CS69001 Computing Laboratory – I

Assignment No: D1

Date: 28–August–2019

Computational (usually optimization) problems are sometimes formulated as linear-programming (LP) problems. In many situations, the LP instances lead to near-optimal solutions of the original problem. Even when the problem is known to be difficult (like NP-complete), the resulting approximation algorithms are often of good quality. In this assignment, you use the GNU Linear Programming Kit (GLPK) API in order to solve the traveling salesperson problem (TSP) sub-optimally and also optimally.

There are n sites (sometimes referred to as cities) in the two-dimensional plane. These sites are provided by their x and y coordinates. The distance between two sites at (x,y) and (x',y') is the usual Euclidean distance $\sqrt{(x-x')^2+(y-y')^2}$. The salesperson starts from one site (like the first one), visits each site once and only once, and eventually comes back to the site from where the tour began. The objective is to minimize the total distance covered by the salesperson.

The input graph of n sites (vertices) is a complete undirected graph with the edges carrying the inter-site distances as costs. The problem is to find a Hamiltonian cycle in this graph, which has the least cost. The number of such cycles is $(n-1)!$ which is exponential in n. Consequently, an exhaustive search is too slow for all practical purposes. We are happy if we obtain near-optimal solutions.

Part 1

Get yourself introduced to the GLPK API. This is installed in the lab machines. If you want to install it in your personal machine (like laptop), you need to download the package from the standard repositories. For Ubuntu, this can be done as follows.

```
$ sudo apt install glpk-utils libglpk-dev glpk-doc
```

Read the user's manual linked from the lab website. Include the following directive in your program.

```
#include <glpk.h>
```

Compile your code with the following flags.

```
gcc -Wall mytsp.c -lglpk -lm
```

GLPK supports both real-valued linear programming and mixed-integer optimization. The basic API calls are `glp_simplex` and `glp_intopt`. The integer optimizer needs an initial solution. You can start with the output of the simplex solver.

Part 2

In this part, you compute a sub-optimal solution of the TSP. The formulation of the problem is presented first. We number the sites as $1,2,3,\ldots,n$. For i,j in this range, define a Boolean variable x_{ij} with the meaning that if the salesperson's tour visits Site j immediately after Site i, then $x_{ij}=1$. Otherwise, $x_{ij}=0$. The objective function to be minimized is

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij},$$

where d_{ij} is the distance (assumed symmetric) between Site i and Site j. Since the salesperson enters and leaves each site only once, we have the following sets of constrains.

$$\sum_{i=1}^{n} x_{ij} = 1 \text{ for all } j=1,2,3,\ldots,n. \qquad \text{[}\textit{Entering constraints}\text{]}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \text{ for all } i=1,2,3,\ldots,n. \qquad \text{[}\textit{Leaving constraints}\text{]}$$

Unfortunately, these constraints alone cannot specify Hamiltonian cycles fully. Collections of node-disjoint cycles also qualify as feasible solutions. Even 2-cycles like $i \to j \to i$ can be included. Add the constraints

$$x_{ij} + x_{ji} \leqslant 1 \qquad \text{[No 2-cycle constraints]}$$

for all $i, j \in \{1,2,3,\ldots,n\}$, $i \neq j$, to avoid the 2-cycles. Likewise, 3-cycles, 4-cycles, 5-cycles, ... can be eliminated by adding more constraints, but the number of constraints will grow exponentially in n.

Solve the LP with only the entering, leaving, and no 2-cycle constraints. If the solution is a Hamiltonian cycle, we are done. Otherwise, you have a collection of node-disjoint cycles. Identify all these cycles. We need to stitch the different cycles together again by solving LP instances. If $i_1 \to i_2 \to \ldots \to i_l \to i_1$ is such a cycle of length l, add the new constraint

$$x_{i_1 i_2} + x_{i_2 i_3} + \cdots + x_{i_{l-1} i_l} + x_{i_l i_1} = l - 1 \qquad \text{[Cycle-breaking constraints]}$$

in order to forcibly remove one edge from the cycle. When all the cycles of the current solution are so handled, re-solve the LP with these additional constraints. It will produce the desired stitching effect, but may still give a solution with multiple cycles (longer than those from the previous attempt though).

Repeat until you get a solution with only one cycle. This solution is not guaranteed to be optimal, but is expected to be close to optimal. Most importantly, this method runs quite fast, despite multiple invocations of the LP solver. This method can handle, in feasible time, instances with n as large as 100.

Write a function **`tspsolve1`** to implement this algorithm.

Part 3

In this part, you write a function **`tspsolve2`** which produces an optimal solution. On the flip side, this algorithm is noticeably slower than the method of Part 2. Indeed, it is very problematic with this method to handle n somewhat larger than 20.

We again start with the entering and leaving constraints of Part 2, but we do not include the no 2-cycle constraints $x_{ij} + x_{ji} \leqslant 1$. Instead we preclude solutions with multiple cycles by introducing n integer-valued variables $u_1, u_2, u_3, \ldots, u_n \in \{1,2,3,\ldots,n\}$, where u_i stands for the serial number of visiting Site i. We assume that the salesperson's tour starts from Site 1, so $u_1 = 1$. Later, if $x_{ij} = 1$ (that is, Site j is visited immediately after Site i), we have $u_j = u_i + 1$, This implies that $u_2, u_3, u_4, \ldots, u_n$ is a permutation of $2,3,4,\ldots,n$. Since $u_1 = 1$ always, we may choose not to introduce this variable.

These new variables are now connected with the earlier variables by the constraints

$$u_i - u_j + n x_{ij} \leqslant n - 1 \text{ for all } i, j \in \{2,3,4,\ldots,n\},\ i \neq j. \qquad \text{[Hamiltonian-cycle constraints]}$$

These constraints ensure that there cannot be a solution with multiple cycles. If such a solution is feasible, there is a cycle not containing Site 1. Let l be the length of this cycle. Adding the constraints over the edges of this cycle gives $nl \leqslant (n-1)l$ because the u variables cancel each other. This implies $n \leqslant n-1$, a contradiction. Conversely, we show that these constraints permit Hamiltonian cycles. If $x_{ij} = 1$, we have $u_j = u_i + 1$, so $u_i - u_j + n x_{ij} = n - 1$. If $x_{ij} = 0$, then $u_i - u_j + n x_{ij} = u_i - u_j$ is evidently $\leqslant n - 1$.

Solve the LP instance consisting of the entering, leaving and Hamiltonian-cycle constraints. The objective function remains the same, and depends only on the variables x_{ij}.

The *main*() function

- The reader supplies n (the number of sites), and the coordinate pairs of the n sites.
- Call **`tspsolve1`**. Report the iterative refining of the solutions in the multiple LP instances.
- Call **`tspsolve2`**, and report the solution obtained.

Submit a single C/C++ source file. Do not use global/static variables.

Sample output

```
n = 16
0.3720751099158428  0.8854997478823642
0.4418202128456069  0.6563730121852704
0.3281098629013215  0.8965254909808400
0.5462351159873582  0.6615548001888929
0.5338516042259762  0.7480234912354609
0.3681871799603976  0.8455706624526393
0.2843932724950804  0.9162659924087422
0.7432693106789464  0.0859196135243027
0.6055622690383169  0.0693114879863856
0.1667971541950466  0.6799476238339895
0.2285322212746982  0.5114901072864840
0.4067291055837316  0.2238342302031043
0.2574509313597581  0.3256484103974180
0.8716194615101532  0.0972275515539700
0.0743328654553429  0.7439751414321247
0.5152220761008663  0.4464079753711857

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++ Solving TSP by Method 1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+++ Attempt No 1

+++ Minimum simplex cost = 2.3837046899265752
+++ Minimum mip cost     = 2.3837046899265752

1  -> 3  -> 7  -> 6  -> 1
2  -> 4  -> 5  -> 2
8  -> 9  -> 14 -> 8
10 -> 11 -> 15 -> 10
12 -> 13 -> 16 -> 12

Solution of Attempt 1 has 5 cycles

+++ Attempt No 2

+++ Minimum simplex cost = 2.3837046899265752
+++ Minimum mip cost     = 2.8226231520429894

1  -> 3  -> 7  -> 15 -> 10 -> 11 -> 2  -> 4  -> 5  -> 6  -> 1
8  -> 9  -> 12 -> 13 -> 16 -> 14 -> 8

Solution of Attempt 2 has 2 cycles

+++ Attempt No 3

+++ Minimum simplex cost = 2.6370442856125105
+++ Minimum mip cost     = 2.9527525149127167

1  -> 3  -> 7  -> 15 -> 10 -> 11 -> 16 -> 14 -> 8  -> 9  -> 12 -> 13 -> 2  -> 4  -> 5  -> 6  -> 1

Solution of Attempt 3 has 1 cycles

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++ Solving TSP by Method 2
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+++ Minimum simplex cost = 2.1543997685660052
+++ Minimum mip cost     = 2.6906287309383856

1  -> 6  -> 5  -> 4  -> 2  -> 16 -> 14 -> 8  -> 9  -> 12 -> 13 -> 11 -> 10 -> 15 -> 7  -> 3  -> 1

Solution has 1 cycles
```