CS4234
Optimiz(s)ation Algorithms

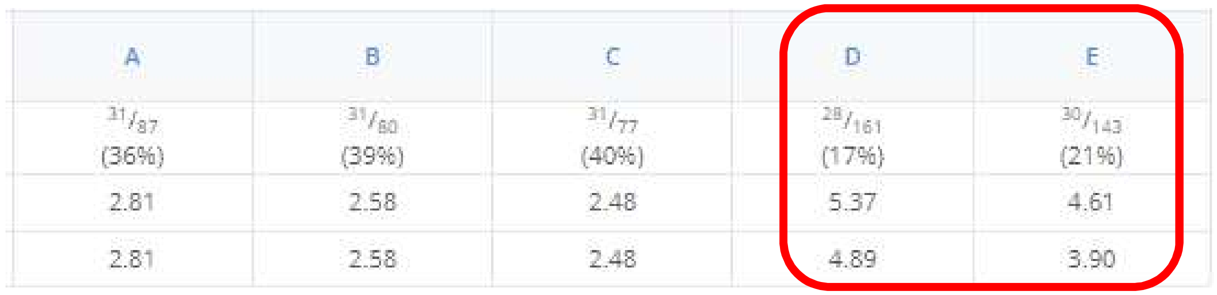
British: Travelling
American: Traveling

L4 – `Travelling-Salesman-Problem (TSP)`

We don't say TSP ~~Problem~~

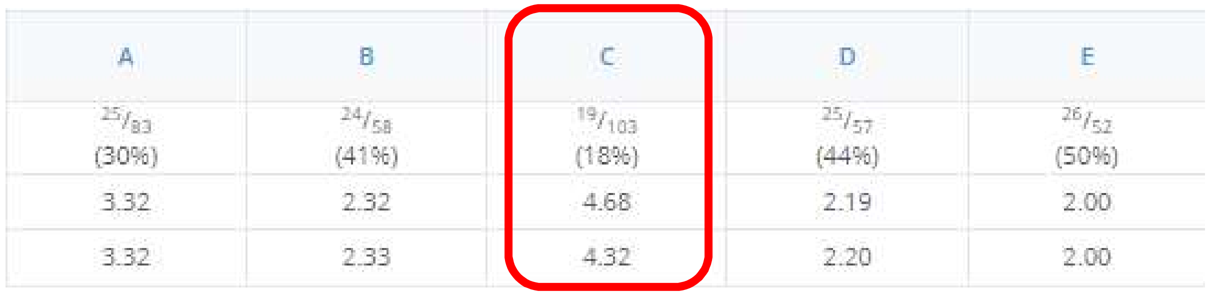
Admins (1)

- PS1 final average = 4.92/5.00 for 31 of you :O



A	B	C	D	E
31/87 (36%)	31/80 (39%)	31/77 (40%)	28/161 (17%)	30/143 (21%)
2.81	2.58	2.48	5.37	4.61
2.81	2.58	2.48	4.89	3.90

- PS2 grading at 18/31 (as of Wed, 01 Sep 2021, 3pm)…
  - That's the number of you who have AC*-ed all 5 tasks…
    - With help of online judge, many of you are "self-correcting" your own issues (at the expense of your own personal time… remember $^3C_2$ of college students?), thereby simplifying the grading only for the non-AC submissions at the end
    - PS: You are *NOT* technically allowed to 'try' at open.kattis first (it skews the stats)



A	B	C	D	E
25/83 (30%)	24/58 (41%)	19/103 (18%)	25/57 (44%)	26/52 (50%)
3.32	2.32	4.68	2.19	2.00
3.32	2.33	4.32	2.20	2.00

Admins (2)

- Good? News: PS3+4 workload will be made "lighter" and *possibly* over "longer period"
  - PS2 pace at the wrong end of the ranklist is slower than expected
  - PS3 will be at 4 tasks
    - And if need be, I can set its deadline to be 'sometime in the middle of recess week'
    - At the expense of not having `recess week break' (for those who struggled to finish PS3)
    - I prefer to clear all PS3 grading during this recess week though…
  - PS4 will also be at 4 tasks
    - And if need be, I can set its start time to be `in the middle of recess week after PS3 ends' too
    - At the expense of `spending a few hours during recess week break' (for those who aim for top 7 in PS4)

Before I forget

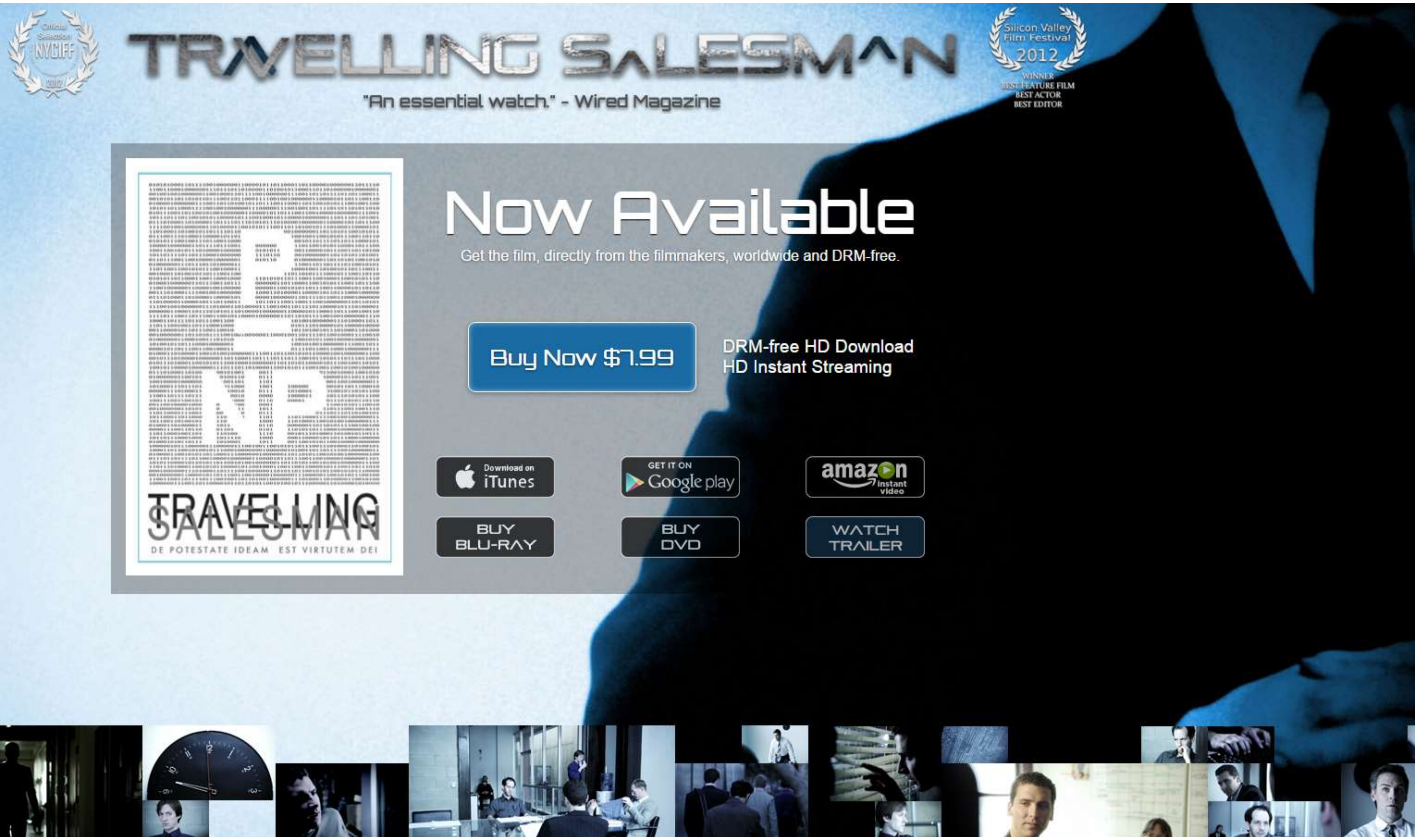
Time to do attendance taking ☺

- https://inetapps.nus.edu.sg/ctr/Home

You need to:

- ~~Declare your temperature regularly~~ (no longer needed)
- Declare your (+family member(s)) health regularly
- Do your FET/ART test if you are scheduled for it
- Should *not* be on SHN/LOA at the moment

One of the most famous COP


TRAVELLING SALESMAN

"An essential watch." - Wired Magazine

Now Available

Get the film, directly from the filmmakers, worldwide and DRM-free.

Buy Now $7.99

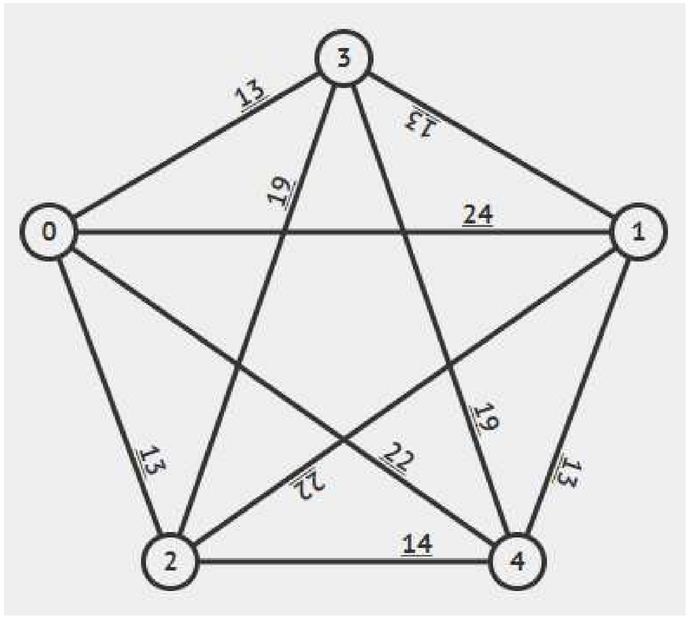
DRM-free HD Download
HD Instant Streaming

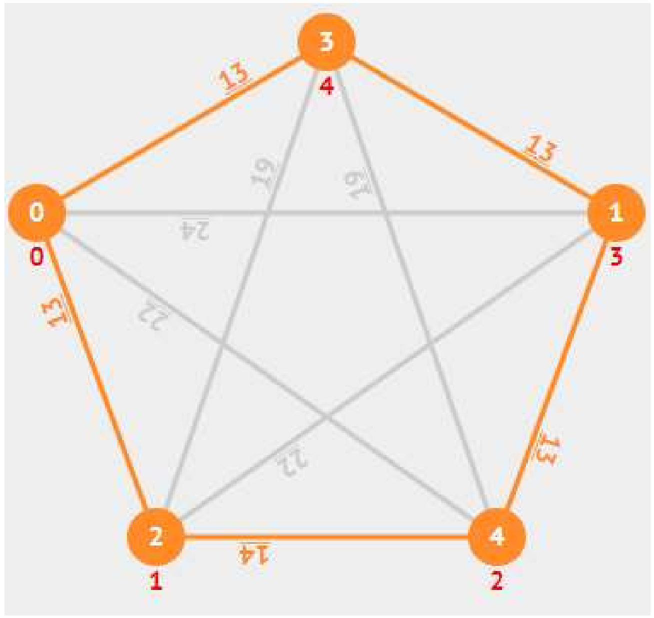
Travelling-Salesman-Problem (TSP)

Given a set of cities (i.e., points, or vertices), the goal of the TSP is to find a minimum cost circuit (or cycle, or tour) that visits all the points. More formally, the problem is stated as follows:

Definition 1 *Given a set V of n points and a distance function $d : V \times V \to \mathbb{R}$, find a cycle C of minimum cost that contains all the points in V. The cost of a cycle $C = (e_1, e_2, \ldots, e_n)$ is defined to be $\sum_{e \in C} d(e)$, and we assume that the distance function is non-negative (i.e., $d(x, y) \geq 0$).*

Complete Graph, $O(V^2)$





Digress a bit to Brute Force and DP

There are N! permutations of N cities

- Or just (N-1)! Permutations if we “fix one city”
  - So, this runs in O(N! * N) time as we run O(N) check per permutation

But many of the sub-tours are repeated

- If we memoize (this word is not a typo) parts of the subtours, we can improve runtime to $O(N^2 * 2^{N-1})$
  - Called Held-Karp DP for TSP
  - A few mini optimizations exist, including “fix one city” (not for CS4234)
  - A short demo (not live code) https://nus.kattis.com/problems/beepers
    - https://github.com/stevenhalim/cpbook-code/blob/master/ch3/dp/beepers_UVa10496.cpp / py / java / ml

Let’s analyze the animations: https://visualgo.net/en/tsp

4 Variants (so far), all NP-hard

The one in VisuAlgo*

	Repeats	No-Repeats
Metric	M-R-TSP	M-NR-TSP
General	G-R-TSP	G-NR-TSP

M-Metric

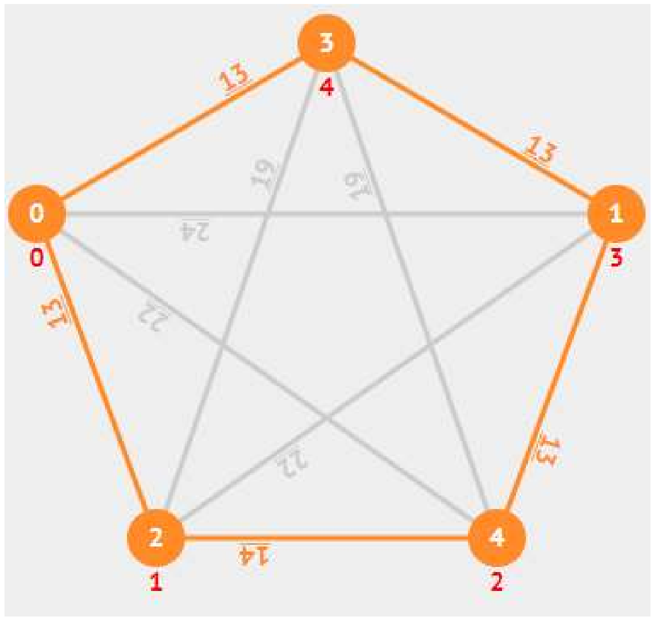
- That ∆ inequality

G-General

R-Repeated-Visits-OK

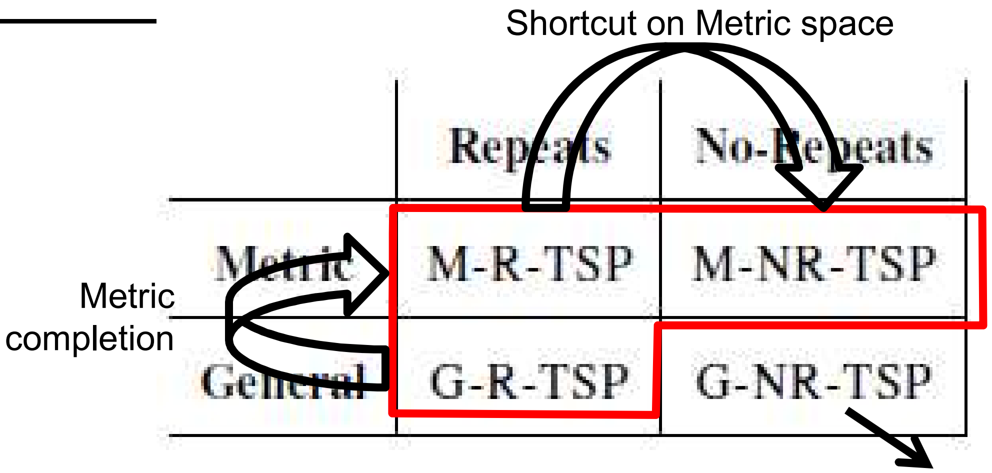
NR-No-Repeat

- Easy reduction from an NP-c `Hamiltonian-Cycle`
  - A Hamiltonian cycle (circuit) is a cycle in an undirected or directed graph that visits each vertex exactly once



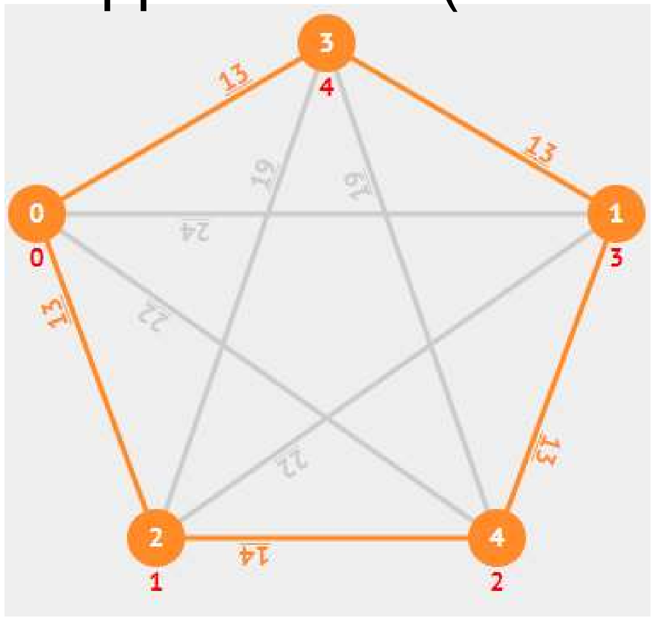
But 3 (out of 4) variants are equivalent

If we have a **c**-approximation algorithm for one variant, we can construct a **c**-approximation algorithm for the other two


Shortcut on Metric space
Metric completion

	Repeats	No-Repeats
Metric	M-R-TSP	M-NR-TSP
General	G-R-TSP	G-NR-TSP

Temporarily ignored, NP-hard even to approximate (see T03)



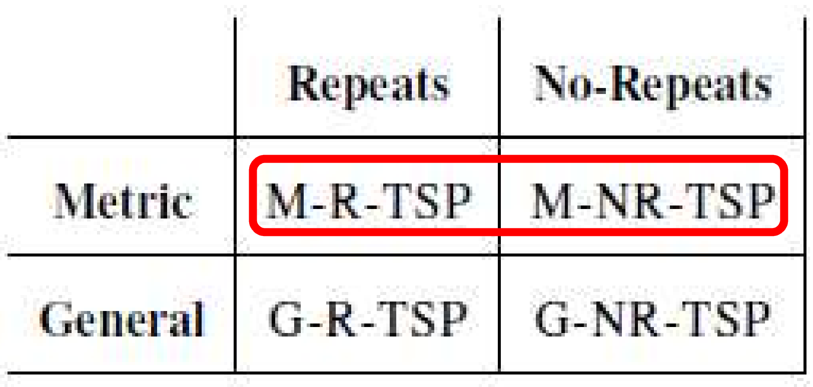
We will learn **2**-approx and **1.5**-approx for these 3 variants

Proof of Equivalency of the 3 variants

Are given in the next few slides and in the more detailed pdf, but are to be skipped in class

We focus on the approximation algorithms for M-R-TSP

M-R-TSP ↔ M-NR-TSP (1)



	Repeats	No-Repeats
Metric	M-R-TSP	M-NR-TSP
General	G-R-TSP	G-NR-TSP

Both are **M-Metric**

NR to **R** → trivial, no change

- A legal **NR** cycle is also legal in the **R** variant
- Recall ILP to LP 'relaxation' in the previous lecture
- So OPT(**R**) ≤ OPT(**NR**)

R to **NR** → we use that shortcut(s) technique

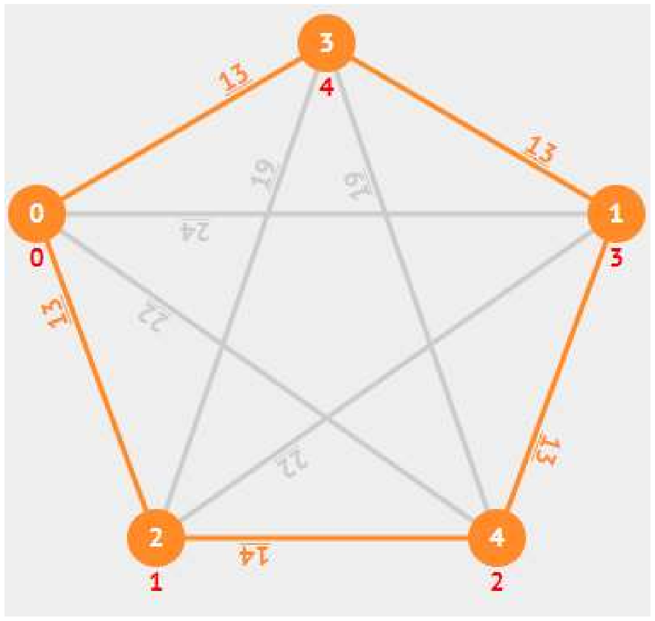
- Recall discussion during Steiner Tree lecture about using shortcut(s) in the metric version of Steiner Tree

M-R-TSP ↔ M-NR-TSP (2)

R to **NR** → we use that shortcut(s) technique

e.g*., C = {0,2,4,1,4,3(,0)}, C′ = {0,2,4,1,3(,0)}

- So OPT(**NR**) ≤ OPT(**R**) (via shortcuts, e.g., d(1, 3) ≤ d(1, 4) + d(4, 3) due to Δ **inequality**)
- As OPT(**R**) ≤ OPT(**NR**) (from previous slide) and OPT(**NR**) ≤ OPT(**R**) (above), we have OPT(**R**) = OPT(**NR**)
- If A that produces cycle C is a c-approx algo for **NR**, then A is a c-approx algo for **R** as d(C) ≤ c*OPT(**NR**) = c*OPT(**R**)
- If A that produces cycle C is a c-approx algo for **R**, then A' (that runs A and skips repeated vertices to produce cycle C') is a c-approx algo for **NR** as d(C) ≤ c*OPT(**R**) = c*OPT(**NR**)

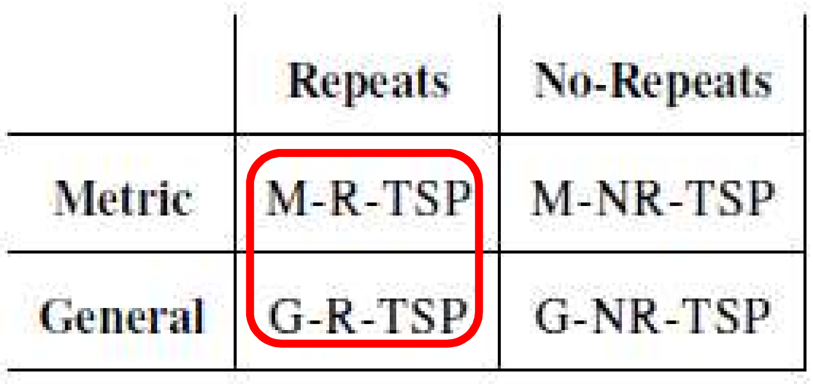


* This example is just for illustration

M-R-TSP ↔ G-R-TSP (1)

Both are **R-Repeated**

We cannot create shortcuts as Δ inequality does not hold, but we can use repeated vertices



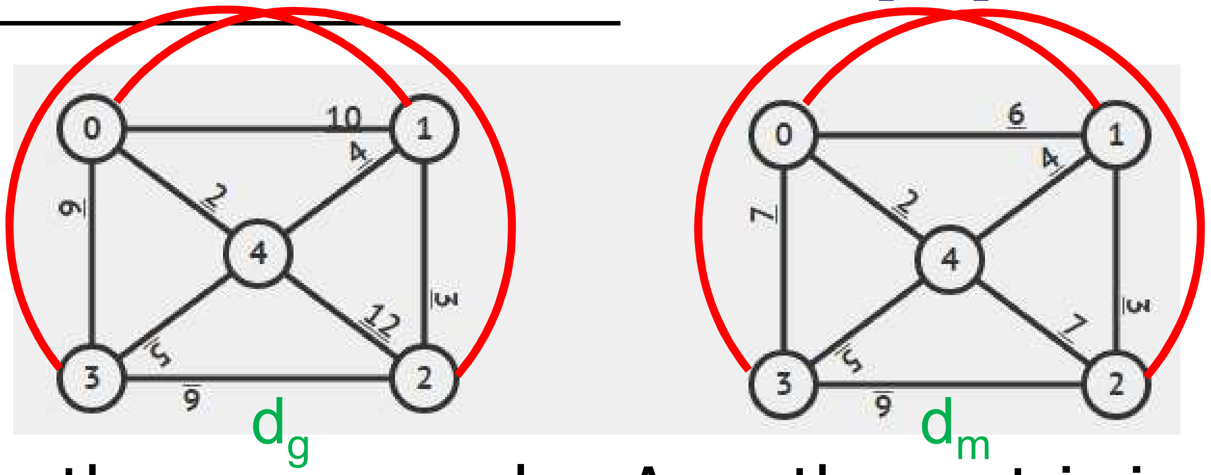
	Repeats	No-Repeats
Metric	M-R-TSP	M-NR-TSP
General	G-R-TSP	G-NR-TSP

G to **M** → trivial, if we have a c-approx algo A for G-R-TSP and the input is metric, no change is needed; we just run A on this metric input as this metric input is a valid input for the general one too, and we are guaranteed to get c-approx result ☺

M to **G** → if we have a c-approx algo for M-R-TSP, we have to do metric completion on input instance (V, d_g) using any All-Pairs-Shortest-Paths (APSP) algorithm like in the Steiner-Tree problem (previous lecture) to get (V, d_m), see the next slide

M-R-TSP ↔ G-R-TSP (2)

Assume the two red edges are ∞



Then, we run the c-approx algo A on the metric instance (V, d_m) to produce a cycle **C, e.g*., C = {0,4,1,2,3,0}**

Then, we undo the process by replacing edge (**u**, **v**) with shortest path from **u** to **v** in the original graph to produce cycle, **e.g*., C'** (may contain repeats – it is fine) = **{0,4,1,2,3,4,0}**

- d(**C'**) = d(**C**) (obvious, hopefully)
- d(C) ≤ c*OPT(V, d_m) (by definition of c-approx algo A)
- OPT(V, d_m) ≤ OPT(V, d_g) (all edge (u, v) in d_m is shortest paths, see PDF)
- So d(C') ≤ c*OPT(V, d_g)

* This example is just for illustration

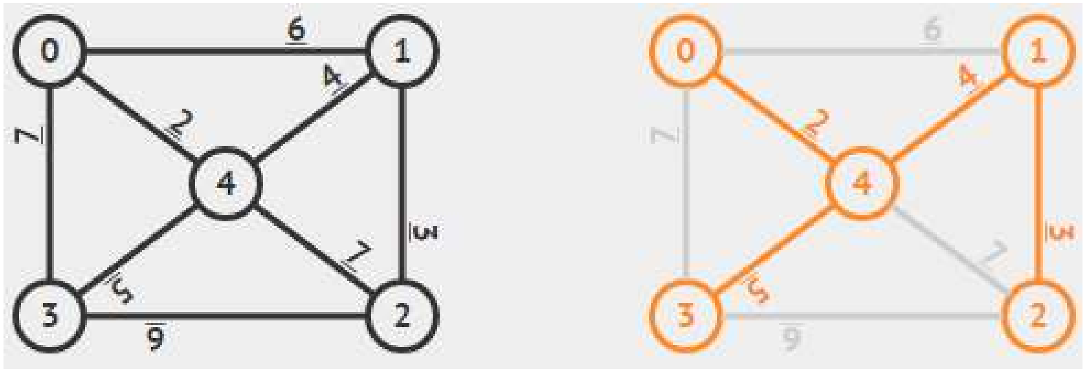
2-Approx for M-NR-TSP

Try this at https://visualgo.net/en/tsp (still not 100% clear though)

As we have shown equivalence between **M-R-TSP**, **M-NR-TSP**, and **G-R-TSP**, this 2-Approx algorithm described below will work for all three variants:

1. Run MST of the input Graph (we only accept metric in VA)
2. Run DFS on the resulting MST (usually from vertex 0)
3. Output the vertices in cycle induced by DFS (no repeat in VA)

Notice similarity with `Metric-Steiner-Tree` approximation



DFS={0,4,1,2,1,4,3,4,0}
Output C={0,4,1,2,3,0}

Assume two non-drawn edges 0-2 and 1-3 are ∞

2-Approx for `G-R-TSP` (Analysis)

We know M-R-TSP and M-NR-TSP are equivalent with G-R-TSP that will be analyzed now; In class, I use M-NR-TSP as VisuAlgo is limited to Metric* input and VisuAlgo avoids repeat vertices at the end

The analysis is similar to `Metric-Steiner-Tree` 2-Approx analysis

Let: **C* = OPT(V, d)** where **d** is not necessarily metric (hence the **G** in **G-R-TSP**) and **E*** be the edges in the optimal cycle **C***

Notice that **G* = (V, E*)** is connected as **C*** is a cycle that includes all vertices in **V**

Let **T*** be the MST of **G = (V, E*)**

We know that **d(T*)** $\leq$ **d(C*) = OPT**
(similar as with `Metric-Steiner-Tree` analysis)

2-Approx for G-R-TSP, Continued

As the tree **T** (the one used by the approximation algorithm) is also an MST, but on **G = (V, E)** (a complete graph) and we know that **E*** $\subseteq$ **E**, we have: **d(T)** $\leq$ **d(T*)**

Finally, since **C** is constructed by a DFS traversal of **T** and **C** includes each edge in **T** exactly twice (we used this in `Metric-Steiner-Tree` analysis too), we have **d(C) = 2 * d(T)**, notice that we haven't drop repeats (hence the **R** in **G-R-TSP**)

Combining everything, we have:

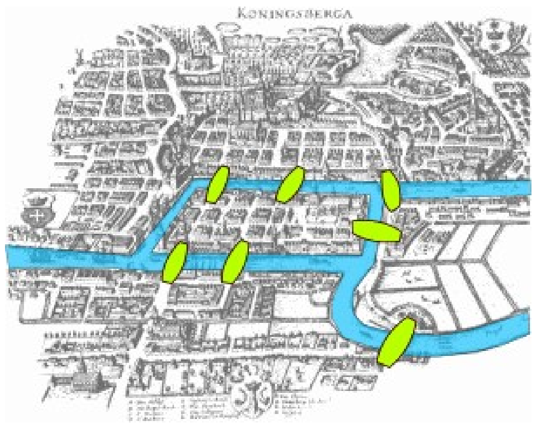
$$\begin{aligned} \mathbf{d(C)} &= \mathbf{2 * d(T)} \\ &\leq \mathbf{2 * d(T^*)} \\ &\leq \mathbf{2 * d(C^*)} \\ &\leq \mathbf{2 * OPT} \end{aligned}$$

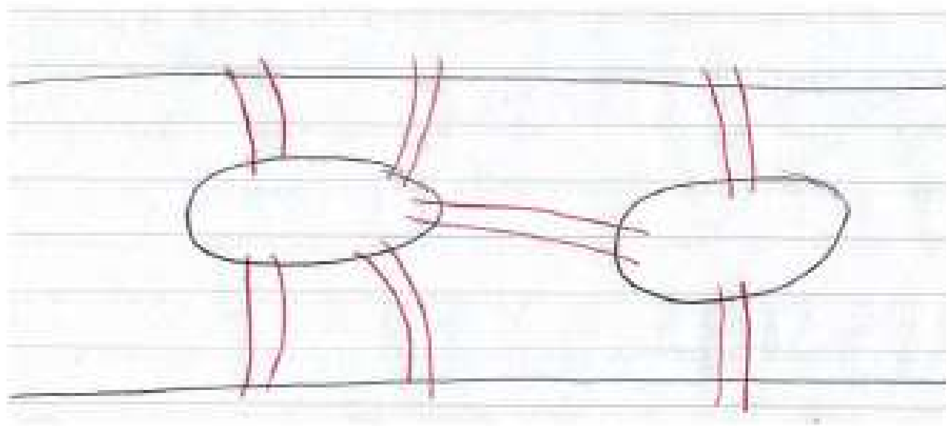
Btw, do you "feel" that this analysis is "not tight"?

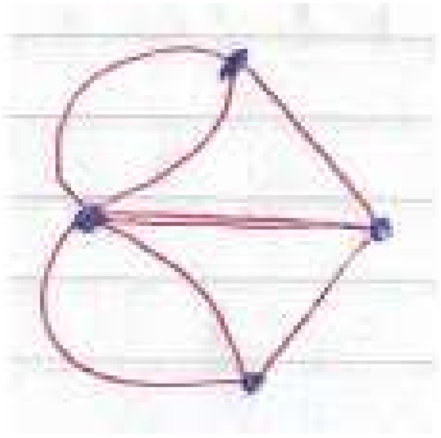
Can we do better? (1)

Preliminary 1: Eulerian Cycle:
a Cycle that crosses each edge exactly once

Illustration: Bridges of Konigsberg





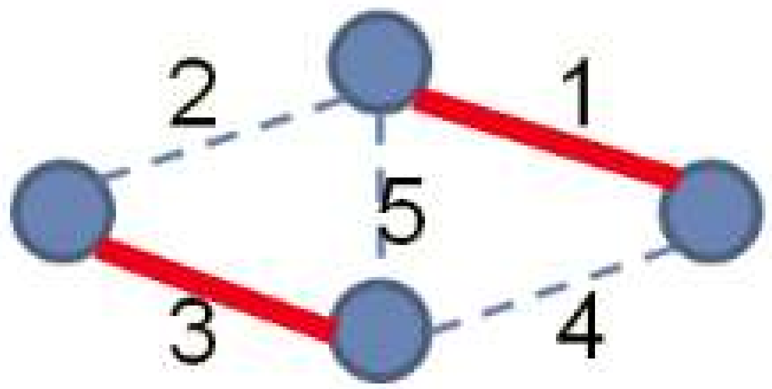


A (multi)graph **G = (V, E)** has an Eulerian Cycle iff it is connected (ignoring vertices with degree 0) and every vertex has even degree (details in PDF)

Can we do better? (2)

Preliminary 2: (Perfect) Matching
Get subset **M** of edges in graph so that no two edges in **M** share an endpoint; it is perfect if **|M| = |V|/2**

PS: How to get a perfect matching on **general** graph (is still) *out of scope* for this module (we can use weighted form of Edmonds' matching (Blossom) algorithm as a black box – Wk07)



1.5-Approx for `M-NR-TSP`

Christofides's Algorithm:

1. T = Min-Spanning-Tree(G) and let E be all the edges in T
2. Let O be set of vertices in T that has odd degree
   - O has even number of vertices (Handshaking lemma)
3. Find M = Min-Weight-Perfect-Matching on subgraph of G induced by O
4. Combine T+M to get a multigraph H whereby all vertices have even degree
5. Get the Eulerian circuit in H
6. Output the vertices in Eulerian cycle (no repeat in VA)

https://en.wikipedia.org/wiki/Christofides_algorithm#Example

Or see https://visualgo.net/en/tsp (the Eulerian circuit in H is a bit buggy; so use manual tracing)

1.5-Approx for M-NR-TSP (Analysis1)

The cost has two components

$$\text{Cost}(E) = \sum_{e \in E} d(e) + \sum_{e \in M} d(e)$$

(all edges in MST) + (all edges in Min-Weight-Perfect-Matching)

We already know that

$$\sum_{e \in E} d(e) \leq \text{OPT}$$

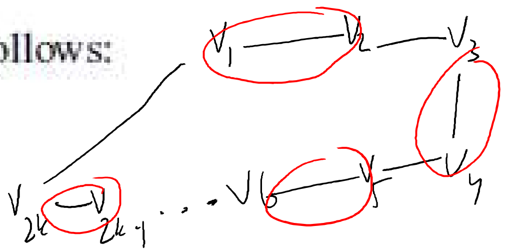
using the now-classic technique: OPT is a TSP cycle of graph G = (V, E), if we remove any edge from this cycle, we will get a spanning tree and the MST of the graph G must have cost no greater than this cycle (usually smaller, as we delete at least one positive weighted edge)

1.5-Approx for M−NR−TSP (Analysis2)

We now need to argue that $2 \times \sum_{e \in M} d(e) \leq OPT$. Let C be the cycle for OPT. Let C' be the same cycle as C where we skip all the vertices in $V - O$, and we skip repeats. That is, C' is a cycle on the odd vertices with no repeats. Notice that $cost(C') \leq cost(C)$, since we have only skipped vertices (and the triangle inequality holds).

Also, notice that cycle C' has an even number of vertices (because there are an even number of odd vertices) and an even number of edges. Assume that $C' = (v_1, v_2, \ldots, v_{2k})$.

We can now construct two different perfect matchings M_1 and M_2. We define them as follows:



$$\begin{aligned} M_1 &= (v_1, v_2), (v_3, v_4), (v_5, v_6), \ldots, (v_{2k-1}, v_{2k}) \\ M_2 &= (v_2, v_3), (v_4, v_5), (v_6, v_7), \ldots, (v_{2k}, v_1) \end{aligned}$$

Notice that each of these perfect matchings has $k = |O|/2$ edges, and both are valid perfect matchings for the set O. Since M is the minimum cost perfect matching, we conclude that:

$$\begin{aligned} cost(M) &\leq cost(M_1) \\ cost(M) &\leq cost(M_2) \end{aligned}$$

Notice, though, that $cost(M_1) + cost(M_2) = cost(C') \leq cost(C)$. So, if we sum the matchings, we get:

$$2 \times cost(M) \leq cost(M_1) + cost(M_2) \leq cost(C) = OPT$$

Thus we have shown that $cost(M) \leq OPT/2$.

1.5-Approx for M-NR-TSP (Analysis3)

Combining them, we have:

$$\begin{aligned} \text{Cost}(E) &= \textstyle\sum_{e \in E} d(e) + \sum_{e \in M} d(e) \\ &\leq \text{cost}(T) + \text{cost}(M) \\ &\leq \text{OPT} + \text{OPT}/2 \\ &\leq 1.5\ \text{OPT} \end{aligned}$$

Remember that the 1.5-Approx at https://visualgo.net/en/tsp has bug involving the Eulerian circuit in H… Use manual tracing for that 1.5-Approx animation until I have time to fix it…

Even better?

and in T03

We will revisit TSP again in the second half of the course…

List of NP-hard COPs so far…

(in order of appearance)

1. `Min-Vertex-Cover` (+weighted version, Lec1+Lec2)
2. `Max-Clique` (mentioned briefly and in Tut01)
3. `Graph-Coloring` (mentioned briefly in Tut01)
4. `Min-Set-Cover` (+weighted version, Lec3)
5. `Steiner-Tree` (3 variants, Lec3)
6. `Min-Feedback-Edge-Set` (+weighted version, Tut02)
7. `Partition` (+weighted version, Tut02)
8. `Travelling-Salesman-Problem` (4 variants, Lec4)
9. `Max-Independent-Set` (Tut03)

- A few more in latter tutorials and/or in PS3+PS4+Mini Project

Summary

- Revisit the `Travelling-Salesman-Problem`
  - One of the most popular COP
- Four variants: Metric and/or Repeats
  - All NP-hard, focus on 3, ignore G-NR-TSP for now
  - The other 3 variants are equivalent
- MST+DFS, 2-Approximation Algorithm
  - Simpler
- Christofides's 1.5-Approximation Algorithm
  - Eulerian Cycle, General Weighted Matching, Harder
- Next week: Back into P problem: Maximum Flow…