Algebra and Discrete Mathematics
ADM

Bc. Xiaolu Hou, PhD.

FIIT, STU
xiaolu.hou @ stuba.sk

Course Outline

- Vectors and matrices
- System of linear equations
- Matrix inverse and determinants
- Vector spaces and matrix transformations
- Fundamental spaces and decompositions
- Eulerian tours
- Hamiltonian cycles
- Midterm
- Paths and spanning trees
- Trees and networks
- Matching

Recommended reading

- Saoub, K. R. (2017). A tour through graph theory. Chapman and Hall/CRC.
  - Sections 2.1, 2.2, 2.3
  - Accessible online (free copy)
  - Alternative download link

Lecture outline

- Existence of Hamiltonian cycles
- Traveling salesman problem
- Digraphs and Asymmetric Traveling Salesman Problem

Hamiltonian cycles

- Existence of Hamiltonian cycles
- Traveling salesman problem
- Digraphs and Asymmetric Traveling Salesman Problem

Hamiltonian cycle and Hamiltonian path

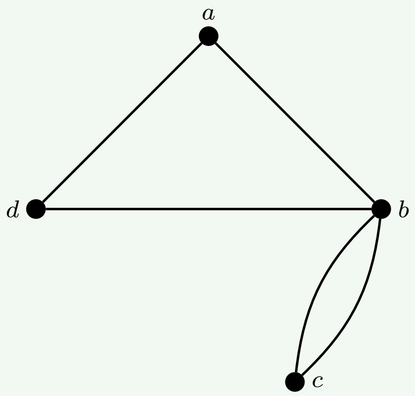
Definition

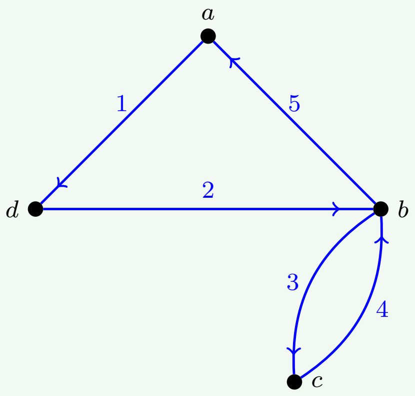
A cycle in a graph G that contains every vertex of G is called a *Hamiltoniain cycle*. A path that contains every vertex is called a *Hamiltonian path*.

- Recall that a cycle or a path can only pass through a vertex once, so the Hamiltonian cycles and paths travel through every vertex exactly once.
- Named after mathematician: Sir William Hamilton

Hamiltonian cycle and Eulerian circuit – example

Example

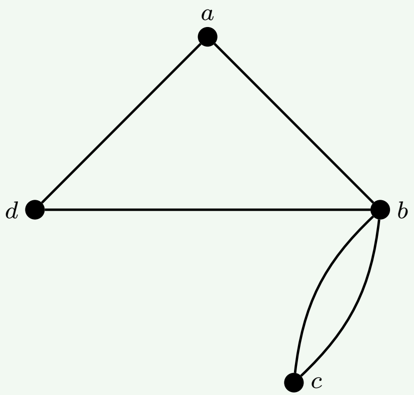


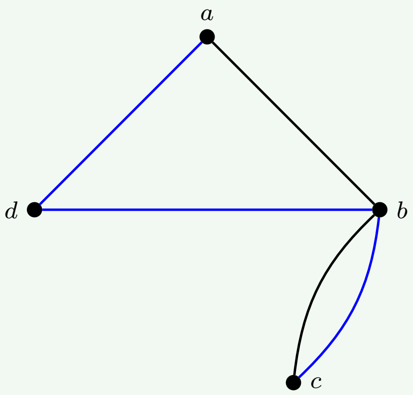


The graph is connected and all vertices are even – it contains an Eulerian circuit

Hamiltonian cycle and Eulerian circuit – example

Example

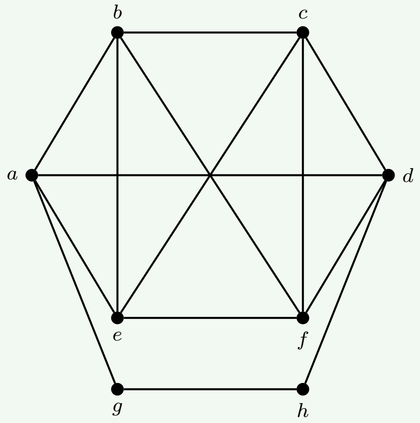


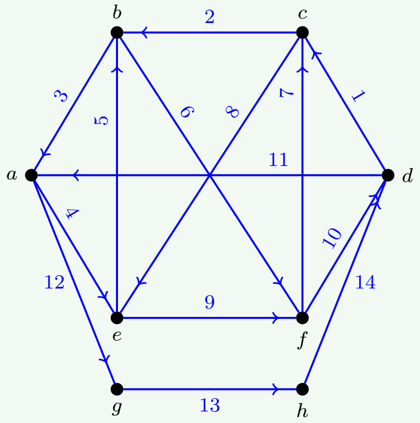


There is no Hamiltonian cycle since we need to include c in the cycle and by doing so we would pass b twice

Hamiltonian cycle and Eulerian circuit – example

Example

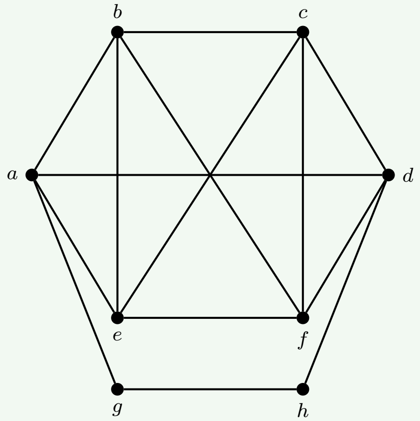


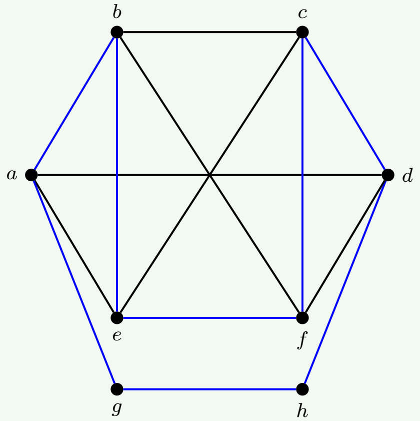


The graph is connected and all vertices are even – it contains an Eulerian circuit

Hamiltonian cycle and Eulerian circuit – example

Example





Hamiltonian cycles and Hamiltonian paths also exist. To find one such path, remove any one of the highlighted edges from the Hamiltonian cycle

Hamiltonian cycle and Hamiltonain path

- If a graph has an Eulerian circuit, then it cannot have an Eulerian trail, and vice versa
- This is not true for the Hamiltonian version
    - If a graph has a Hamiltonian cycle, it automatically has a Hamiltonian path (just leave off the last edge of the cycle to obtain a path).
    - If a graph has a Hamiltonian path, it may or may not have a Hamiltonian cycle.

Necessary conditions for Hamiltonian cycle

- G must be connected
- No vertex of G can have degree less than 2
- G cannot contain a cut-vertex

Cut-vertex: a vertex whose removal disconnects the graph

Note

These are not sufficient conditions

A sufficient condition for Hamiltonian cycle

Theorem (Dirac's Theorem)

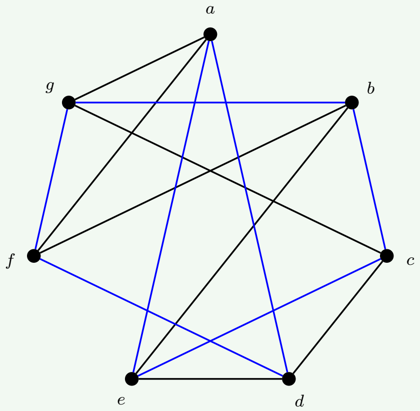
Let G be a graph with $n \geq 3$ vertices. If every vertex v of G satisfies $\deg(v) \geq \frac{n}{2}$, then G has a Hamiltonian cycle.

Remark

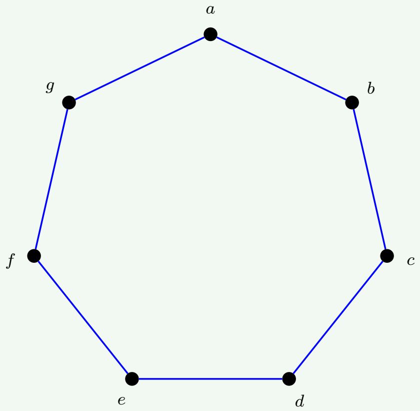
- The proof of this theorem boils down to the fact that each vertex has so many edges incident to it that in trying to find a cycle we will never get stuck at a vertex.
- There are a few sufficient conditions for a graph to have Hamiltonian cycle, but but no known necessary and sufficient condition that works for all graphs.

Existence of Hamiltonian cycle – example

Example



$n = 7$
$\deg = 4 \geq \frac{7}{2}$
for all vertices



$n = 7$
$\deg = 2 < \frac{7}{2}$
for all vertices

Complete graph

Definition

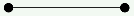
A simple graph G is *complete* if every pair of distinct vertices is adjacent. The complete graph on n vertices is denoted K_n.

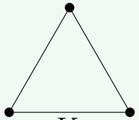
- If we think of an edge as describing a relationship between two objects, then a complete graph represents a scenario where every pair of vertices satisfies this relationship.

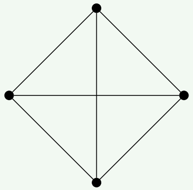
The first six complete graphs

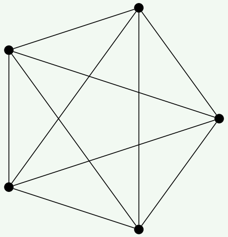
Example

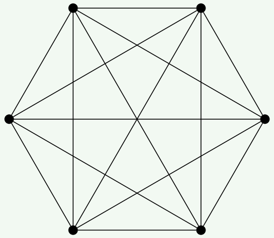
K_1


K_2


K_3


K_4


K_5


K_6

Properties of K_n

- Each vertex in K_n has degree $n-1$
- K_n has $\frac{n(n-1)}{2}$ edges
- K_n contains the most edges out of all simple graphs on n vertices

Note

- Any complete graph (on at least 3 vertices) satisfies the conditions of Dirac's Theorem, and therefore contains a Hamiltonian cycle

Number of Hamiltonian cycles in a complete graph

- When modeling a problem whose solution is a Hamiltonian cycle (or path) in the appropriate graph, there is often a desired "home" location or starting vertex – *reference point*

Theorem

Given a specified reference point, the complete graph K_n has $(n-1)!$ distinct Hamiltonian cycles. Half of these cycles are reversals of the others.

- When starting at the designated vertex there are $n-1$ possible edges to choose from
- Once that edge has been traveled to arrive at a new vertex, we cannot pick the edge just traveled and so there remain $n-2$ edges to choose from
- At the next vertex, we cannot travel back to either of the previously chosen vertices, and so there remains $n-3$ edges available
- Continuing in this manner, we have a total of

$$(n-1)(n-2)(n-3)\cdots(2)(1) = (n-1)!$$

Hamiltonian cycles

- Existence of Hamiltonian cycles
- Traveling salesman problem
- Digraphs and Asymmetric Traveling Salesman Problem

The question

- How should a delivery service plan its route through a city to ensure each customer is reached?
- A traveling salesman has customers in numerous cities. He must visit each of them and return home, but wishes to do this with the least total cost
- Shortest Hamiltonian cycle visiting all cities in one country
- Model: weighted complete graph
- Weights: different distances
- Complete: it is reasonable to assume it's possible to travel between any two cities

Brute Force Algorithm

- To find the Hamiltonian cycle of least total weight, one obvious method is to find all possible Hamiltonian cycles and pick the cycle with the smallest total.
- The method of trying every possibility to find an optimal solution is referred to as an *exhaustive search*, or use of the *Brute Force Algorithm*.
- Input: weighted complete graph K_n
- Output: Minimum Hamiltonian cycle

Brute Force Algorithm

1. Choose a starting vertex, call it v
2. Find all Hamiltonian cycles starting at v. Calculate the total weight of each cycle.
3. Compare all $(n-1)!$ cycles. Pick one with the least total weight.

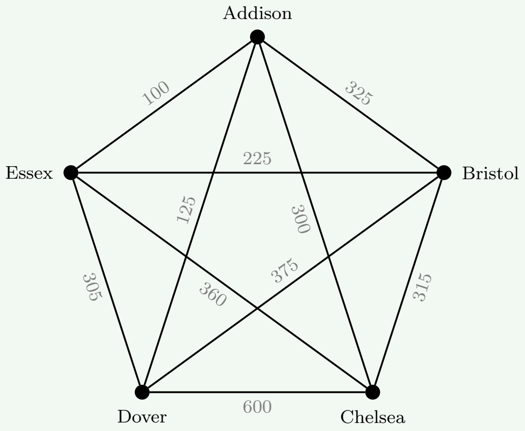
Note

In step 3, there should be at least two options

Brute Force Algorithm – example

Example

Liz is planning her next business trip from her hometown of Addison and has determined the cost for travel between any of the five cities she must visit.



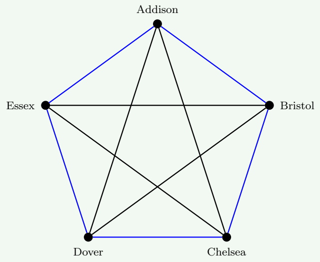
Brute Force Algorithm – example

Example

- Number of possible Hamiltonian cycles: $4 \times 3 \times 2 \times 1 = 24$
- One method for finding all Hamiltonian cycles, and ensuring you indeed have all 24, is to use alphabetical or lexicographic ordering of the cycles.
- Note that all cycles must start and end at Addison, and we will abbreviate all cities with their first letter
- For example, the first cycle is $abcdea$

Brute Force Algorithm – example

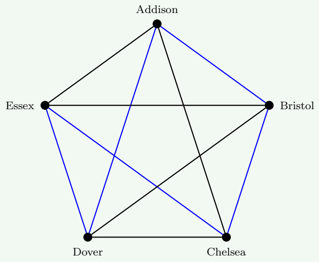
Example



$abcdea$

$aedcba$

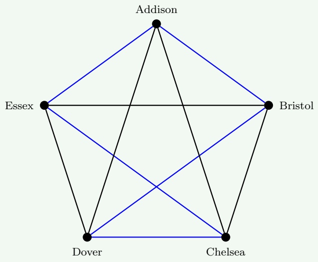
1645



$abceda$

$adecba$

1430



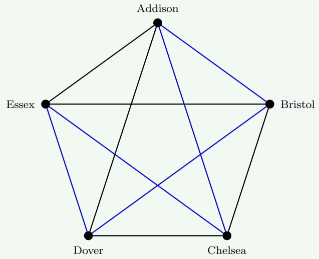
$abdcea$

$aecdba$

1760

Brute Force Algorithm – example

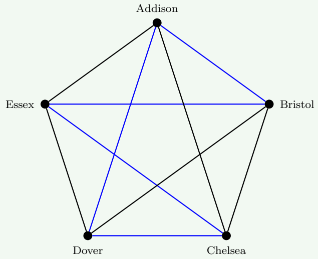
Example



$abdeca$

$acedba$

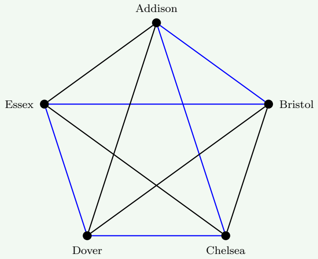
1665



$abecda$

$adceba$

1635



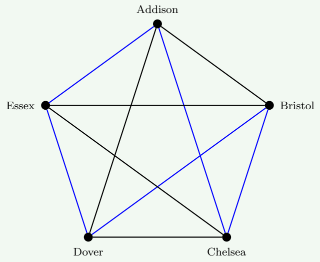
$abedca$

$acdeba$

1755

Brute Force Algorithm – example

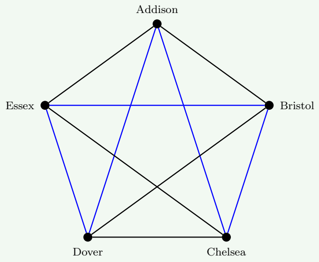
Example



acbdea

aedbca

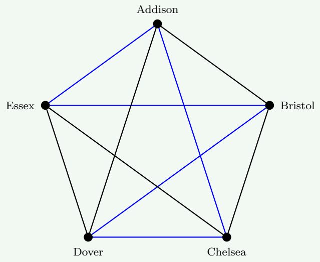
1395



acbeda

adebca

1270



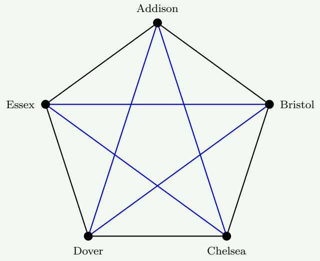
acdbea

aebdca

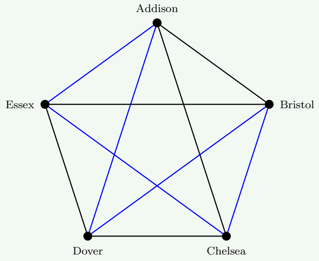
1600

Brute Force Algorithm – example

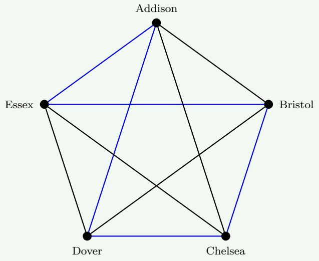
Example



$acebda$
$adbeca$
1385



$adbcea$
$aecbda$
1275



$adcbea$
$aebcda$
1365

Estimated effort

- Effort for using Brute Force Algorithm to find the optimal Hamiltonian cycle in K_n
- Best: best supercomputer
- Top 500: top 500 supercomputers
- Earth is about 4.54×10^9 years old

	Supercomputers	
n	**Best**	**Top 500**
5	2×10^{-15} seconds	2×10^{-16} seconds
15	2×10^{-5} seconds	2×10^{-6} seconds
20	40 seconds	4 seconds
21	14 minutes	2 minutes
22	5 hours	32 minutes
23	4.5 days	12 hours
24	16 weeks	12 days
25	7.5 years	10 months
26	2 centuries	2 decades
30	132 million years	14 million years
40	4.1×10^{23} years	4.3×10^{22} years
50	1.4×10^{40} years	1.5×10^{39} years

Other algorithms

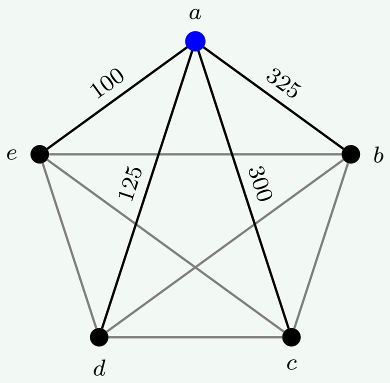
- We will discuss a few more efficient algorithms
- Mathematicians have been searching for algorithms that will find the optimal cycle in a relatively short time span; that is, an algorithm that is both efficient and optimal
- Not only has no such algorithm been found for the Traveling Salesman Problem, but some mathematicians believe no such algorithm even exists.

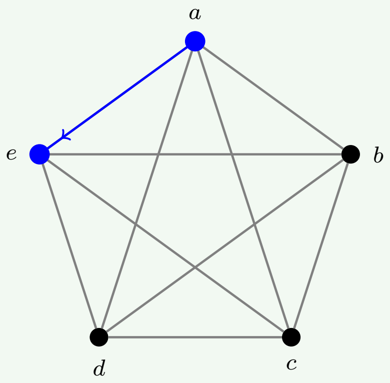
Nearest Neighbor Algorithm

1. Choose a starting vertex, say v. Highlight v
2. Among all edges incident to v, pick the one with the smallest weight. If more than one possible choices have the same weight, randomly pick one.
3. Highlight the edge and move to its other endpoint u. Highlight u
4. Repeat steps 2 and 3, where only edges to unhighlighted vertices are considered
5. Close the cycle by adding the edge to v from the last vertex highlighted. Calculate the total weight.

Nearest Neighbor Algorithm – example

Example

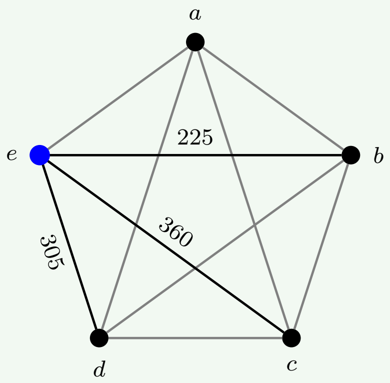


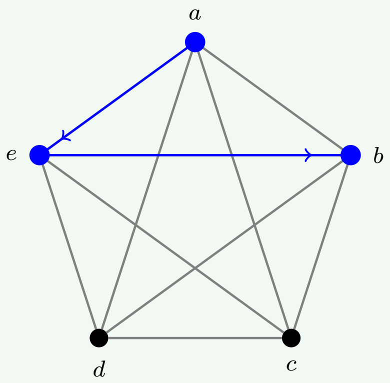


- Step 1: Starting vertex is a
- Step 2: Edge of smallest weight: ae
- Step 3: Move to vertex e. Highlight e

Nearest Neighbor Algorithm – example

Example

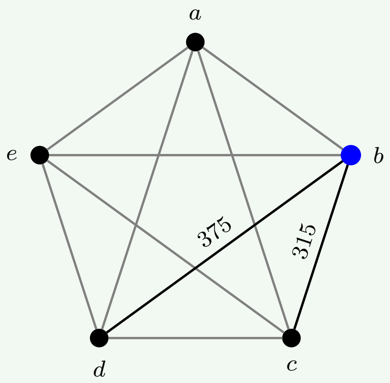


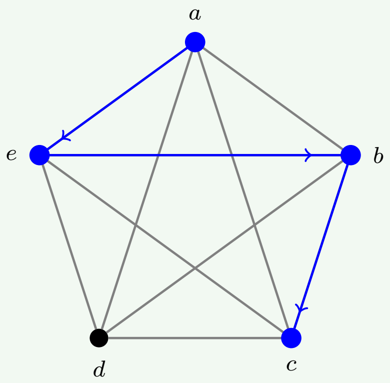


- Step 2: Edge of smallest weight: eb, note here we do not consider edge ea
- Step 3: Move to vertex b. Highlight b

Nearest Neighbor Algorithm – example

Example

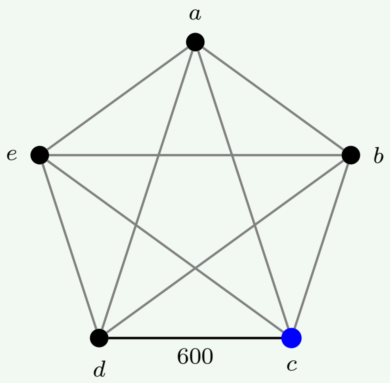


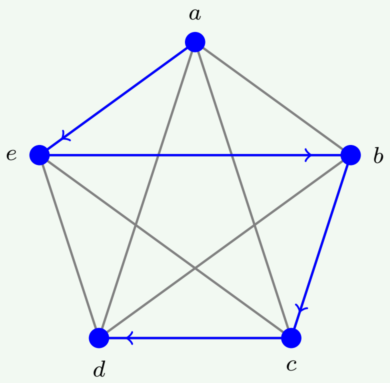


- Step 2: Edge of smallest weight: bc
- Step 3: Move to vertex c. Highlight c

Nearest Neighbor Algorithm – example

Example

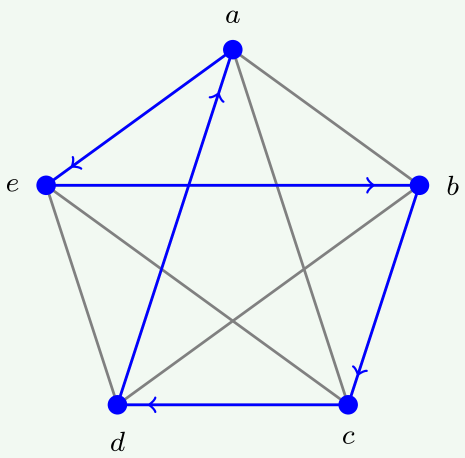




- Step 2: Only choice of edge: cd
- Step 3: Move to vertex d. Highlight d

Nearest Neighbor Algorithm – example

Example



- Step 5: close the cycle. Total weight is 1365

Remarks

- The last two edges are completely determined since we cannot travel back to vertices that have already been chosen
    - This could force us to use the heaviest edges in the graph, as happened above
- The arbitrary choice of a starting vertex could cause light edges to be eliminated from consideration

Though we cannot do anything about the former concern, we can address the latter – by using a different starting vertex, the Nearest Neighbor Algorithm may identify a new Hamiltonian cycle, which may be better or worse than the initial cycle.

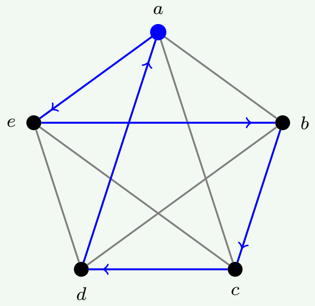
Repetitive Nearest Neighbor Algorithm

1. Choose a starting vertex, say v
2. Apply the Nearest Neighbor Algorithm
3. Repeat steps 1 and 2 so each vertex of K_n serves as the starting vertex
4. Choose the cycle of least total weight. Rewrite it with the desired reference point.

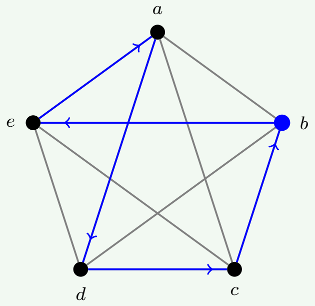
Repetitive Nearest Neighbor Algorithm – example

Example

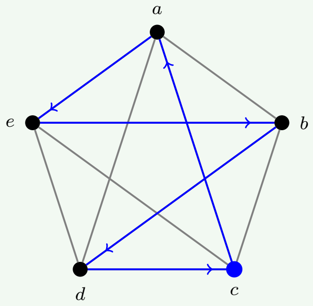
With the same example as before, we can obtain the following cycles with the Repetitive Nearest Neighbor Algorithm



$aebcda$
$aebcda$
1365



$beadcb$
$adcbea$
1365

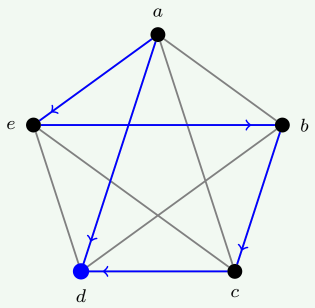


$caebdc$
$aebdca$
1600

Repetitive Nearest Neighbor Algorithm – example

Example

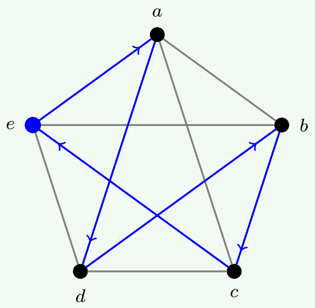
With the same example as before, we can obtain the following cycles with the Repetitive Nearest Neighbor Algorithm



$daebcd$

$aebcda$

1365



$eadbce$

$adbcea$

1275

Remarks

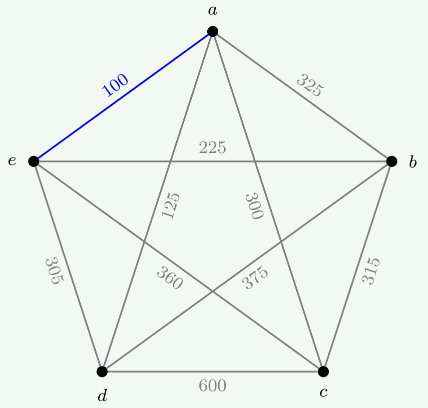
- It should come as no surprise that Repetitive Nearest Neighbor performs better than Nearest Neighbor; however, there is no guarantee that this improvement will produce the optimal cycle.
- Even though Repetitive Nearest Neighbor addressed the concern of missing small weight edges, it is still possible that some of these will be bypassed as we travel a cycle
- Cheapest Link Algorithm: choosing edges in order of weight as opposed to edges along a tour

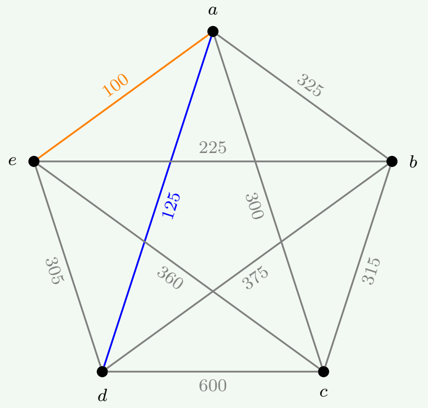
Cheapest Link Algorithm

1. Among all edges in the graph, pick the one with the smallest weight. If more than one possible choices have the same weight, randomly choose one. Highlight the chosen edge
2. Repeat step 1 with the added conditions
   - no vertex has three highlighted edges incident to it
   - no edge is chosen if it would create a cycle that does not include all vertices
3. Calculate the total weight

Cheapest Link Algorithm – example

Example

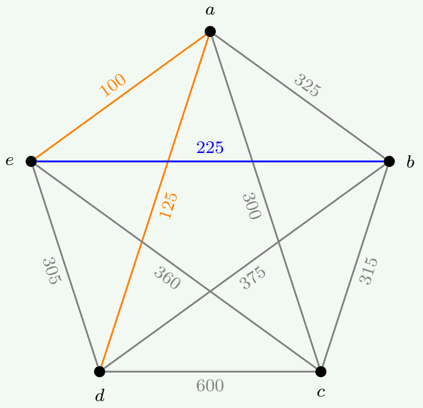


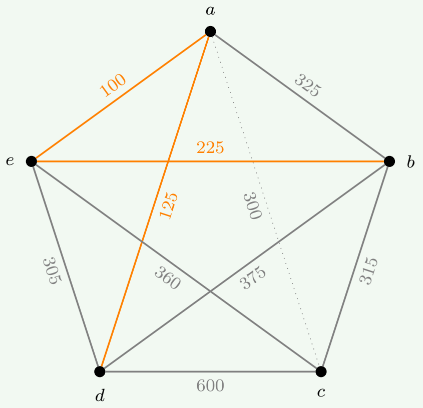


- Step 1. The smallest weight is 100 for edge ae
- Step 2. The next smallest weight is 125 with edge ad

Cheapest Link Algorithm – example

Example

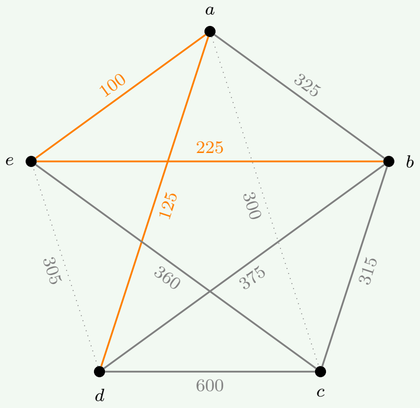


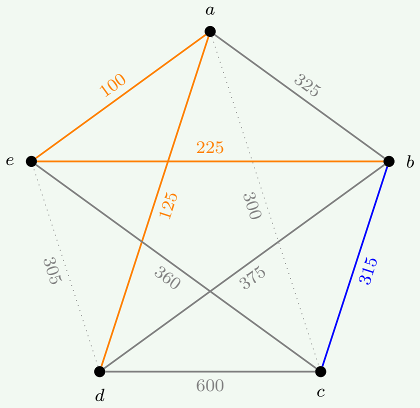


- Step 2. The next smallest weight is 225 for be
- Step 2. ac has the next smallest weight, but highlighting ac will cause a to have three incident edges

Cheapest Link Algorithm – example

Example

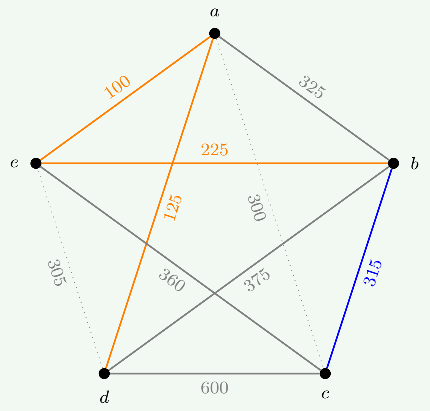


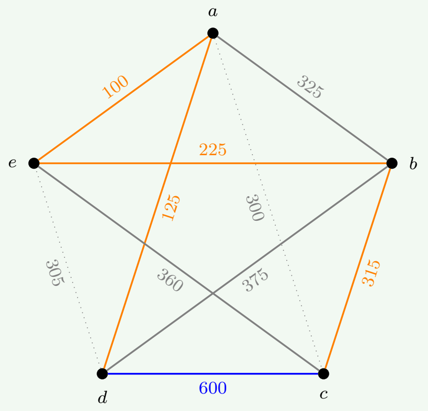


- Step 2. ed has the next smallest weight, but highlighting ed will create a cycle $eade$ that does not contain all vertices, also give e three incident highlighted edges
- Step 2. next available is bc

Cheapest Link Algorithm – example

Example





- Step 2. at this point we must close the cycle and there is only one choice cd
- Output: resulting cycle is $aebcda$ with total weight 1365

Remarks

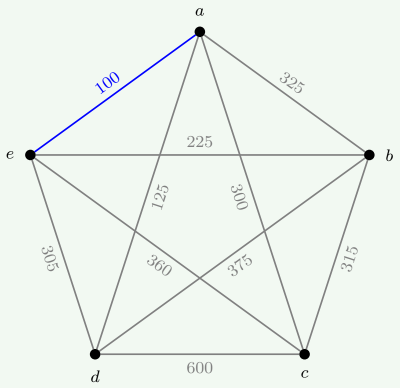
- In the example above, Cheapest Link ran into the same trouble as the initial cycle created using Nearest Neighbor
- Although the lightest edges were chosen, the heaviest also had to be included due to the outcome of the previous steps
- The downfall of the last two algorithms is the focus on choosing the smallest weighted edges at every opportunity.
- In some instances, it may be beneficial to work for a balance —choose more moderate weight edges to avoid later using the heaviest.
- Nearest Insertion Algorithm: forms a small circuit initially from the smallest weighted edge and balances newly added edges with the removal of a previously chosen edge.

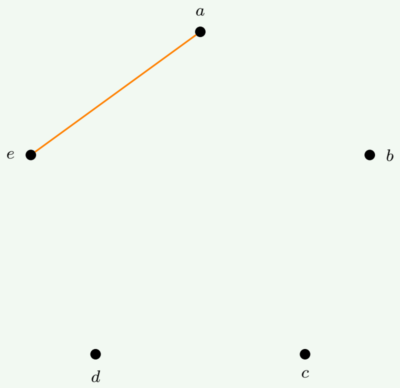
Nearest Insertion Algorithm

1. Among all edges in the graph, pick one with the smallest weight. Highlight the edge and its endpoints.
2. Pick a vertex that is closest to one of the two already chosen vertices. Highlight the new vertex and its edges to both of the previously chosen vertices.
3. Identify the unvisited vertex that is closest to any vertex already in the current cycle. Insert this vertex into the existing cycle by connecting it to the nearest chosen vertex. Then, add a second edge to complete the insertion and remove one existing edge to maintain a cycle. Choose the scenario with the smallest total weight.
4. Repeat step 3 until all vertices have been included in the cycle
5. Calculate the total weight

Nearest Insertion Algorithm – example

Example

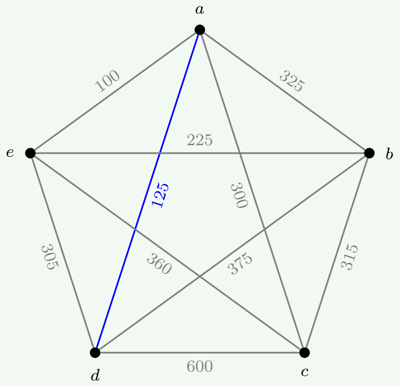


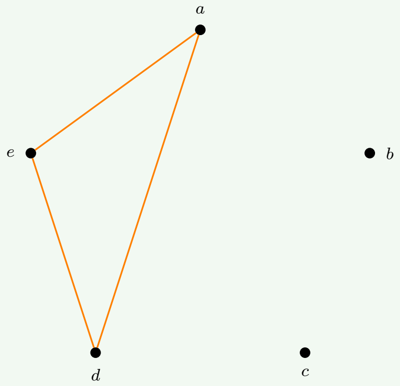


- Step 1. the smallest weight edge is ae

Nearest Insertion Algorithm – example

Example

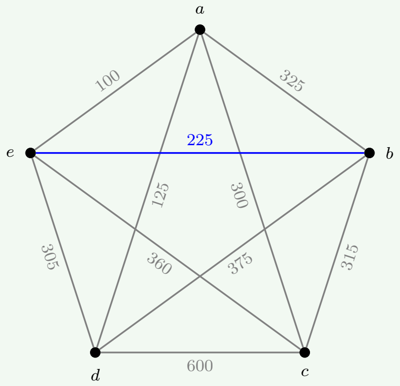


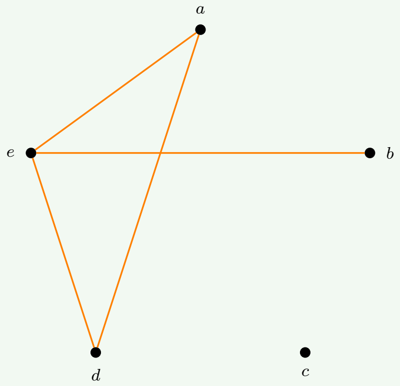


- Step 2: closest vertex to either a or e is d through edge ad. Form a cycle by adding ad and de
- Step 3: closes vertex to any of a, d or e is b through the edge be

Nearest Insertion Algorithm – example

Example



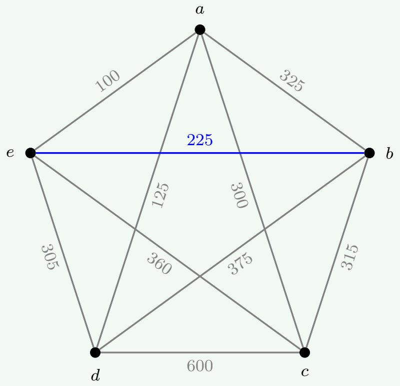


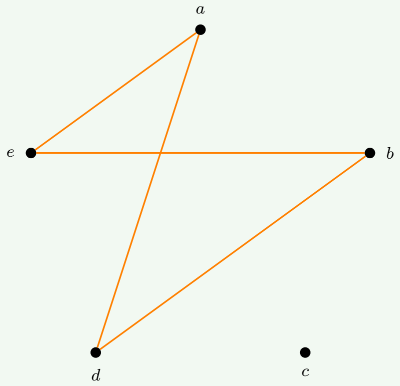
Step 3: After adding edge be, we need to decide whether to add ba and delete ea or add bd and delete ed

$$\begin{aligned} \omega(ba) - \omega(ea) &= 325 - 100 = 225 \\ \omega(bd) - \omega(ed) &= 375 - 305 = \mathbf{70} \end{aligned}$$

Nearest Insertion Algorithm – example

Example

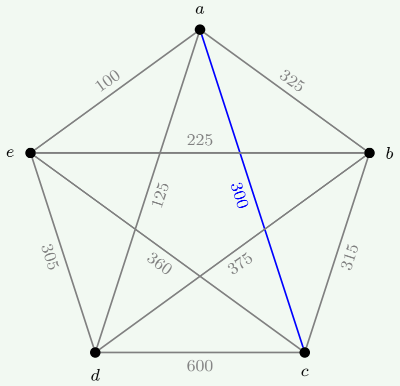


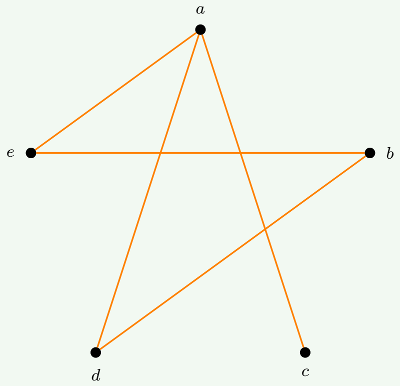


- Step 3: We add edge bd and remove edge ed to get a bigger cycle than before
- Step 3: Only vertex left is c, c is closest to a

Nearest Insertion Algorithm – example

Example



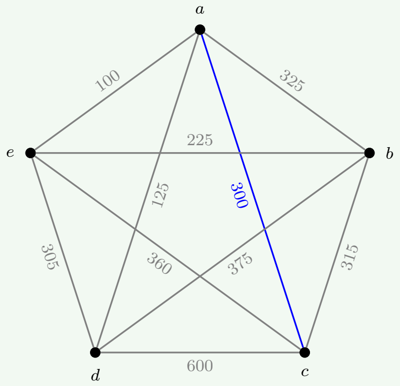


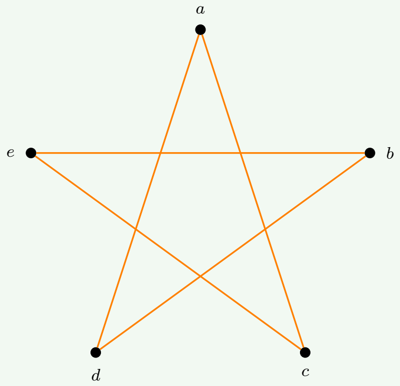
Step 3: After adding edge ca, we need to decide whether to add cd and delete ad or add ce and delete ae

$$\begin{aligned} \omega(cd) - \omega(ad) &= 600 - 125 = 475 \\ \omega(ce) - \omega(ae) &= 360 - 100 = \mathbf{260} \end{aligned}$$

Nearest Insertion Algorithm – example

Example





- Step 3: We add edge ce and remove edge ae to get a bigger cycle than before
- Step 5: The cycle is $acebda$ with total weight 1385.

Hamiltonian cycles

- Existence of Hamiltonian cycles
- Traveling salesman problem
- Digraphs and Asymmetric Traveling Salesman Problem

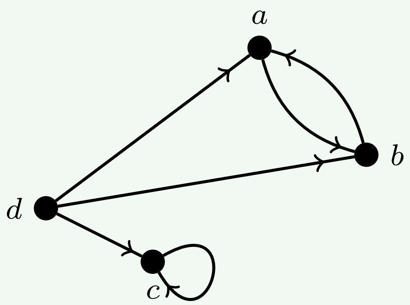
Directed graph

Definition

A *directed graph*, or *digraph*, is a graph $G = (V, A)$ that consists of a vertex set V and an *arc set* A. An *arc* is an ordered pair of vertices.

Example

Let G be a digraph with $V = \{a, b, c, d\}$ and $A = \{ab, ba, cc, dc, db, da\}$.



Terminologies

Definition

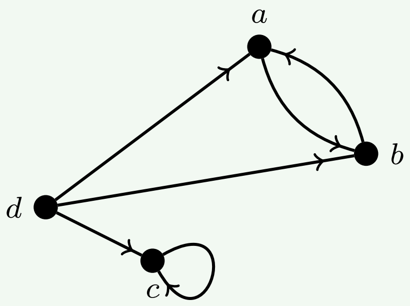
Let $G = (V, A)$ be a digraph

- Arc yx, x is *head*, y is *tail*
- *in-degree* of vertex x: the number of arcs for which x is a head, denoted $\deg^-(x)$
- *out-degree* of vertex x: the number of arcs for which x is the tail, denoted $\deg^+(x)$
- The *underlying graph* for a digraph is the graph $G' = (V, E)$ which is formed by removing the direction from each arc to form an edge
- A *directed path* is a path in which the head of an arc is the tail of the next arc in the path
- A *directed cycle* is a cycle in which the head of an arc is the tail of the next arc in the cycle

Terminologies – example

Example

- a is the tail of arc ab and the head of arcs da and ba
- $\deg^{-1}(a) = 2$, $\deg^{-1}(b) = 2$, $\deg^{-1}(c) = 2$, $\deg^{-}(d) = 0$
- $\deg^{+}(a) = 1$, $\deg^{+}(b) = 1$, $\deg^{+}(c) = 1$, $\deg^{+}(d) = 3$
- $d \rightarrow a \rightarrow b$ is a directed path
- $c \rightarrow d \rightarrow a \rightarrow b$ is not a directed path since cd is not an arc in the graph
- $a \rightarrow b \rightarrow a$ is a directed cycle
- $d \rightarrow a \rightarrow b \rightarrow d$ is not a directed cycle



Degrees of vertices

Theorem

Let $G = (V, A)$ be a digraph and $|A|$ denote the number of arcs in G. Then both the sum of the in-degrees of the vertices and the sum of the out-degrees equals the number of arcs. In other words, if $V = \{v_1, v_2, \ldots, v_n\}$, then

$$\deg^-(v_1) + \cdots \deg^-(v_n) = |A| = \deg^+(v_1) + \cdots \deg^+(v_n)$$

- Since each arc will contribute to the in-degree to its head and to the out-degree of its tail, the sum of the in-degrees equals the sum of the out-degrees.
- Each arc is counted exactly once in the sum of the in-degrees (or out-degrees) since it has a unique head (or tail).

Hamiltonian cycles

A few properties should be immediately apparent for a digraph to have a Hamiltonian cycle

- G must be connected
- No vertex of G can have in-degree or out-degree 0
- G cannot contain a cut-vertex

A sufficient condition for Hamiltonian cycles

Theorem

Let G be a digraph with $n \geq 3$ vertices. If

$$\deg^{-}(v) \geq \frac{n}{2}, \quad \deg^{+}(v) \geq \frac{n}{2}$$

for every vertex v of G, then G has a Hamiltonian cycle.

Asymmetric Traveling Salesman Problem

- The Asymmetric Traveling Salesman Problem is to find the optimal Hamiltonian *directed* cycle on a digraph.
- The weight of the arc ab need not equal the weight of arc ba
- *Complete digraph*: for each pair of vertices x and y both arcs xy and yx exist in the graph
- We will only consider complete digraphs in which each arc has a positive weight
- To utilize the algorithms discussed before, we convert a digraph into an undirected graph

Undirecting Algorithm

Input: Weighted complete digraph $G = (V, A, \omega)$

Steps:

1. For each vertex x, make a clone x'. Form the edge xx' with weight 0
2. For each arc xy form the edge $x'y$
3. The weight of an edge is equal to the weight of its corresponding arc
    - $\omega(xx') = 0$
    - $\omega(x'y) = \omega(xy)$
    - $\omega(xy') = \omega(yx)$

Output: weighted clone graph $G' = (V', E, \omega')$

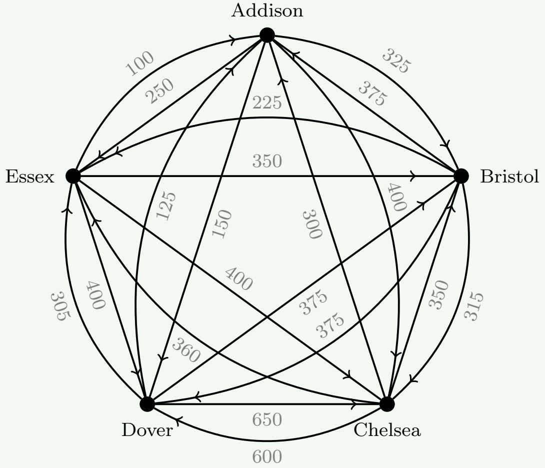
Note

- We need to ensure that when forming a cycle an arc into a vertex is immediately followed by an arc out of that same vertex
- By giving the edge between a vertex and its clone a weight of 0 (when all other arcs have positive weights), we ensure this edge will always be included in any Hamiltonian cycle.
- In addition, this extra edge does not impact the total weight of the final cycle

Undirecting Algorithm – example

Example

Liz is heading out on the road to visit her customers, the direction of a route impacts its cost



Undirecting Algorithm – example

Example

Table representation. e.g. Addison to Bristol - 325

	Addison	Bristol	Chelsea	Dover	Essex
Addison	.	325	400	150	250
Bristol	375	.	315	375	225
Chelsea	300	350	.	600	360
Dover	125	375	650	.	305
Essex	100	350	400	400	.

Undirecting Algorithm – example

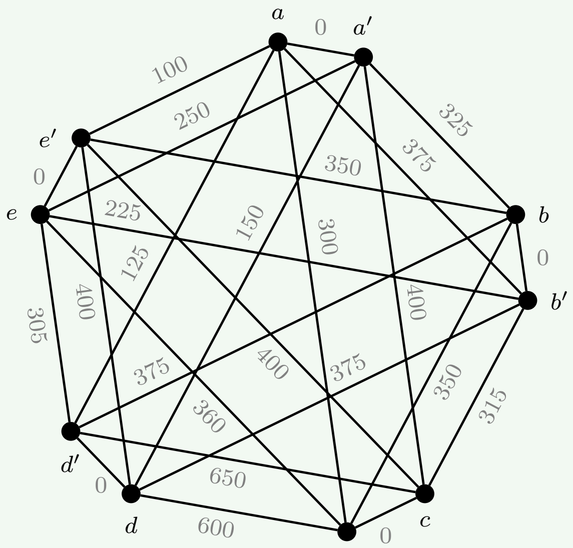
Example

Note: original edges do not exist. The table is symmetric, with a copy of the original table in the lower left quadrant and its mirror image in the upper right quadrant.

	a	b	c	d	e	a'	b'	c'	d'	e'
a	.	325	400	150	250	0	375	300	125	100
b	375	.	315	375	225	325	0	350	375	350
c	300	350	.	600	360	400	315	0	650	400
d	125	375	650	0	305	150	375	600	0	400
e	100	350	400	400	0	250	225	360	305	0
a'	0	325	400	150	250	.	.	.	.	.
b'	375	0	315	375	225	.	.	.	.	.
c'	300	350	0	600	360	.	.	.	.	.
d'	125	375	650	0	305	.	.	.	.	.
e'	100	350	400	400	0	.	.	.	.	.

Undirecting Algorithm – example

Example



Remarks

- Even though the input of the algorithm is a complete digraph, the output is not a complete graph
- The Undirecting Algorithm can be applied to a digraph that is not complete; however, the resulting graph may not have a Hamiltonian cycle.
- When applied to a complete digraph G with n vertices, the resulting graph G' has $2n$ vertices, each of degree n, is guaranteed to have a Hamiltonian cycle.
- The cycle from G' can then be translated to a Hamiltonian directed cycle of the digraph G where vertex copies get reduced back to a single vertex

Asymmetric Traveling Salesman Problem – example

Example

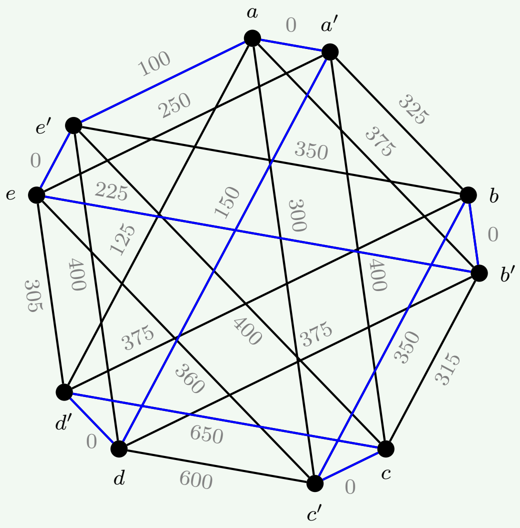
Continue from the previous example, the table below lists the four cycles found using Nearest Neighbor and their conversion to a directed cycle in the digraph.

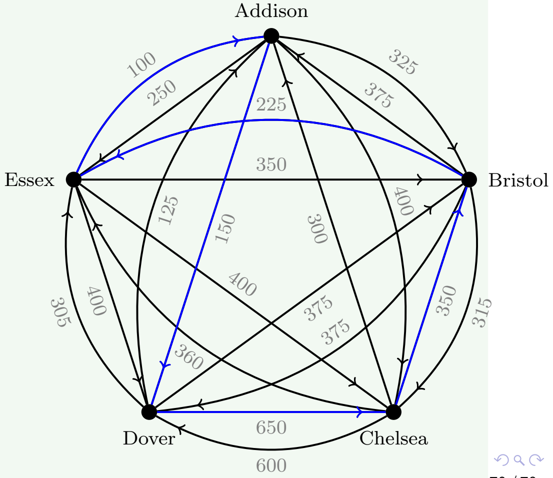
Nearest Neighbor Cycle	Conversion	Total Weight
$a\ a'\ d\ d'\ e\ e'\ b\ b'\ c\ c'\ a$	$a \to d \to e \to b \to c \to a$	1420
$a'\ a\ e'\ e\ b'\ b\ c'\ c\ d'\ d\ a'$	$a \to d \to c \to b \to e \to a$	1475
$e\ e'\ a\ a'\ d\ d'\ b\ b'\ c\ c'\ e$	$e \to a \to d \to b \to c \to e$	1300
$e'\ e\ b'\ b\ c'\ c\ a'\ a\ d'\ d\ e'$	$e \to d \to a \to c \to b \to e$	1500

Note that cycles beginning with a clone vertex must be reversed in the translation back into a direct cycle.

Asymmetric Traveling Salesman Problem – example

Example





Remarks

- Unlike in the undirected case, reversals of a Hamiltonian cycle in a digraph (that is not a complete digraph) might not exist, and those that do will most likely result in a different total weight.
- Thus when applying the Nearest Neighbor Algorithm to graphs formed using the Undirecting Algorithms, we must consider starting at both copies of a vertex, such as a and a', e and e' as in the example

Cheapest Link Algorithm

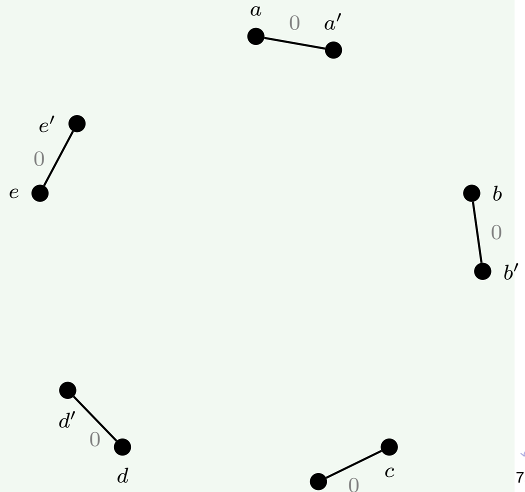
- All edges of weight 0 must be included in the final cycle since picking the edges of minimum weight would initially result in choosing all the weight 0 edges as no two of these are adjacent (so there is no concern of a vertex having degree 3 or closing the circuit too early)

Cheapest Link Algorithm – example

Example

Continuing from the previous example.

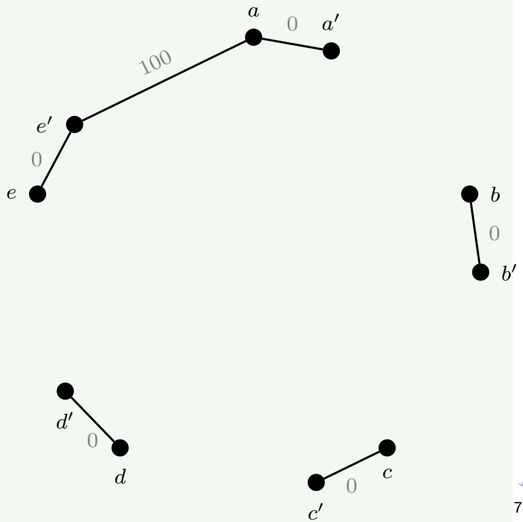
- Step 1. Since all edges between a vertex and its clone have a weight of 0, all of these edges will be chosen



Cheapest Link Algorithm – example

Example

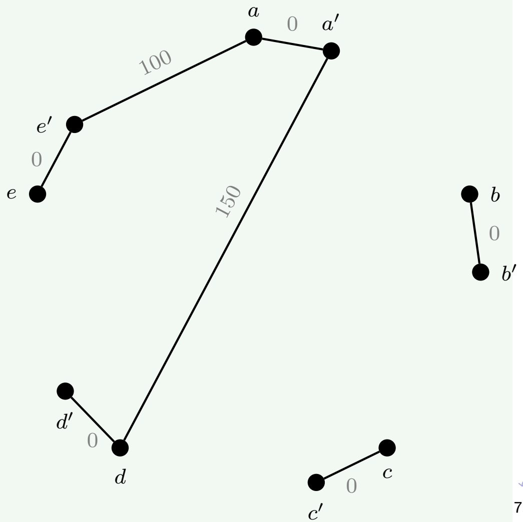
- Step 2. The edge of smallest weight is ae'



Cheapest Link Algorithm – example

Example

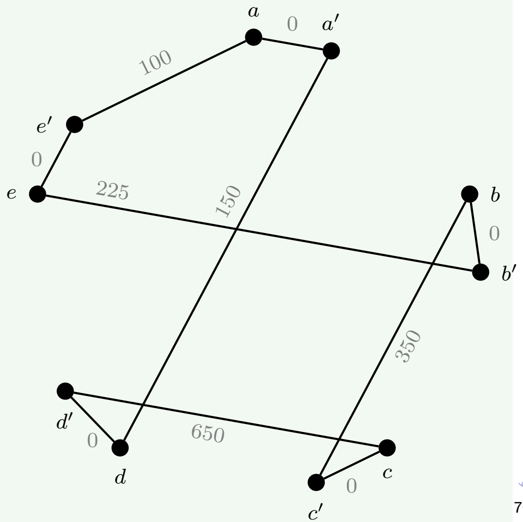
- Step 2. The next lowest is ad'. However, we cannot choose this edge since it would cause a to be incident to three highlighted edges. We skip this edge and move to $a'd$.



Cheapest Link Algorithm – example

Example

- Step 2. Continuing with the Algorithm, we add edges $b'e$, $c'b$. Then we close the cycle with edge cd'.

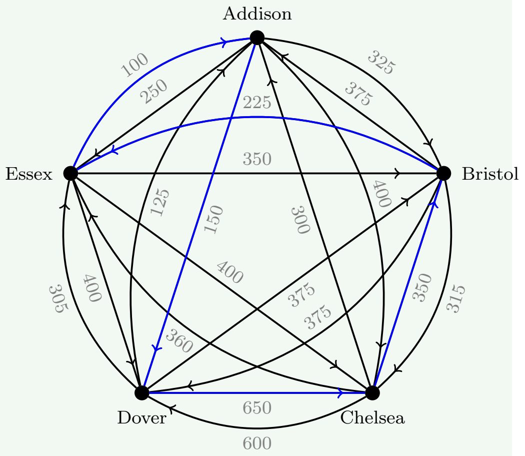


Cheapest Link Algorithm – example

Example

- Step 2. Convert the cycle $aa'dd'cc'bb'ee'a$ to directed cycle in the digraph

$$a \to d \to c \to b \to e \to a$$



Nearest Insertion Algorithm

- The Nearest Insertion Algorithm in its original form does not function on the weighted clone graphs
- There are no cycles of length 3 in the graph created by the undirecting Algorithm and the second step of Nearest Insertion results in a cycle on three vertices originating from the cheapest edge of the graph
- A modification of Nearest Insertion for the weighted clone graphs, which treats the weight 0 edges differently will be discussed during the tutorial

Midterm

- Room 1.37 (same as for lecture and tutorial)
- Written
- Bring calculator!
- Time: next Friday (14th Nov), $9:30 - 11:30$am
- 50 marks
- 8 questions
- Covers: Lectures 1 – 4, Tutorials 1 – 5
- Write your answers on the provided answer sheets. Additional sheets will be supplied upon request. Please ensure that your full name is clearly written on each page of the answer sheets.
- Include detailed computation steps for all solutions. Answers without supporting calculations will receive a score of zero.