EE3301

Optimization Methods for Engineering

Group Learning Report 2023

Name: Ng Chung Wah

Student ID: 57147463

Supervisor: Prof. Moshe Zukerman

Contents

Overview

Week	Number of Posts	Dates and Times
1	1	06-09-2023 (10:13 pm)
2	1	16-09-2023 (4:27 pm)
3	1	22-09-2023 (6:24 pm)
4	1	30-09-2023 (9:03 am)
5	1	7-10-2023 (11:43 am)
6	0	
7	1	21-10-2023 (2:50 pm)
8	1	28-10-2023 (9:10 pm)
9	2	1-11-2023 (10:02 am), 4-11-2023 (11:33 am)
10	1	11-11-2023 (10:56 pm)
11	1	18-11-2023 (10:28 pm)
12	1	25-11-2023 (2:25 pm)
13	1	02-12-2023 (12:54 am)

Total number of posts = 13

Self-Assessment and Self-Justification

I expect a mark of A- in Group Learning because of my proactive extracurricular research and continuous collaborative contributions throughout the semester.

My primary focus has been on delving into concepts beyond the prescribed notes, reflecting my commitment for a deeper understanding of the subject matter. I also did not forget to include practical solutions to course homework and exercises, serving as tangible evidence of my grasp on the course content. This approach not only showcases my intellectual curiosity as a passionate learner but also demonstrates a commitment to master content beyond the established curriculum.

Throughout the semester, my commitment to self-learning and novelty has been evident in my proactive exploration of extracurricular algorithms and concepts, including but not limited to: proof of new theorems, network topology, feasibility constraints and unbounded knapsack problem. I embarked on original research ("Proof of Ore's Theorem") inspired by peers (Chi Yan Wan), introducing several new algorithms, showcasing a genuine passion for knowledge beyond the prescribed curriculum.

Demonstrating a keen sense of initiative, I tackled problems strategically, offering solutions surpassing surface-level understanding. Using both C++ programming and mathematical calculations, I have ensured not only correctness but also a profound depth of comprehension. Beyond individual achievement, my collaborative spirit is reflected in constructive feedback to peers and sharing of original, elegant solutions.

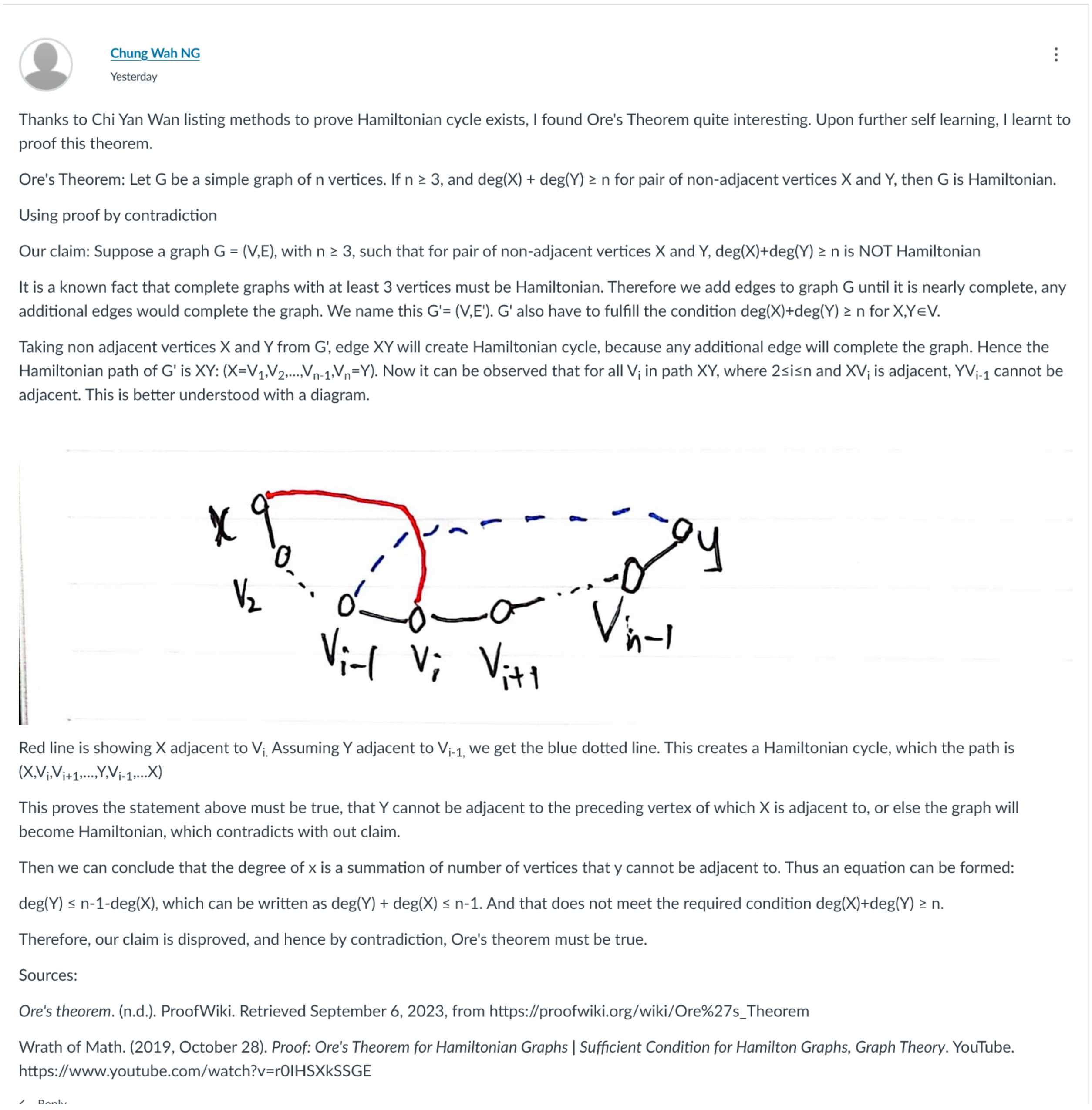
The use of PDF and hand-drawn diagrams underscores my commitment to facilitate understanding through effective presentation. Upholding the highest standards of academic ethics, I've maintained integrity in all aspects of my posts by referencing peers when creditable and sources in APA7 format.

In addition to the aforementioned accomplishments, my continuous participation serves as proof of my engagement throughout the semester. While the requirement was a minimum of 10 posts, I surpassed it by contributing a total of 13 posts, consistently maintaining on average one post per week. This active involvement demonstrates my sustained commitment to the intellectual discourse, fostering a collaborative and learning environment for knowledge exchange.

In essence, my pursuit of excellence is evidenced by a combination of rigorous research, practical application, and an unwavering commitment to extracurricular self-learning. I am confident that these aspects, collectively, position me for an A grade, reflecting both a depth of understanding and a proactive approach to academic challenges.

Week 1 (from 6 Sep 2023 to 12 Sep 2023)

1. Proof of Ore's Theorem used in determining existence of Hamiltonian cycle.



Chung Wah NG

Yesterday

Thanks to Chi Yan Wan listing methods to prove Hamiltonian cycle exists, I found Ore's Theorem quite interesting. Upon further self learning, I learnt to proof this theorem.

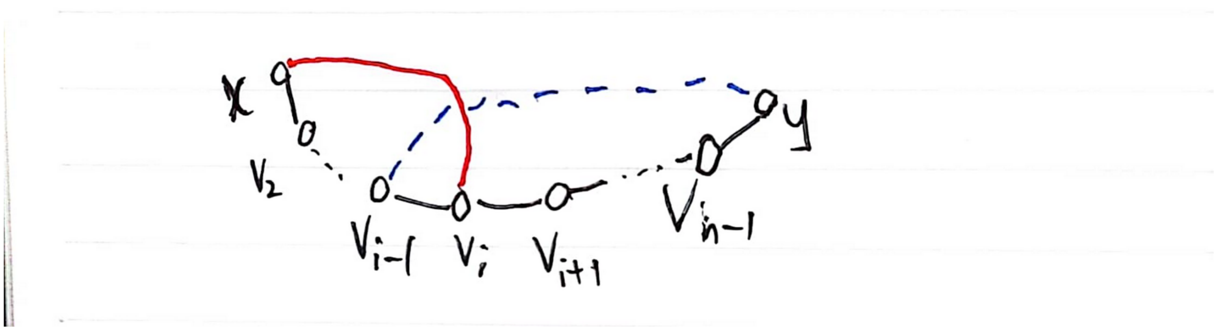
Ore's Theorem: Let G be a simple graph of n vertices. If n ≥ 3, and deg(X) + deg(Y) ≥ n for pair of non-adjacent vertices X and Y, then G is Hamiltonian.

Using proof by contradiction

Our claim: Suppose a graph G = (V,E), with n ≥ 3, such that for pair of non-adjacent vertices X and Y, deg(X)+deg(Y) ≥ n is NOT Hamiltonian

It is a known fact that complete graphs with at least 3 vertices must be Hamiltonian. Therefore we add edges to graph G until it is nearly complete, any additional edges would complete the graph. We name this G'= (V,E'). G' also have to fulfill the condition deg(X)+deg(Y) ≥ n for X,Y∈V.

Taking non adjacent vertices X and Y from G', edge XY will create Hamiltonian cycle, because any additional edge will complete the graph. Hence the Hamiltonian path of G' is XY: ($X=V_1,V_2,\ldots,V_{n-1},V_n=Y$). Now it can be observed that for all V_i in path XY, where $2 \le i \le n$ and XV_i is adjacent, YV_{i-1} cannot be adjacent. This is better understood with a diagram.



Red line is showing X adjacent to V_i. Assuming Y adjacent to V_{i-1}, we get the blue dotted line. This creates a Hamiltonian cycle, which the path is ($X,V_i,V_{i+1},\ldots,Y,V_{i-1},\ldots X$)

This proves the statement above must be true, that Y cannot be adjacent to the preceding vertex of which X is adjacent to, or else the graph will become Hamiltonian, which contradicts with out claim.

Then we can conclude that the degree of x is a summation of number of vertices that y cannot be adjacent to. Thus an equation can be formed:

deg(Y) ≤ n-1-deg(X), which can be written as deg(Y) + deg(X) ≤ n-1. And that does not meet the required condition deg(X)+deg(Y) ≥ n.

Therefore, our claim is disproved, and hence by contradiction, Ore's theorem must be true.

Sources:

Ore's theorem. (n.d.). ProofWiki. Retrieved September 6, 2023, from https://proofwiki.org/wiki/Ore%27s_Theorem

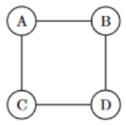
Wrath of Math. (2019, October 28). *Proof: Ore's Theorem for Hamiltonian Graphs | Sufficient Condition for Hamilton Graphs, Graph Theory*. YouTube. https://www.youtube.com/watch?v=r0IHSXkSSGE

Reply

Below is the comment section on this post:

Kam To MA
Sep 9, 2023

This is such a faster way to disprove the existence of a Hamiltonian cycle in a graph!
While Ore's Theorem provides a necessary condition for the existence of a Hamiltonian cycle in a graph, it does not provide a sufficient condition. Let's see the graph below:



- Vertices: A, B, C, D
- Edges: AB, AC, BD, CD

Let's calculate the degrees of each vertex:

- deg(A) = 2
- deg(B) = 2
- deg(C) = 2
- deg(D) = 2

Let's check the degree sum condition for every pair of non-adjacent vertices:

1. For vertices A and B: deg(A) + deg(B) = 2 + 2 = 4 ≥ 4
2. For vertices A and C: deg(A) + deg(C) = 2 + 2 = 4 ≥ 4
3. For vertices A and D: deg(A) + deg(D) = 2 + 2 = 4 ≥ 4
4. For vertices B and C: deg(B) + deg(C) = 2 + 2 = 4 ≥ 4
5. For vertices B and D: deg(B) + deg(D) = 2 + 2 = 4 ≥ 4
6. For vertices C and D: deg(C) + deg(D) = 2 + 2 = 4 ≥ 4

This graph satisfies the degree sum condition of Ore's Theorem for every pair of non-adjacent vertices. However, it does not have a Hamiltonian cycle because there is no way to visit all vertices exactly once and return to the starting vertex.

Chung Wah NG
Sep 9, 2023

Thank you for the extra information. I am aware that proving a Hamiltonian cycle is a NP-Complete problem, meaning there it cannot be completely proofed, with this new information I now fully understand Ore's theorem more.

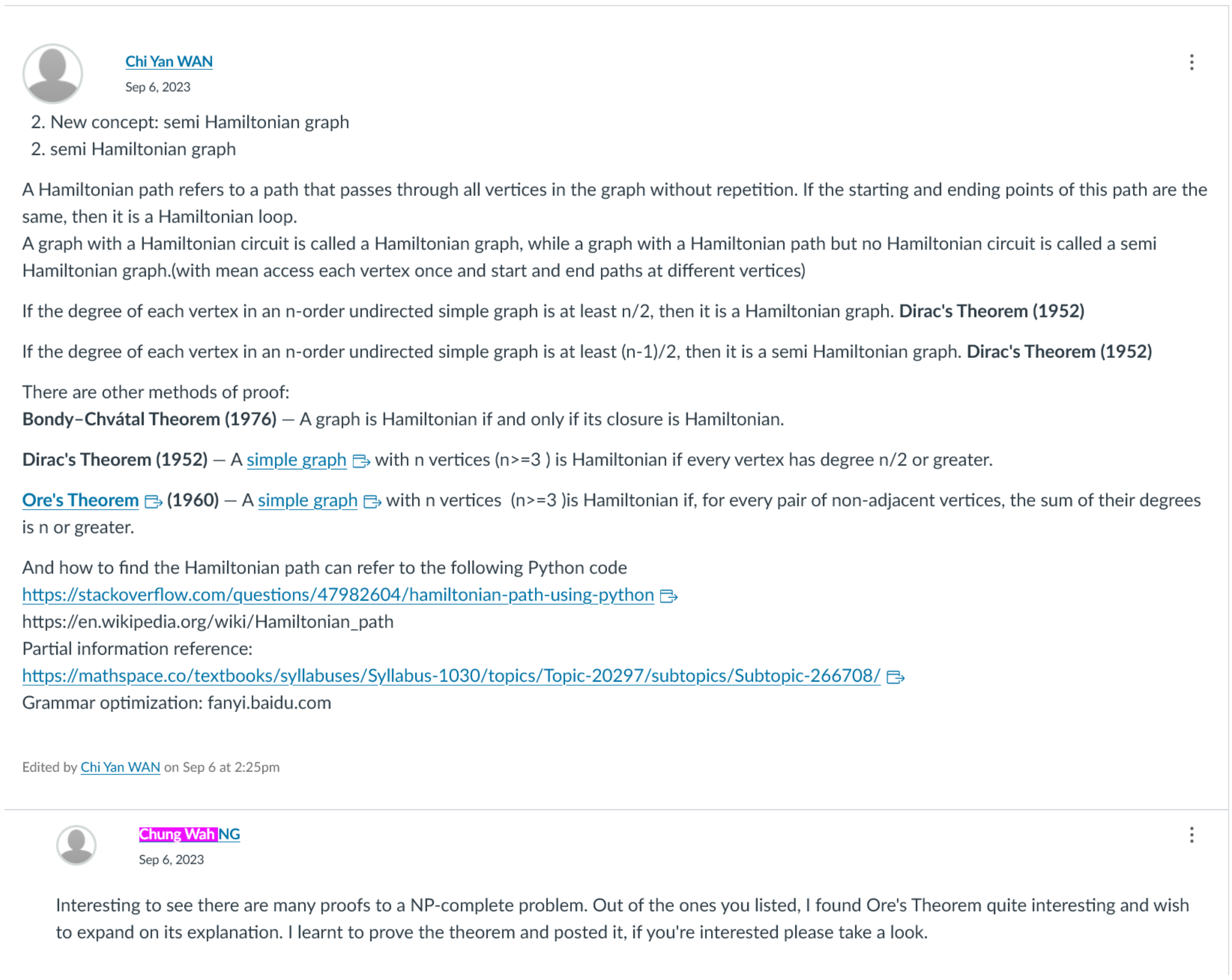
Chi Yan WAN
Sep 9, 2023

Thank you for the derivation of my answer. The Ole's theorem provides a sufficient condition, not a necessary condition. That is to say, if a graph satisfies the conditions of the Olay theorem, it must be a Hamiltonian graph, but if a graph does not satisfy the conditions of the Olay theorem, it cannot be concluded that it is not a Hamiltonian graph.

Wang Kit LIU
Sep 10, 2023

Big thanks for sharing this Ore's Theorem breakdown! You saved me from drowning in graph theory confusion. Your explanation made it crystal clear. Learning about Ore's Theorem has really piqued my interest in graph theory. It's fascinating how a simple concept can have such profound implications in understanding graphs.

Comments on other student's post:



Chi Yan WAN
Sep 6, 2023

2. New concept: semi Hamiltonian graph
2. semi Hamiltonian graph

A Hamiltonian path refers to a path that passes through all vertices in the graph without repetition. If the starting and ending points of this path are the same, then it is a Hamiltonian loop.
A graph with a Hamiltonian circuit is called a Hamiltonian graph, while a graph with a Hamiltonian path but no Hamiltonian circuit is called a semi Hamiltonian graph.(with mean access each vertex once and start and end paths at different vertices)

If the degree of each vertex in an n-order undirected simple graph is at least n/2, then it is a Hamiltonian graph. **Dirac's Theorem (1952)**

If the degree of each vertex in an n-order undirected simple graph is at least (n-1)/2, then it is a semi Hamiltonian graph. **Dirac's Theorem (1952)**

There are other methods of proof:
Bondy–Chvátal Theorem (1976) — A graph is Hamiltonian if and only if its closure is Hamiltonian.

Dirac's Theorem (1952) — A simple graph with n vertices (n>=3) is Hamiltonian if every vertex has degree n/2 or greater.

Ore's Theorem (1960) — A simple graph with n vertices (n>=3)is Hamiltonian if, for every pair of non-adjacent vertices, the sum of their degrees is n or greater.

And how to find the Hamiltonian path can refer to the following Python code
https://stackoverflow.com/questions/47982604/hamiltonian-path-using-python
https://en.wikipedia.org/wiki/Hamiltonian_path
Partial information reference:
https://mathspace.co/textbooks/syllabuses/Syllabus-1030/topics/Topic-20297/subtopics/Subtopic-266708/
Grammar optimization: fanyi.baidu.com

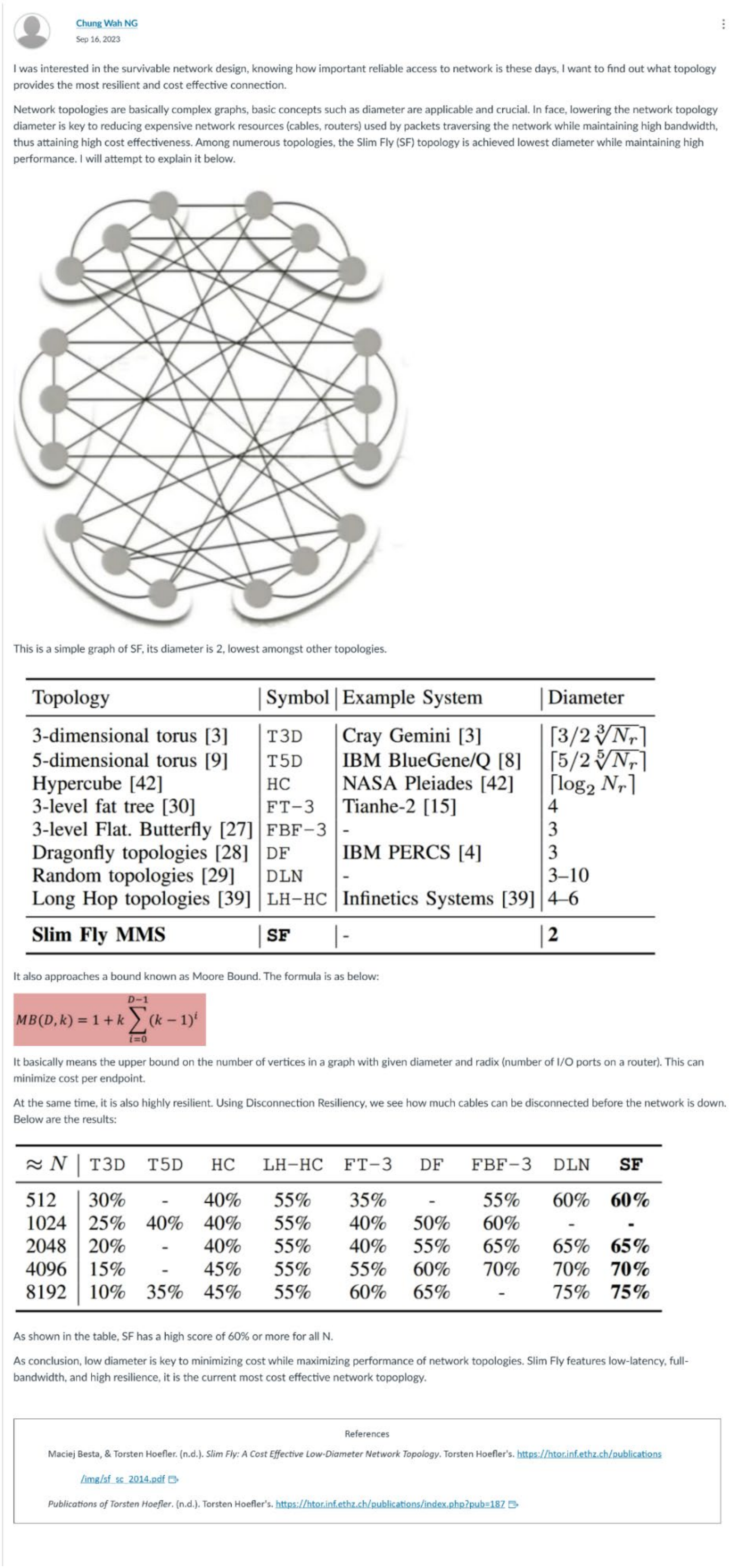
Edited by Chi Yan WAN on Sep 6 at 2:25pm

Chung Wah NG
Sep 6, 2023

Interesting to see there are many proofs to a NP-complete problem. Out of the ones you listed, I found Ore's Theorem quite interesting and wish to expand on its explanation. I learnt to prove the theorem and posted it, if you're interested please take a look.

Week 2 (from 10 Sep 2023 to 16 Sep 2023)

1. Research on the most resilient and cost effective network topology based on graph theory.

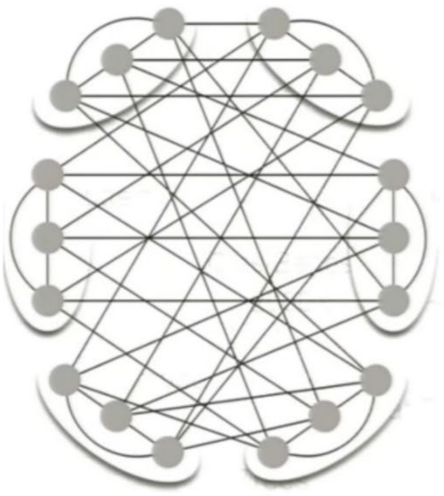


Chung Wah NG

Sep 16, 2023

I was interested in the survivable network design, knowing how important reliable access to network is these days, I want to find out what topology provides the most resilient and cost effective connection.

Network topologies are basically complex graphs, basic concepts such as diameter are applicable and crucial. In face, lowering the network topology diameter is key to reducing expensive network resources (cables, routers) used by packets traversing the network while maintaining high bandwidth, thus attaining high cost effectiveness. Among numerous topologies, the Slim Fly (SF) topology is achieved lowest diameter while maintaining high performance. I will attempt to explain it below.



This is a simple graph of SF, its diameter is 2, lowest amongst other topologies.

Topology	Symbol	Example System	Diameter
3-dimensional torus [3]	T3D	Cray Gemini [3]	$\lceil 3/2 \sqrt[3]{N_r} \rceil$
5-dimensional torus [9]	T5D	IBM BlueGene/Q [8]	$\lceil 5/2 \sqrt[5]{N_r} \rceil$
Hypercube [42]	HC	NASA Pleiades [42]	$\lceil \log_2 N_r \rceil$
3-level fat tree [30]	FT-3	Tianhe-2 [15]	4
3-level Flat. Butterfly [27]	FBF-3	-	3
Dragonfly topologies [28]	DF	IBM PERCS [4]	3
Random topologies [29]	DLN	-	3–10
Long Hop topologies [39]	LH-HC	Infinetics Systems [39]	4–6
Slim Fly MMS	**SF**	-	**2**

It also approaches a bound known as Moore Bound. The formula is as below:

$$MB(D,k) = 1 + k \sum_{i=0}^{D-1} (k-1)^i$$

It basically means the upper bound on the number of vertices in a graph with given diameter and radix (number of I/O ports on a router). This can minimize cost per endpoint.

At the same time, it is also highly resilient. Using Disconnection Resiliency, we see how much cables can be disconnected before the network is down. Below are the results:

$\approx N$	T3D	T5D	HC	LH-HC	FT-3	DF	FBF-3	DLN	**SF**
512	30%	-	40%	55%	35%	-	55%	60%	**60%**
1024	25%	40%	40%	55%	40%	50%	60%	-	**-**
2048	20%	-	40%	55%	40%	55%	65%	65%	**65%**
4096	15%	-	45%	55%	55%	60%	70%	70%	**70%**
8192	10%	35%	45%	55%	60%	65%	-	75%	**75%**

As shown in the table, SF has a high score of 60% or more for all N.

As conclusion, low diameter is key to minimizing cost while maximizing performance of network topologies. Slim Fly features low-latency, full-bandwidth, and high resilience, it is the current most cost effective network topoplogy.

References

Maciej Besta, & Torsten Hoefler. (n.d.). *Slim Fly: A Cost Effective Low-Diameter Network Topology*. Torsten Hoefler's. https://htor.inf.ethz.ch/publications/img/sf_sc_2014.pdf

Publications of Torsten Hoefler. (n.d.). Torsten Hoefler's. https://htor.inf.ethz.ch/publications/index.php?pub=187

Week 3 (from 17 Sep 2023 to 23 Sep 2023)

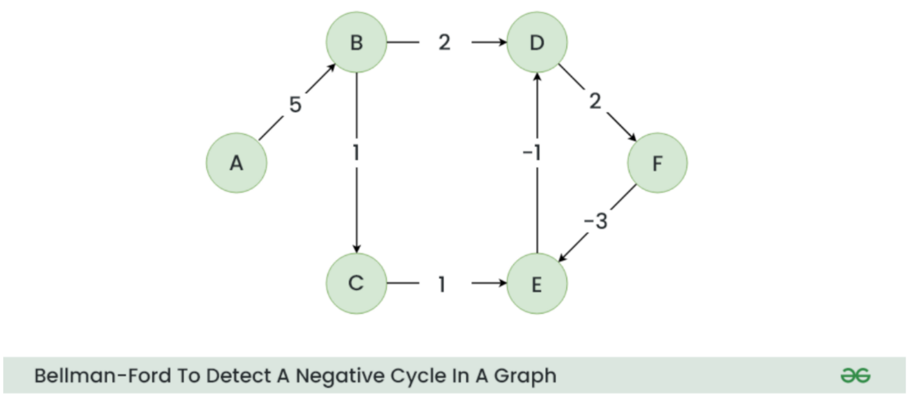
1. Researched on Bellman-Ford's Algorithm to detect negative cycles in graph.

Chung Wah NG

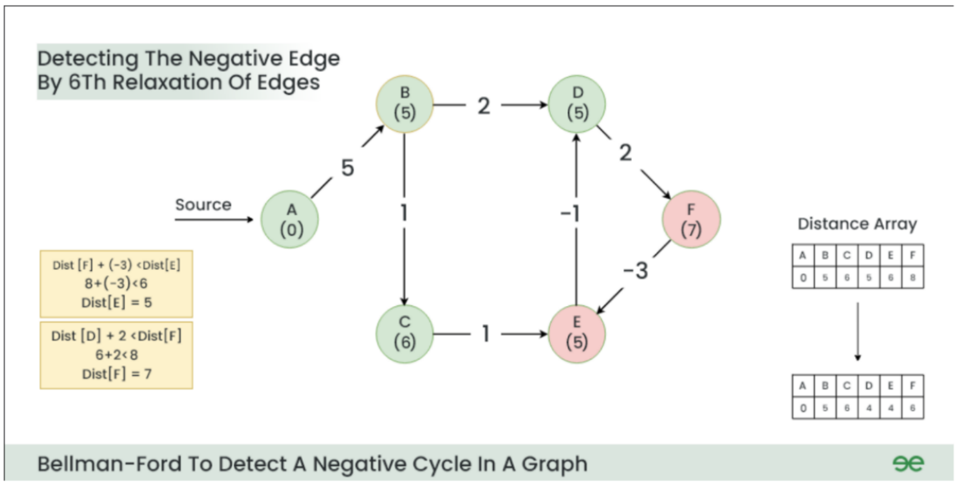
Sep 22, 2023

I wonder what will happen if path weight is a negative number, and I see that Dijkstra algorithm cannot handle negative numbers. So I found another algorithm, namely the Bellman-Ford Algorithm. Its slower than Dijkstra, but more versatile as it also considers negative cycles. Negative cycles are important, as it means traversing the graph will reduce total cost of distance.

The algorithm uses principle of relaxation, stating that for the graph having **N** vertices, all the edges should be relaxed **N-1** times to compute the single source shortest path. It isnt too complicated, below is an example.



- Initialize distance array dist[] for each vertex 'v' as **dist[v] = INFINITY**.
- Assume any vertex (let's say '0') as source and assign **dist = 0**. This case, node A will be 0.
- Now we perform something called relaxation. Its simply an equation:
  - **dist[v] = minimum(dist[v], distance[u] + weight)**
  - **At node B: dist(B) > dist(A)+weight(AB), hence B takes new value, dist(B) = 5**
- Now repeat for every node after B. B is connected to 2 nodes, C and D. So we apply the equation to those two seperately.
  - B To C: **dist(C) > dist(B)+weight(BC), INF>5+1, dist(C) = 6**
  - B To D: **dist(D) > dist(B)+weight(BD), INF>5+2, dist(D) = 7**
  - D To F: **dist(F) > dist(D)+weight(DF), INF>7+2, dist(F) = 9**
  - C to E: **INF>6+1, dist(E) = 7**
  - D To E: **INF>7+(-1), dist(E) = 6, note that E has a new minimum dist, so we take 6 as new dist(E)**
  - C to E: **INF>6+1, dist(E) = 7**
  - F to E: **INF>9+(-3), dist(E) = 6<7, dist(E) = 6**
  - C to E: **INF>6+1, dist(E) = 7**
  - **E to D: 7>7-1, dist(D)=6**
  - **F to E: 7>9-3, dist(E) = 6**
  - **D to F: 9>6+2, dist(F) = 8**
  - **E to D: 6>6-1, dist(D) = 5**
  - **F to E: 6>8-3, dist(E) = 5**
  - **D to F: 8>5+2, dist(F) = 7**
- There should be in total 6 relaxations. After all 6, it should look like this.



A	B	C	D	E	F
0	5	6	5	6	8

A	B	C	D	E	F
0	5	6	4	4	6

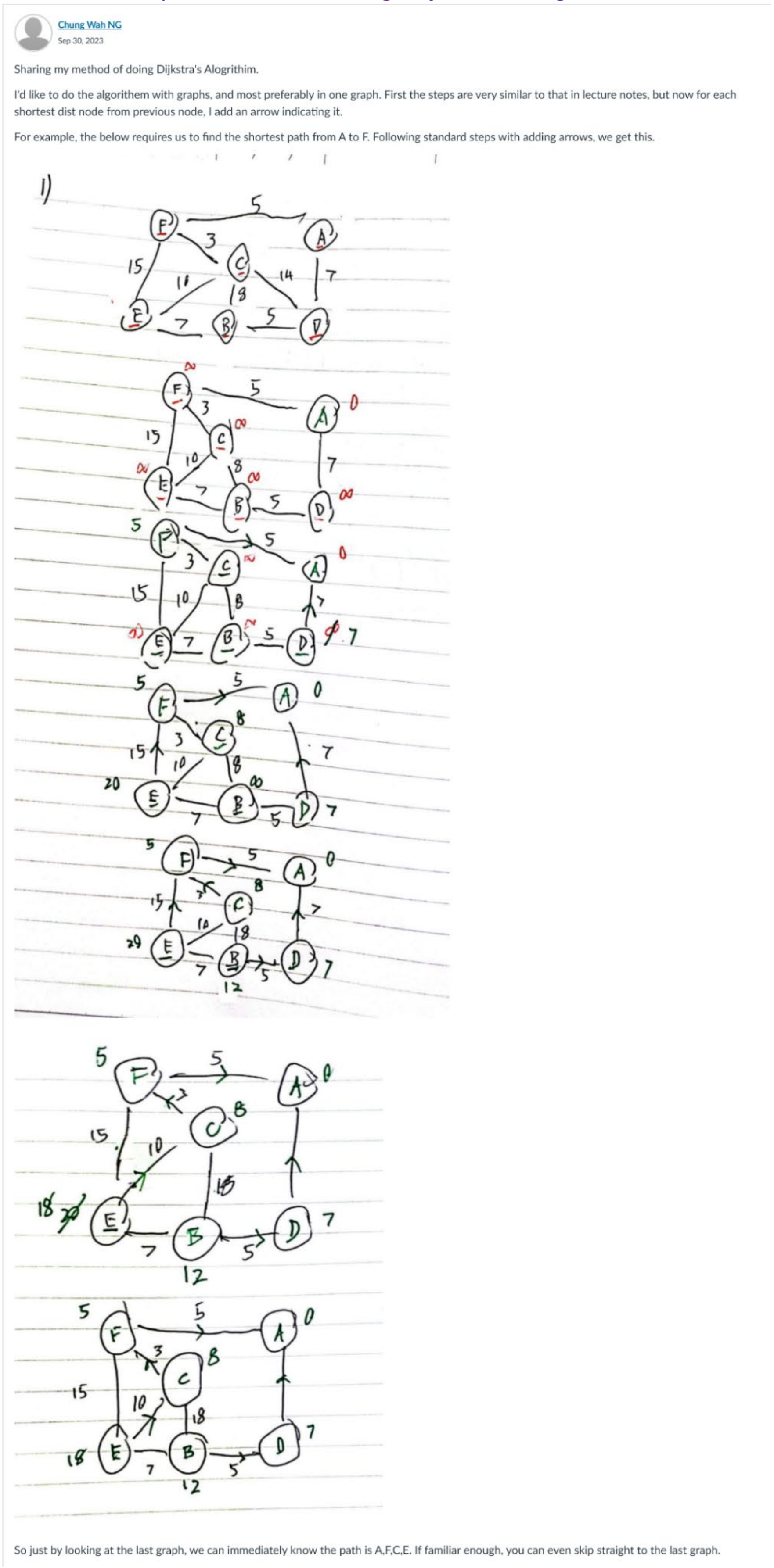
Hence negative cycle (D->F->E) exists

References

Bellman–Ford algorithm | DP-23. (2023, February 15). GeeksforGeeks. https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/

Week 4 (from 24 Sep 2023 to 30 Sep 2023)

1. Shared my method of solving Dijkstra's Algorithm.

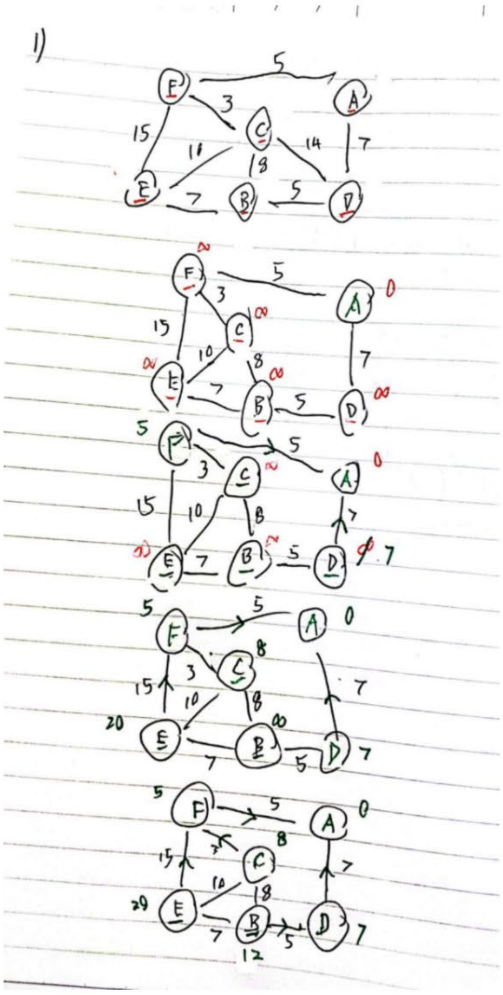


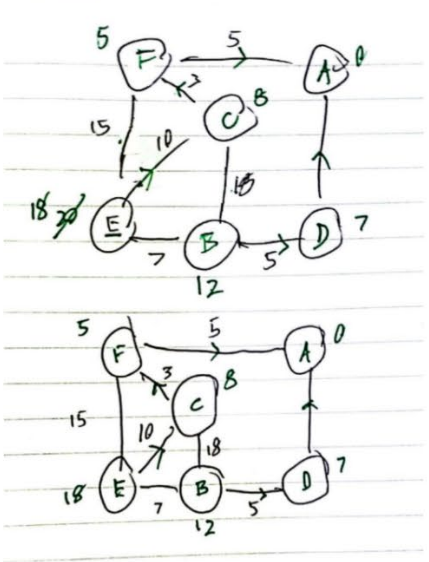
Chung Wah NG
Sep 30, 2023

Sharing my method of doing Dijkstra's Alogrithim.

I'd like to do the algorithem with graphs, and most preferably in one graph. First the steps are very similar to that in lecture notes, but now for each shortest dist node from previous node, I add an arrow indicating it.

For example, the below requires us to find the shortest path from A to F. Following standard steps with adding arrows, we get this.

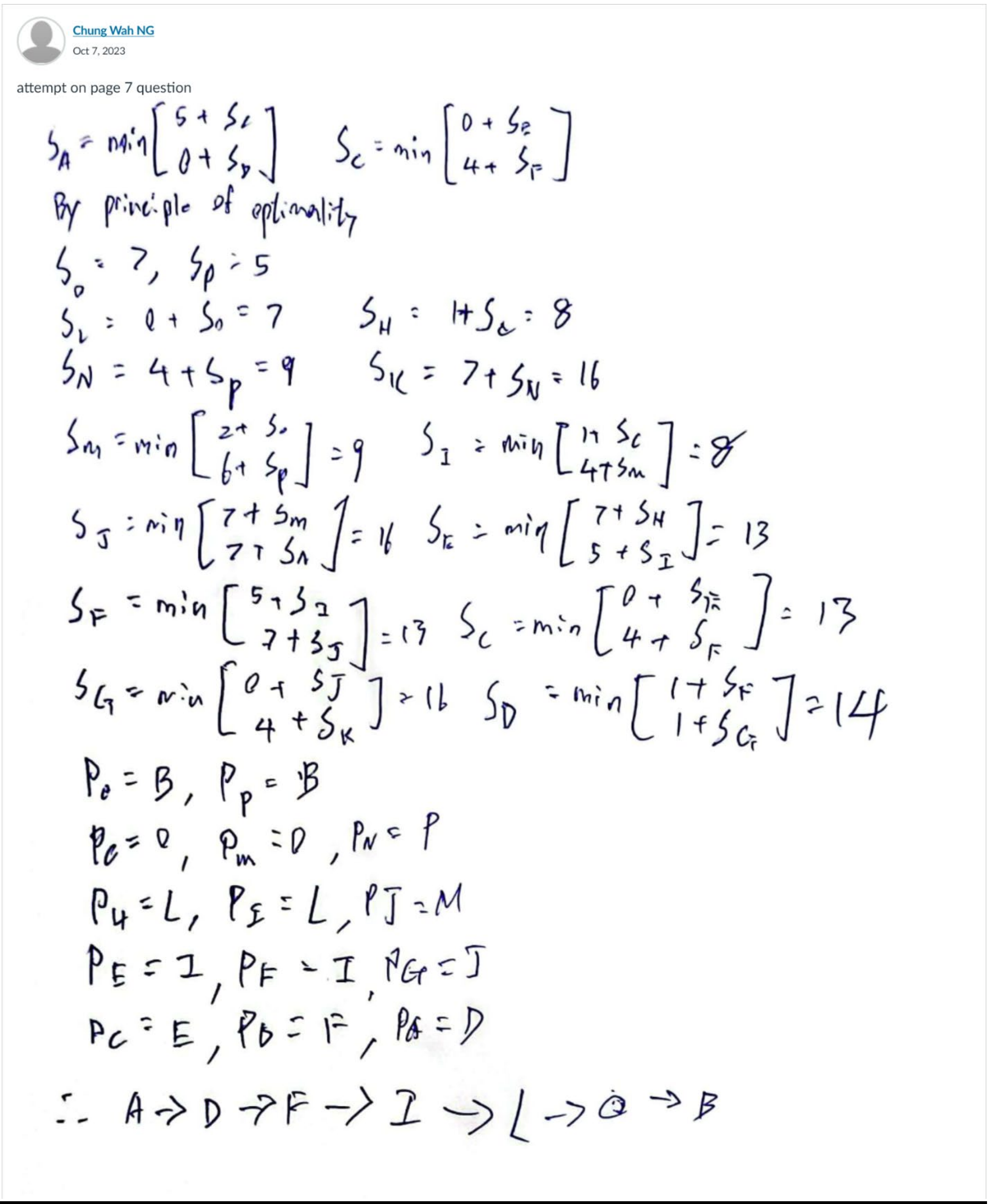




So just by looking at the last graph, we can immediately know the path is A,F,C,E. If familiar enough, you can even skip straight to the last graph.

Week 5 (from 1 Oct 2023 to 7 Oct 2023)

1. Attempted a question from the lecture notes.



Chung Wah NG

Oct 7, 2023

attempt on page 7 question

$S_A = \min\begin{bmatrix} 5 + S_C \\ 0 + S_D \end{bmatrix}$ $\quad S_C = \min\begin{bmatrix} 0 + S_E \\ 4 + S_F \end{bmatrix}$

By principle of optimality

$S_O = 7, \ S_P = 5$

$S_L = 0 + S_O = 7 \qquad S_H = 1 + S_L = 8$

$S_N = 4 + S_P = 9 \qquad S_K = 7 + S_N = 16$

$S_M = \min\begin{bmatrix} 2 + S_O \\ 6 + S_P \end{bmatrix} = 9 \qquad S_I = \min\begin{bmatrix} 1 + S_L \\ 4 + S_M \end{bmatrix} = 8$

$S_J = \min\begin{bmatrix} 7 + S_M \\ 7 + S_N \end{bmatrix} = 16 \qquad S_E = \min\begin{bmatrix} 7 + S_H \\ 5 + S_I \end{bmatrix} = 13$

$S_F = \min\begin{bmatrix} 5 + S_I \\ 7 + S_J \end{bmatrix} = 13 \qquad S_C = \min\begin{bmatrix} 0 + S_E \\ 4 + S_F \end{bmatrix} = 13$

$S_G = \min\begin{bmatrix} 0 + S_J \\ 4 + S_K \end{bmatrix} = 16 \qquad S_D = \min\begin{bmatrix} 1 + S_F \\ 1 + S_G \end{bmatrix} = 14$

$P_O = B, \ P_P = B$

$P_L = O, \ P_M = O, \ P_N = P$

$P_H = L, \ P_I = L, \ P_J = M$

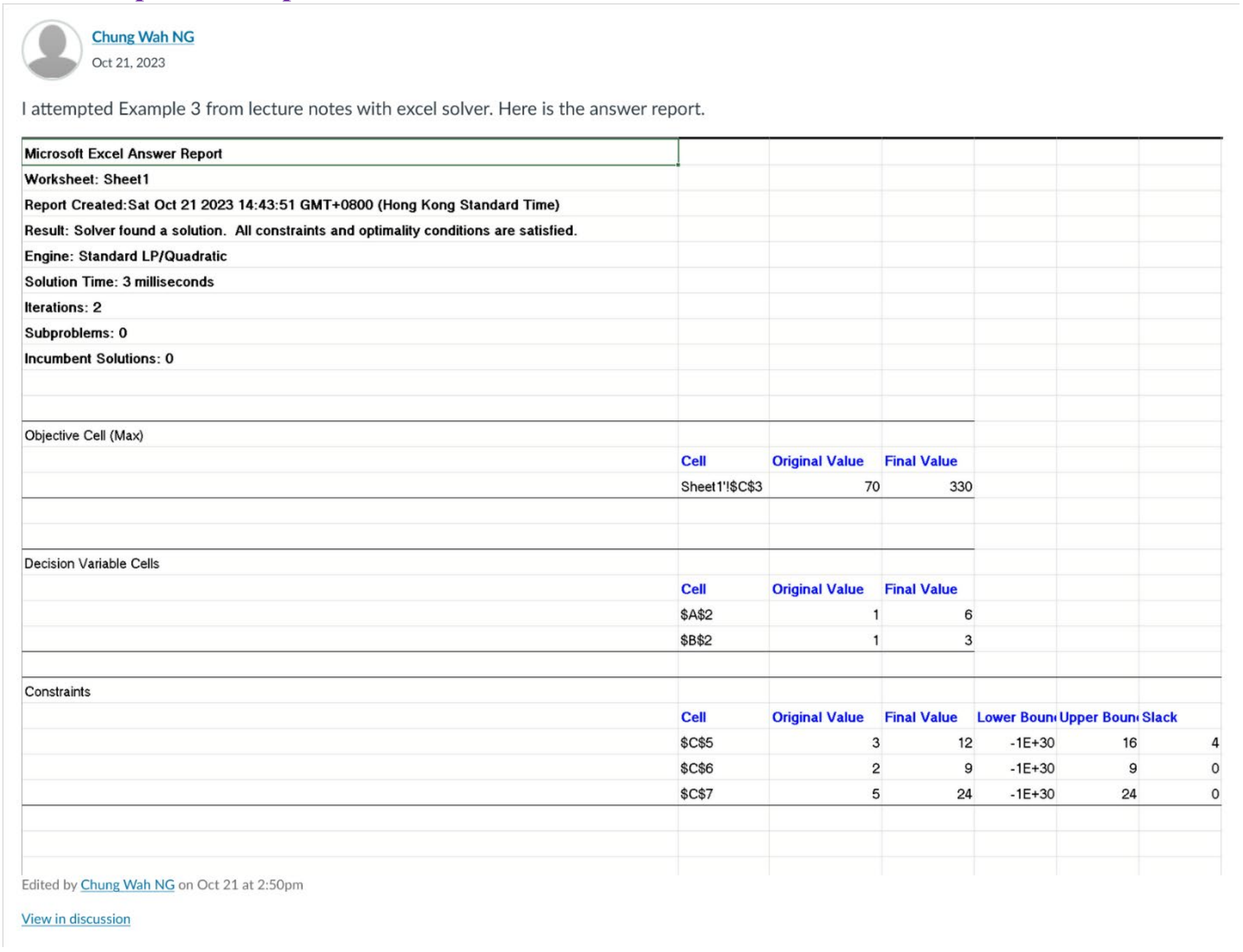
$P_E = I, \ P_F = I, \ P_G = J$

$P_C = E, \ P_D = F, \ P_A = D$

$\therefore A \to D \to F \to I \to L \to O \to B$

Week 7 (from 15 Oct 2023 to 21 Oct 2023)

1. Attempted Example 3 from the lecture notes with excel solver.



Chung Wah NG

Oct 21, 2023

I attempted Example 3 from lecture notes with excel solver. Here is the answer report.

Microsoft Excel Answer Report

Worksheet: Sheet1

Report Created:Sat Oct 21 2023 14:43:51 GMT+0800 (Hong Kong Standard Time)

Result: Solver found a solution. All constraints and optimality conditions are satisfied.

Engine: Standard LP/Quadratic

Solution Time: 3 milliseconds

Iterations: 2

Subproblems: 0

Incumbent Solutions: 0

Objective Cell (Max)

Cell	Original Value	Final Value
Sheet1'!$C$3	70	330

Decision Variable Cells

Cell	Original Value	Final Value
$A$2	1	6
$B$2	1	3

Constraints

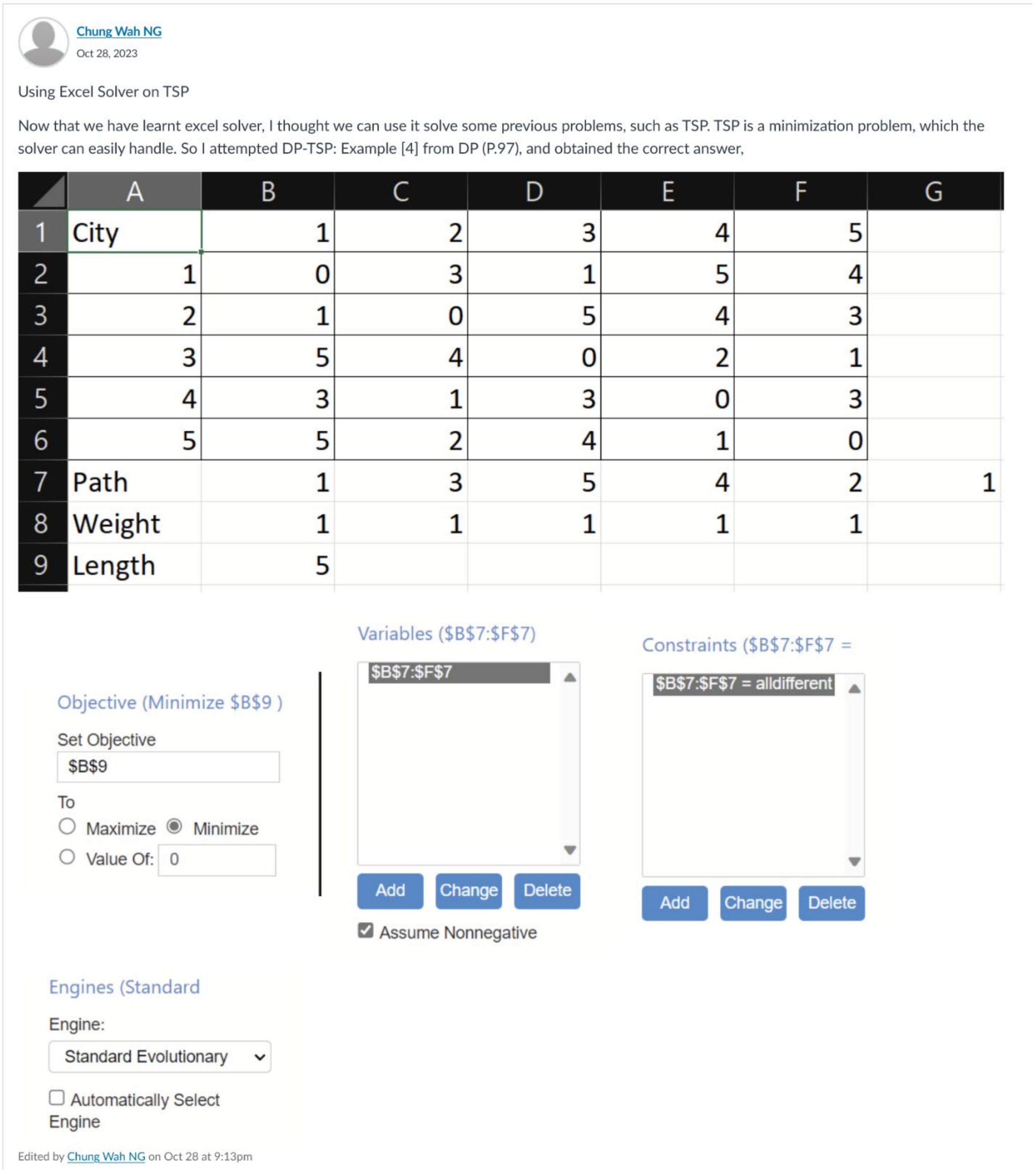
Cell	Original Value	Final Value	Lower Boun	Upper Boun	Slack
$C$5	3	12	-1E+30	16	4
$C$6	2	9	-1E+30	9	0
$C$7	5	24	-1E+30	24	0

Edited by Chung Wah NG on Oct 21 at 2:50pm

View in discussion

Week 8 (from 22 Oct 2023 to 28 Oct 2023)

1. Worked on using Excel Solver on TSP.



Chung Wah NG
Oct 28, 2023

Using Excel Solver on TSP

Now that we have learnt excel solver, I thought we can use it solve some previous problems, such as TSP. TSP is a minimization problem, which the solver can easily handle. So I attempted DP-TSP: Example [4] from DP (P.97), and obtained the correct answer,

	A	B	C	D	E	F	G
1	City	1	2	3	4	5	
2	1	0	3	1	5	4	
3	2	1	0	5	4	3	
4	3	5	4	0	2	1	
5	4	3	1	3	0	3	
6	5	5	2	4	1	0	
7	Path	1	3	5	4	2	1
8	Weight	1	1	1	1	1	
9	Length	5					

Objective (Minimize $B$9)

Set Objective: $B$9

To: ( ) Maximize [✓ Minimize] ( ) Value Of: 0

Variables ($B$7:$F$7)

$B$7:$F$7

Assume Nonnegative ✓

Constraints ($B$7:$F$7 =

$B$7:$F$7 = alldifferent

Engines (Standard

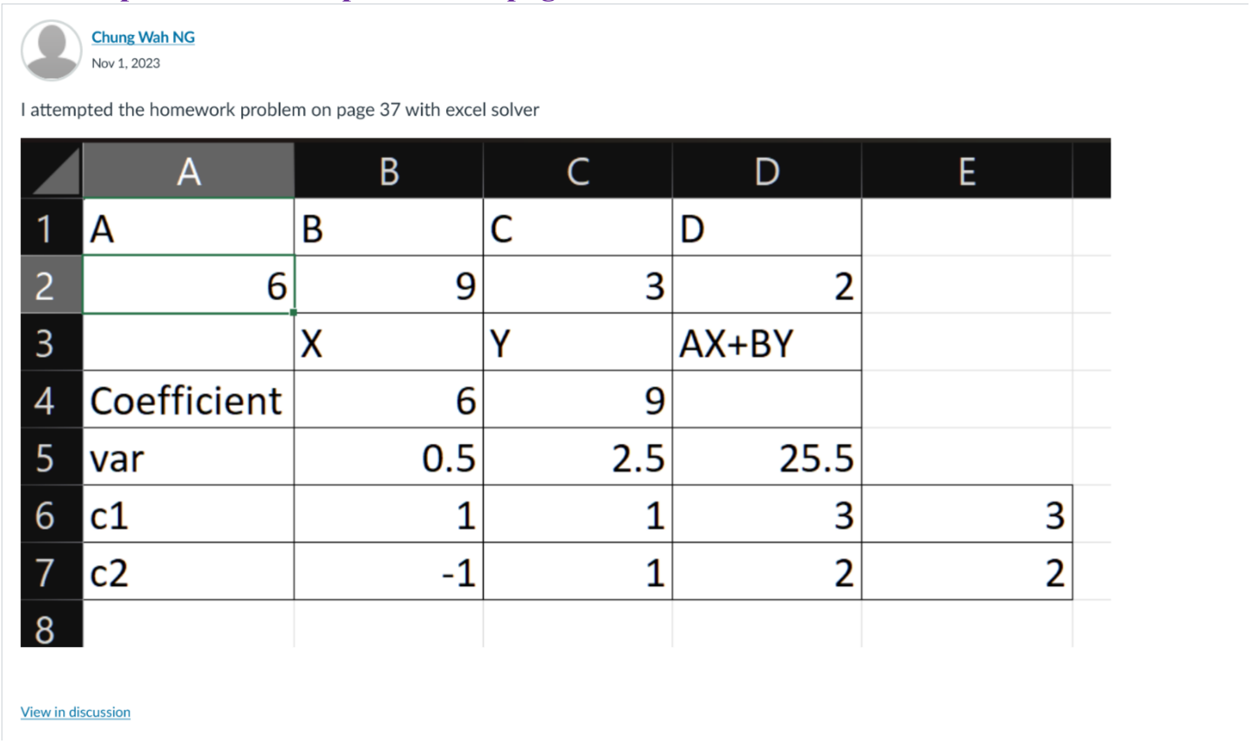
Engine: Standard Evolutionary

☐ Automatically Select Engine

Edited by Chung Wah NG on Oct 28 at 9:13pm

Week 9 (from 29 Oct 2023 to 4 Nov 2023)

1. Attempted homework problem on page 37



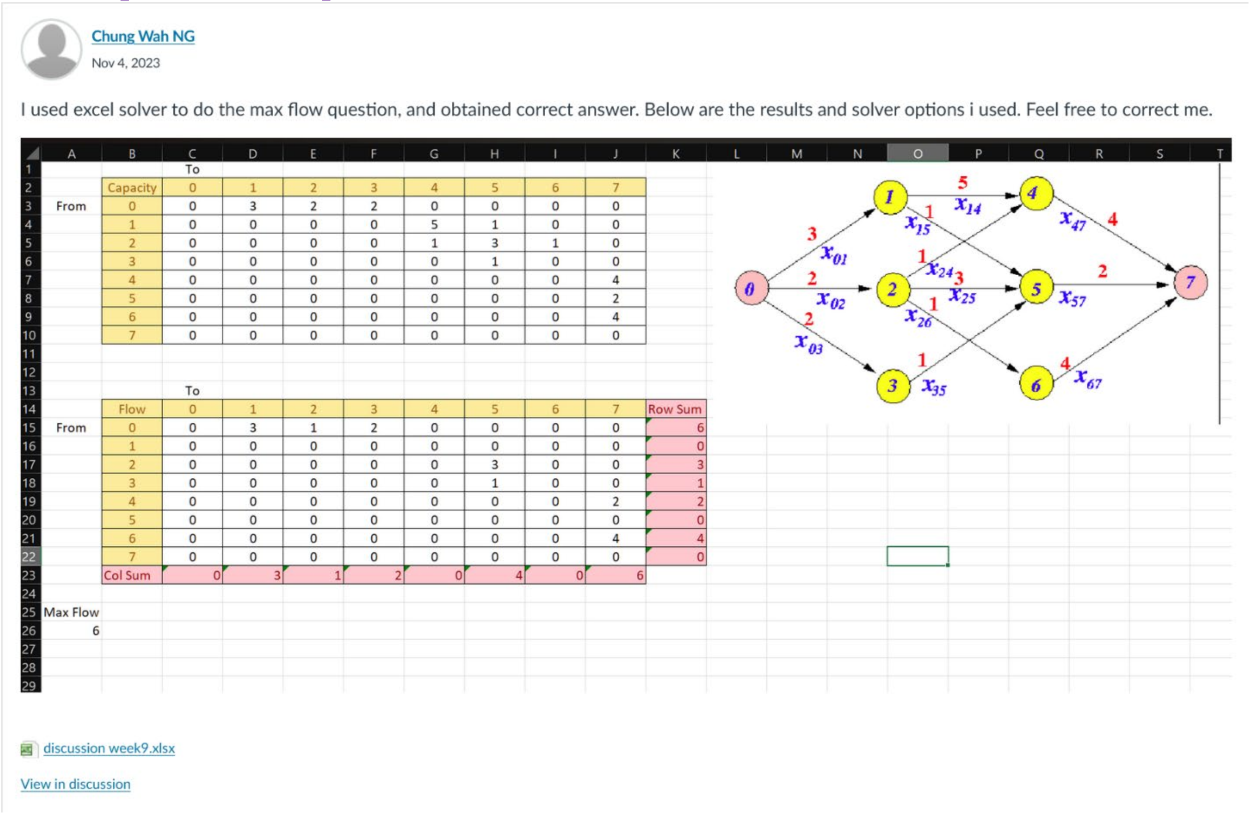
Chung Wah NG
Nov 1, 2023

I attempted the homework problem on page 37 with excel solver

	A	B	C	D	E
1	A	B	C	D	
2	6	9	3	2	
3		X	Y	AX+BY	
4	Coefficient	6	9		
5	var	0.5	2.5	25.5	
6	c1	1	1	3	3
7	c2	-1	1	2	2
8					

View in discussion

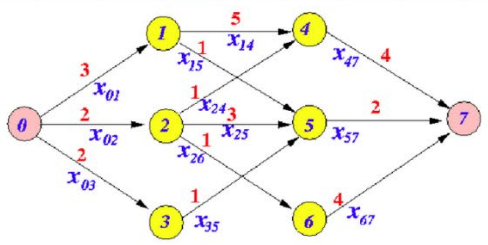
2. Attempted max flow problem with excel solver



Chung Wah NG
Nov 4, 2023

I used excel solver to do the max flow question, and obtained correct answer. Below are the results and solver options i used. Feel free to correct me.

	A	B	C	D	E	F	G	H	I	J	K
1			To								
2		Capacity	0	1	2	3	4	5	6	7	
3	From	0	0	3	2	2	0	0	0	0	
4		1	0	0	0	0	5	1	0	0	
5		2	0	0	0	0	1	3	1	0	
6		3	0	0	0	0	0	1	0	0	
7		4	0	0	0	0	0	0	0	4	
8		5	0	0	0	0	0	0	0	2	
9		6	0	0	0	0	0	0	0	4	
10		7	0	0	0	0	0	0	0	0	
11											
12											
13			To								
14		Flow	0	1	2	3	4	5	6	7	Row Sum
15	From	0	0	3	1	2	0	0	0	0	6
16		1	0	0	0	0	0	0	0	0	0
17		2	0	0	0	0	0	3	0	0	3
18		3	0	0	0	0	0	1	0	0	1
19		4	0	0	0	0	0	0	0	2	2
20		5	0	0	0	0	0	0	0	0	0
21		6	0	0	0	0	0	0	0	4	4
22		7	0	0	0	0	0	0	0	0	0
23		Col Sum	0	3	1	2	0	4	0	6	
24											
25	Max Flow										
26	6										

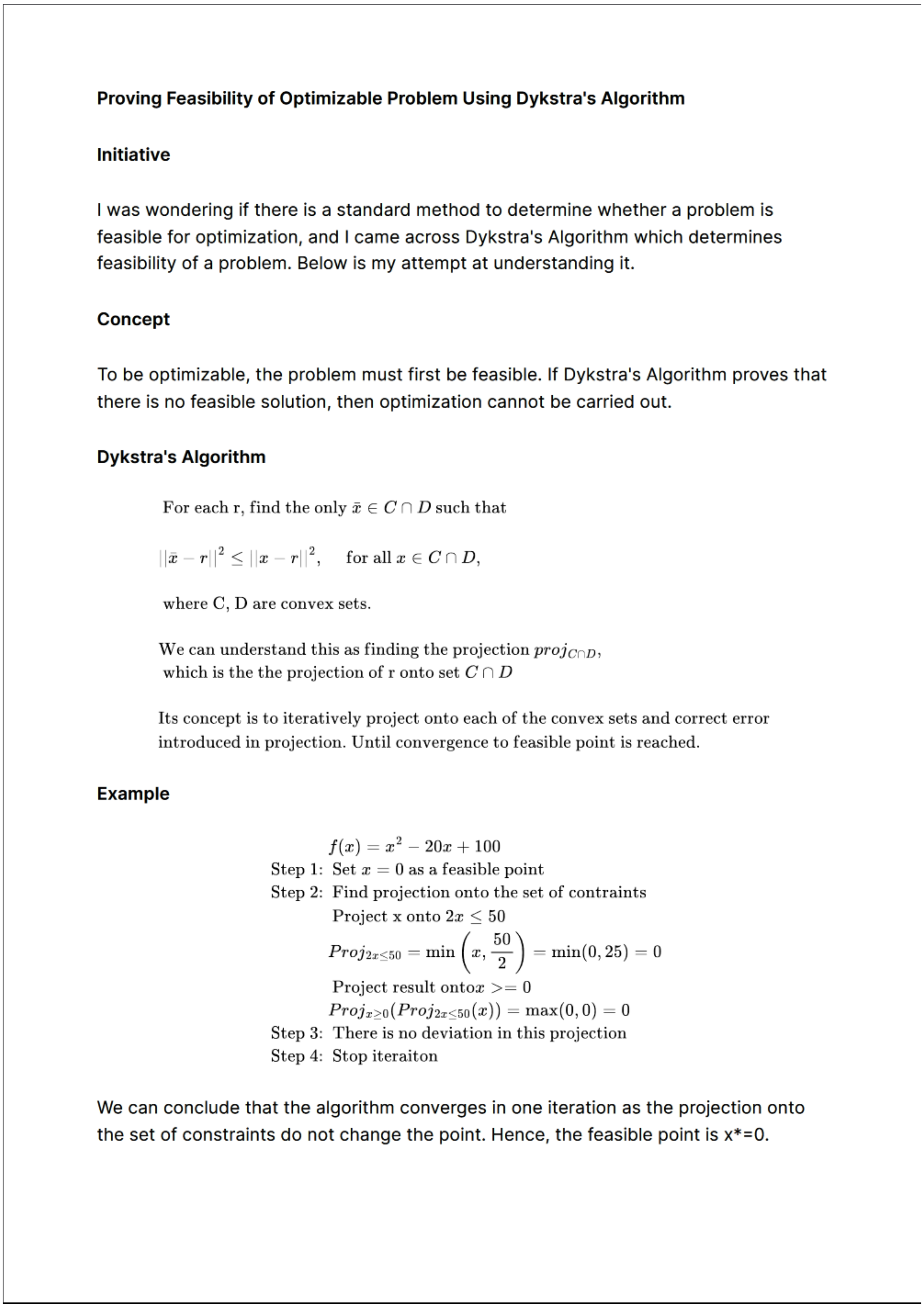


discussion week9.xlsx

View in discussion

Week 10 (from 5 Nov 2023 to 11 Nov 2023)

1. Researched on "Proving Feasibility of Optimizable Problem Using Dykstra's Algorthm"



Proving Feasibility of Optimizable Problem Using Dykstra's Algorithm

Initiative

I was wondering if there is a standard method to determine whether a problem is feasible for optimization, and I came across Dykstra's Algorithm which determines feasibility of a problem. Below is my attempt at understanding it.

Concept

To be optimizable, the problem must first be feasible. If Dykstra's Algorithm proves that there is no feasible solution, then optimization cannot be carried out.

Dykstra's Algorithm

For each r, find the only $\bar{x} \in C \cap D$ such that

$$||\bar{x} - r||^2 \leq ||x - r||^2, \quad \text{for all } x \in C \cap D,$$

where C, D are convex sets.

We can understand this as finding the projection $proj_{C \cap D}$,
which is the the projection of r onto set $C \cap D$

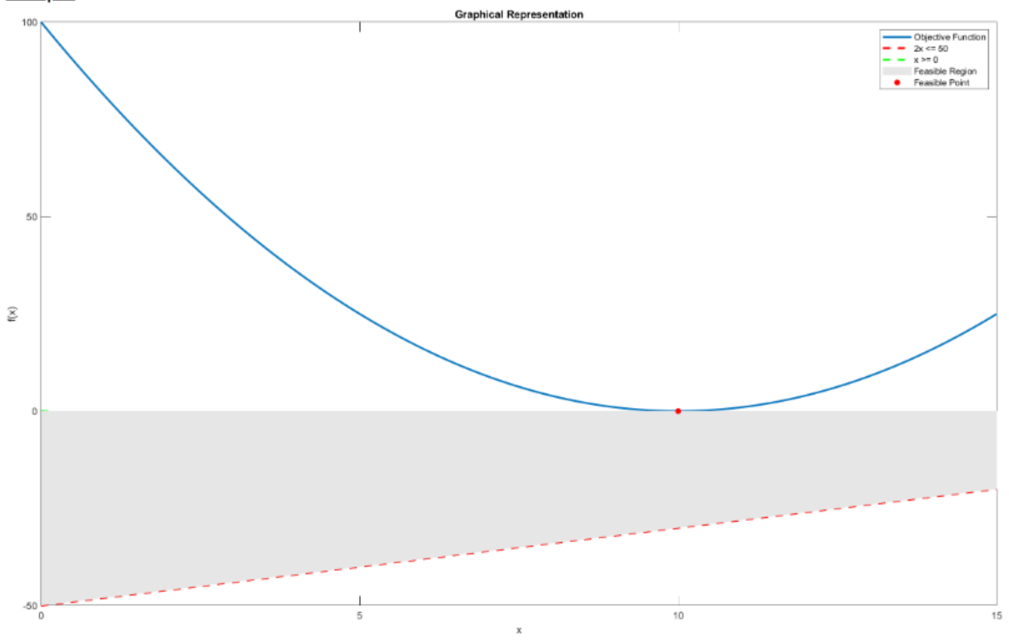
Its concept is to iteratively project onto each of the convex sets and correct error introduced in projection. Until convergence to feasible point is reached.

Example

$$\begin{aligned}
& f(x) = x^2 - 20x + 100 \\
\text{Step 1:}\ & \text{Set } x = 0 \text{ as a feasible point} \\
\text{Step 2:}\ & \text{Find projection onto the set of contraints} \\
& \text{Project x onto } 2x \leq 50 \\
& Proj_{2x \leq 50} = \min\left(x, \frac{50}{2}\right) = \min(0, 25) = 0 \\
& \text{Project result onto} x >= 0 \\
& Proj_{x \geq 0}(Proj_{2x \leq 50}(x)) = \max(0, 0) = 0 \\
\text{Step 3:}\ & \text{There is no deviation in this projection} \\
\text{Step 4:}\ & \text{Stop iteraiton}
\end{aligned}$$

We can conclude that the algorithm converges in one iteration as the projection onto the set of constraints do not change the point. Hence, the feasible point is x*=0.

Graph

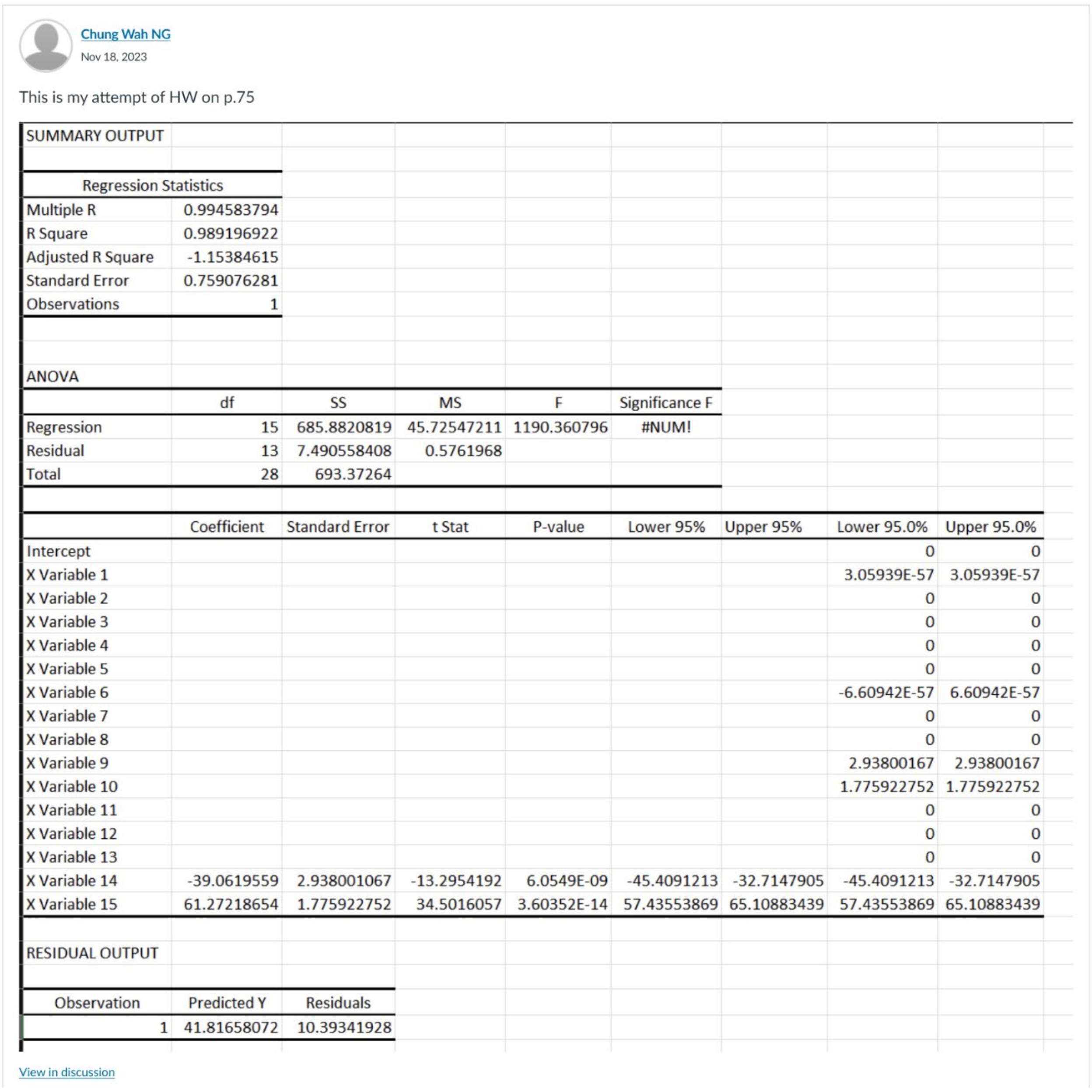


References

Dykstra's projection algorithm. (2023, August 22). Wikipedia, the free encyclopedia. Retrieved November 11, 2023, from https://en.wikipedia.org/wiki/Dykstra%27s_projection_algorithm

Week 11 (from 12 Nov 2023 to 18 Nov 2023)

1. Attempted the Homework on page 75



Chung Wah NG

Nov 18, 2023

This is my attempt of HW on p.75

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.994583794
R Square	0.989196922
Adjusted R Square	-1.15384615
Standard Error	0.759076281
Observations	1

ANOVA

	df	SS	MS	F	Significance F
Regression	15	685.8820819	45.72547211	1190.360796	#NUM!
Residual	13	7.490558408	0.5761968		
Total	28	693.37264			

	Coefficient	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0%
Intercept							0	0
X Variable 1							3.05939E-57	3.05939E-57
X Variable 2							0	0
X Variable 3							0	0
X Variable 4							0	0
X Variable 5							0	0
X Variable 6							-6.60942E-57	6.60942E-57
X Variable 7							0	0
X Variable 8							0	0
X Variable 9							2.93800167	2.93800167
X Variable 10							1.775922752	1.775922752
X Variable 11							0	0
X Variable 12							0	0
X Variable 13							0	0
X Variable 14	-39.0619559	2.938001067	-13.2954192	6.0549E-09	-45.4091213	-32.7147905	-45.4091213	-32.7147905
X Variable 15	61.27218654	1.775922752	34.5016057	3.60352E-14	57.43553869	65.10883439	57.43553869	65.10883439

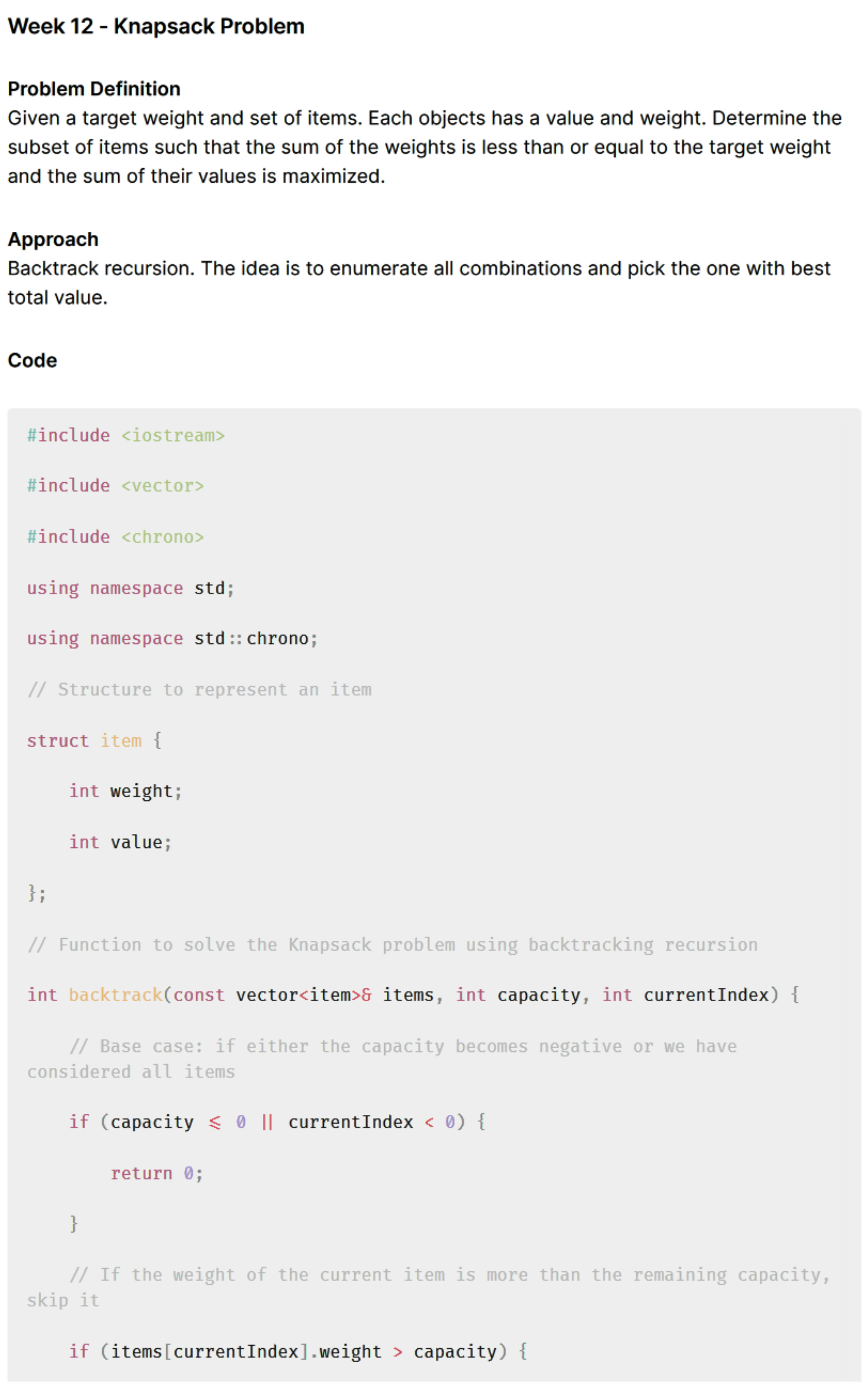
RESIDUAL OUTPUT

Observation	Predicted Y	Residuals
1	41.81658072	10.39341928

View in discussion

Week 12 (from 19 Nov 2023 to 25 Nov 2023)

1. Provided C++ code and Steps to solve Knapsack Problem.



Week 12 - Knapsack Problem

Problem Definition

Given a target weight and set of items. Each objects has a value and weight. Determine the subset of items such that the sum of the weights is less than or equal to the target weight and the sum of their values is maximized.

Approach

Backtrack recursion. The idea is to enumerate all combinations and pick the one with best total value.

Code

```cpp
#include <iostream>

#include <vector>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Structure to represent an item

struct item {

    int weight;

    int value;

};

// Function to solve the Knapsack problem using backtracking recursion

int backtrack(const vector<item>& items, int capacity, int currentIndex) {

    // Base case: if either the capacity becomes negative or we have
considered all items

    if (capacity <= 0 || currentIndex < 0) {

        return 0;

    }

    // If the weight of the current item is more than the remaining capacity,
skip it

    if (items[currentIndex].weight > capacity) {
        return backtrack(items, capacity, currentIndex - 1);

    }

    // Try both including and excluding the current item

    int includeCurrent = items[currentIndex].value +

                         backtrack(items, capacity -
items[currentIndex].weight, currentIndex - 1);

    int excludeCurrent = backtrack(items, capacity, currentIndex - 1);

    // Return the maximum of the two values

    return max(includeCurrent, excludeCurrent);

}

// Function to solve the 0/1 Knapsack problem and measure running time

int knapsack(const vector<item>& items, int capacity) {

    // Start the timer

    auto start_time = high_resolution_clock::now();

    // Start from the last item

    int currentIndex = items.size() - 1;

    int maxValue = backtrack(items, capacity, currentIndex);

    // Stop the timer

    auto stop_time = high_resolution_clock::now();

    auto duration = duration_cast<microseconds>(stop_time - start_time);

    // Print the running time in seconds and microseconds

    cout << "Running time: " << duration.count() / 1e6 << " seconds" << endl;

    // Print the theoretical big O notation

    cout << "Big O notation: O(2^n)" << endl;

    return maxValue;

}

int main() {

    vector<item> items = {{2, 10}, {5, 25}, {1, 7}, {3, 15}};
    int capacity = 7;

    int maxValue = knapsack(items, capacity);

    cout << "Maximum value that can be obtained: " << maxValue << endl;
    return 0;
}
```

Important Steps

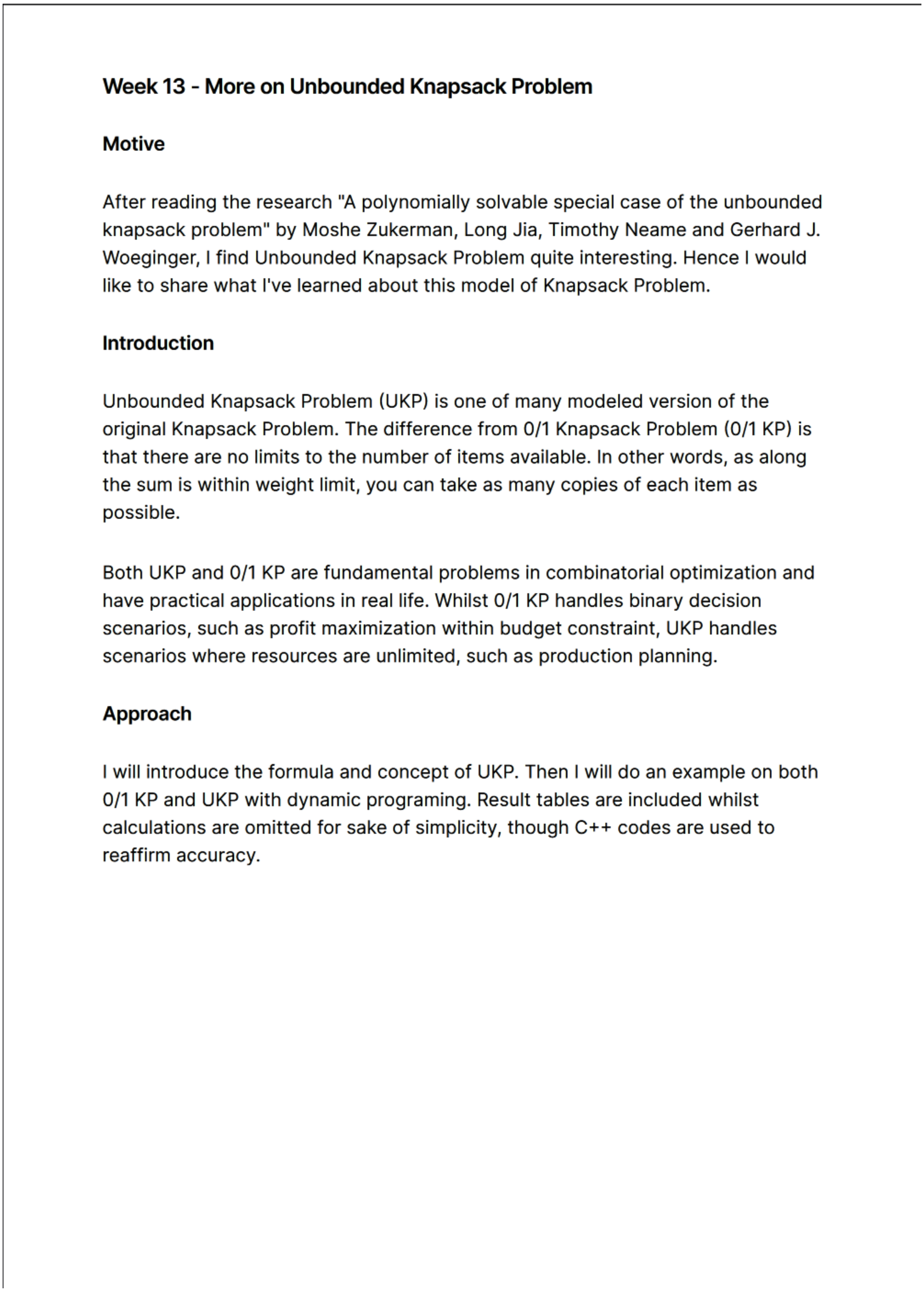
- **1: Define Item Structure**
  - Create a structure (`struct`) to represent items, with attributes for weight and value.
- **2: Recursive Function**
  - Implement a recursive helper function (`backtrack`) that explores combinations of items.
  - Base Case:
    - If remaining capacity is negative or all items are considered, return 0.
  - Decision:
    - If the weight of the current item exceeds remaining capacity, skip to the next item.
  - Explore:
    - Try both including and excluding the current item through recursive calls.
  - Return:
    - Return the maximum value obtained from the two possibilities.

References

(n.d.). Stanford University.
https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1228/section/section4/Knapsack%20Problem%20(Optional%20Section%20Slides).pdf

Week 13 (from 26 Nov 2023 to 2 Dec 2023)

1. **Expanded more on Unbounded Knapsack Problem.**



Week 13 - More on Unbounded Knapsack Problem

Motive

After reading the research "A polynomially solvable special case of the unbounded knapsack problem" by Moshe Zukerman, Long Jia, Timothy Neame and Gerhard J. Woeginger, I find Unbounded Knapsack Problem quite interesting. Hence I would like to share what I've learned about this model of Knapsack Problem.

Introduction

Unbounded Knapsack Problem (UKP) is one of many modeled version of the original Knapsack Problem. The difference from 0/1 Knapsack Problem (0/1 KP) is that there are no limits to the number of items available. In other words, as along the sum is within weight limit, you can take as many copies of each item as possible.

Both UKP and 0/1 KP are fundamental problems in combinatorial optimization and have practical applications in real life. Whilst 0/1 KP handles binary decision scenarios, such as profit maximization within budget constraint, UKP handles scenarios where resources are unlimited, such as production planning.

Approach

I will introduce the formula and concept of UKP. Then I will do an example on both 0/1 KP and UKP with dynamic programing. Result tables are included whilst calculations are omitted for sake of simplicity, though C++ codes are used to reaffirm accuracy.

Formulation

Let:
n be number of items
w_i be the weight of the i-th item
v_i be value of i-th item
C be the knapsack capacity
dp be a 2D array where $dp[i][j]$ is maximum value

Formula

$$dp[i][j] = \max(dp[i-1][j], dp[i][j-w_i] + v_i)$$

Formula Explanation

$dp[i-1][j]$: Maximum value obtained not including i-th item
$dp[i][j-w_i] + v_i$: Maxumum value obtained including i-th item.
Subtract weight w_i from current capacity j

Example

Using the example from lecture notes p.6, I'll solve both 0/1 KP and UKP with dynamic programming. Then use C++ to prove accuracy and compare the two KP's.

Sample Question

Given N = 5, W = 6. The table is as follow.

Item	Weight	Value
1	3	4
2	1	2
3	2	3
4	3	5
5	1	1

Case 1: 0/1 KP

Dynamic Programming

By applying the formula of 0/1 KP Dynamic Programming given in lecture notes:

$$V[i][w] = \max(V[i-1][w], V[i-1][w-w_i] + v_i)$$

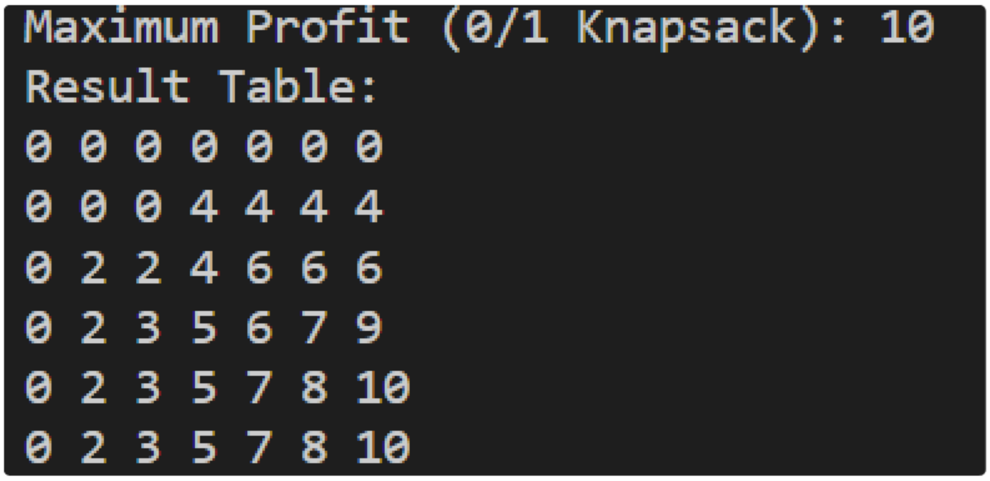
we can get the following table.

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

The answer is coherent with lecture notes.

Coding

I also used C++ codes to confirm accuracy. Codes are in attachment. Below is the output.



```
Maximum Profit (0/1 Knapsack): 10
Result Table:
0 0 0 0 0 0 0
0 0 0 4 4 4 4
0 2 2 4 6 6 6
0 2 3 5 6 7 9
0 2 3 5 7 8 10
0 2 3 5 7 8 10
```

Case 2: UKP

Dynamic Programming

By applying the formula defined above, we can get the following table.

	0	**1**	**2**	**3**	**4**	**5**	**6**
0	0	0	0	0	0	0	0
1	0	0	0	4	4	4	8
2	0	2	4	6	8	10	12
3	0	2	4	6	8	10	12
4	0	2	4	6	8	10	12
5	0	2	4	6	8	10	12

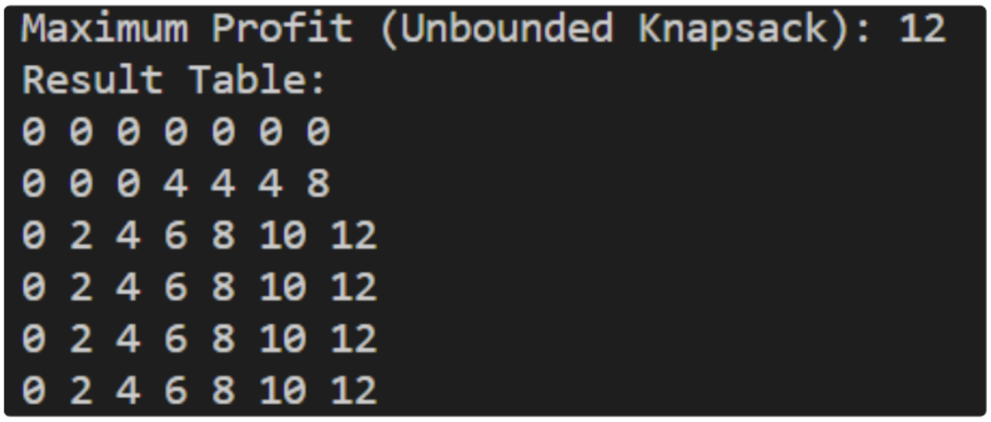
The answer is larger than that in lecture notes. Logically it makes sense, as there are less limitations.

Remark: Note that results of UKP may not always be larger than that of 0/1 KP.

Coding

To convert the code for 0/1 KP to solve UKP, there is one crucial change. Instead of considering $dp[i-1][j-wt[i-1]]$, it's modified to $dp[i][j-wt[i-1]]$ to represent the possibility of using the same item multiple times.

The new maximum value and code output is as follow:



```
Maximum Profit (Unbounded Knapsack): 12
Result Table:
0 0 0 0 0 0 0
0 0 0 4 4 4 8
0 2 4 6 8 10 12
0 2 4 6 8 10 12
0 2 4 6 8 10 12
0 2 4 6 8 10 12
```

Reference

M. Zukerman, L. Jia, T. D. Neame and G. J. Woeginger, “A polynomially Solvable special case of the unbounded knapsack problem”, Operations Research Letters, vol. 29, no. 1, August 2001, pp. 13-16.

ChatGPT, https://chat.openai.com/

Knapsack calculator. (n.d.). Augustine Aykara. https://augustineaykara.github.io/Knapsack-Calculator/

(n.d.). Stanford University. https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1228/section/section4/Knapsack%20Problem%20 (Optional%20Section%20Slides).pdf

0/1 knapsack problem. (2023, October 6). GeeksforGeeks. https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/

Unbounded knapsack (Repetition of items allowed). (2023, September 11). GeeksforGeeks. https://www.geeksforgeeks.org/unbounded-knapsack-repetition-items-allowed/