),(1,2)]
tour = tsp_with_extra_constraints(n, cost_matrix, co
nstraints)
i = 0
tour_cost = 0
for j in tour[1:]:
    tour_cost += cost_matrix[i][j]
    i = j
tour_cost += cost_matrix[i][0]
print(f'Tour:{tour}')
print(f'Cost of your tour: {tour_cost}')
assert abs(tour_cost-13) <= 0.001, 'Expected cost wa
s 13'
for i in range(n):
    num = sum([1 if j == i else 0 for j in tour])
    assert  num == 1, f'Vertex {i} repeats {num} tim
es in tour'
for (i, j) in constraints:
    assert tour.index(i) < tour.index(j), f'Tour doe
s not respect constraint {(i,j)}'
print('Test Passed (3 points)')
```

```
Congratulations! All test cases in this ce
ll passed.
```

In [19]:

Grade cell: `cell-474fa5a6876c7de0` Score: 16.67 / 16.67 (Top)

```
from random import uniform, randint

def create_cost(n):
    return [ [uniform(0, 5) if i != j else None for
j in range(n)] for i in range(n)]

for trial in range(20):
    print(f'Trial # {trial}')
    n = randint(6, 11)
    cost_matrix = create_cost(n)
    constraints = [(1, 3), (4, 2), (n-1, 1), (n-2,
2)]
    tour = tsp_with_extra_constraints(n, cost_matri
x, constraints)
    i = 0
    tour_cost = 0
    for j in tour[1:]:
        tour_cost += cost_matrix[i][j]
        i = j
    tour_cost += cost_matrix[i][0]
    print(f'Tour:{tour}')
    print(f'Cost of your tour: {tour_cost}')
    for i in range(n):
        num = sum([1 if j == i else 0 for j in tou
r])
        assert  num == 1, f'Vertex {i} repeats {num}
times in tour'
    for (i, j) in constraints:
        assert tour.index(i) < tour.index(j), f'Tour
does not respect constraint {(i,j)}'
print('Test Passed (10 points)')
```

```
Congratulations! All test cases in this ce
ll passed.
```

Answers to Select Problems

1A part A

```
Tour:[0, 5, 1, 4, 2, 3]
Cost of your tour: 12.071881535757976
Trial # 4
Tour:[0, 5, 1, 4, 3, 2]
Cost of your tour: 10.338286469095433
```

1A part A

- Vertex 0: k edges leave and k edges enter.
- Vertex 1, ..., n-1: 1 edge leaves and 1 edge enters (same as TSP).

1A part B

This is a trick question. There is no need to change any of the time stamp related constraints.