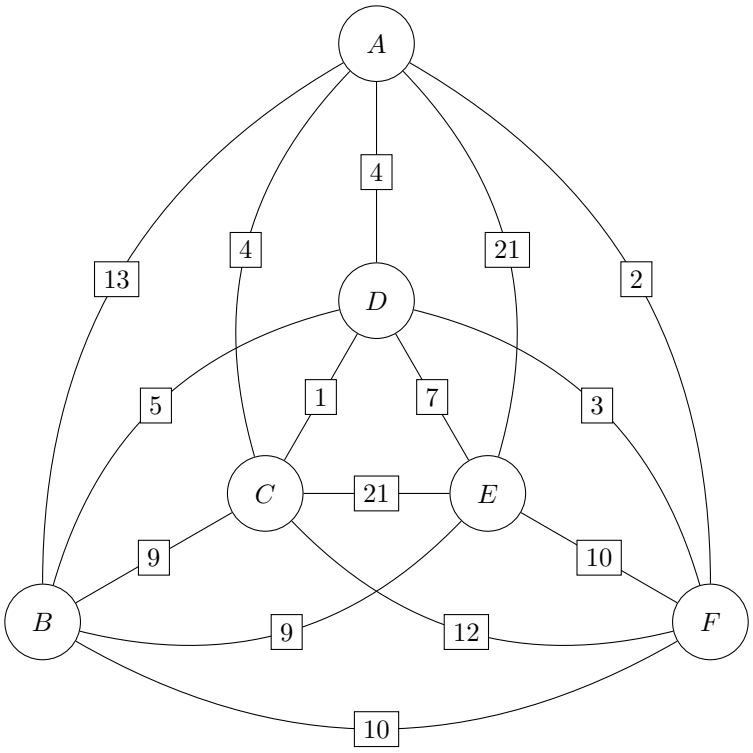
5CCS2FC2: Foundations of Computing II

Tutorial Sheet 9

Solutions

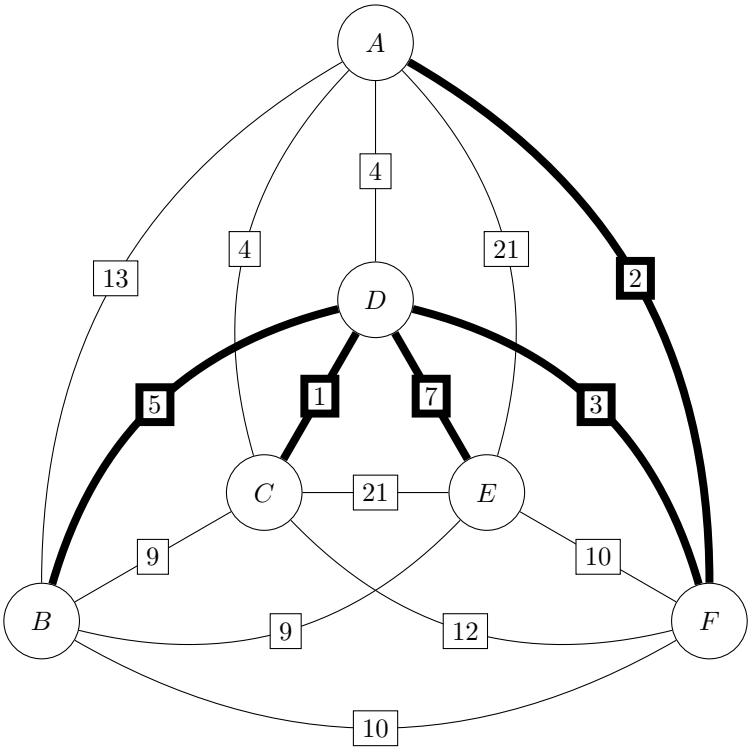
9.1 Consider the following complete weighted graph:



Apply the *2-opt optimisation algorithm* to find a local optimum traversal.

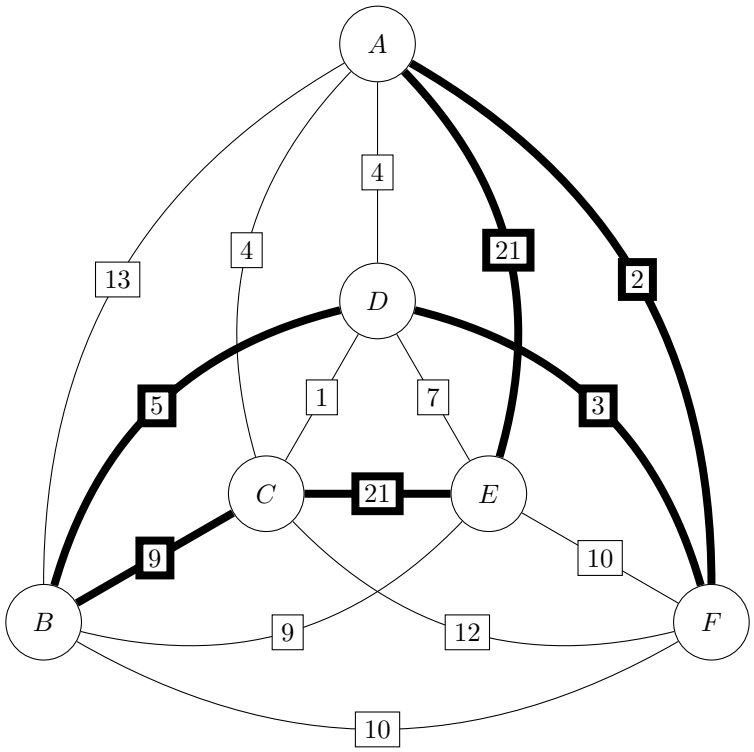
SOLUTION:

- The minimum spanning tree is given by



  and has a total weight of 18.

- The pre-order traversal of the minimum spanning tree gives



$$A \xrightarrow{2} F \xrightarrow{3} D \xrightarrow{5} B \xrightarrow{9} C \xrightarrow{21} E \xrightarrow{21} A$$

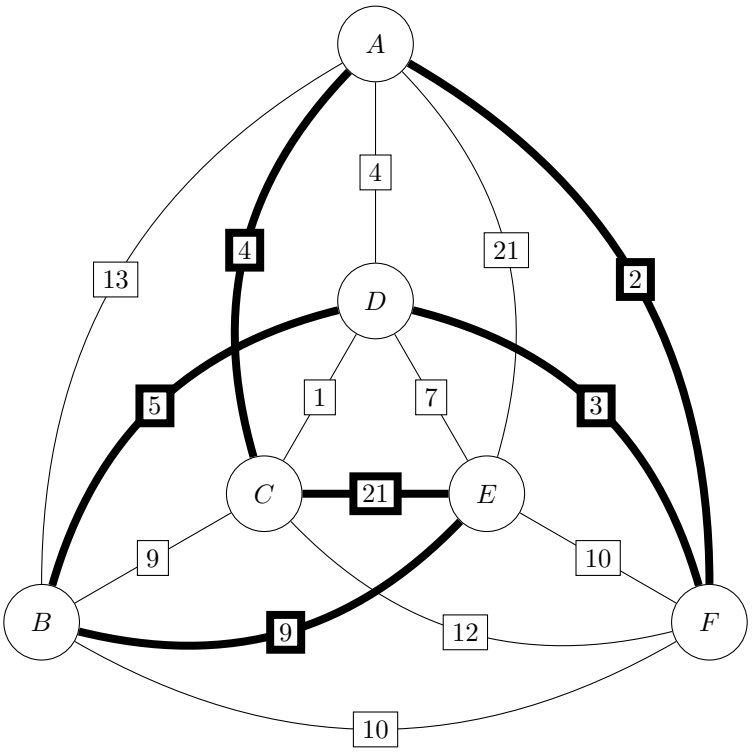
Total Weight : 61

(N.B. Note that the above graph does *not* satisfy the triangle inequality, so the weight of this initial walk *exceeds* twice the weight of the MST.)

- We can choose any pair of non-adjacent edges to apply the 2-opt swap move (swapping the edges if the swap is determined to result in a shorter path).

  A good heuristic would be to examine the non-adjacent edges with the highest combined weight. For example, the pair (A, E) and (B, C), result in a swap, since:

$$d(A,E) + d(B,C) \;=\; 30 \;>\; 13 \;=\; d(B,E) + d(A,C)$$

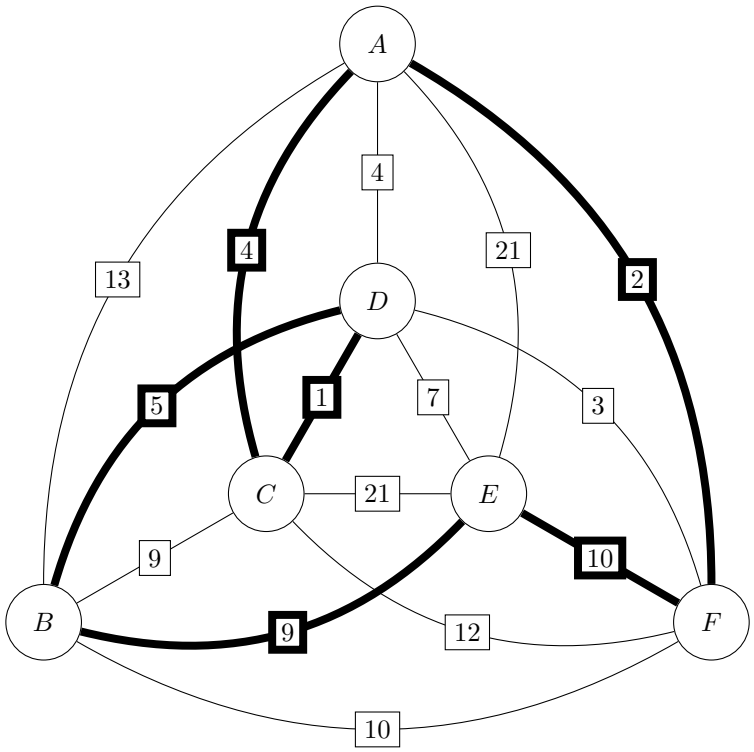


$$A \xrightarrow{2} F \xrightarrow{3} D \xrightarrow{5} B \xrightarrow{9} E \xrightarrow{21} C \xrightarrow{4} A$$

Total Weight : 44

- The next two largest non-adjacent adges are (C, E) and (D, F), with a combined weight of 24

$$d(C, E) + d(D, F) \;=\; 24 \;>\; 11 \;=\; d(C, D) + d(E, F)$$



$$A \xrightarrow{2} F \xrightarrow{10} E \xrightarrow{9} B \xrightarrow{5} D \xrightarrow{1} C \xrightarrow{4} A$$

Total Weight : 31

- No further applications of the 2-opt swap move can improve upon the length of this path, so the algorith terminates with the final path

  $$A \xrightarrow{2} F \xrightarrow{10} E \xrightarrow{9} B \xrightarrow{5} D \xrightarrow{1} C \xrightarrow{4} A$$

  whose total weight is 31.

  N.B. In fact, this a *global optimum* for this instance of TSP. (There are also 11 other global optima, each with a weight of 31.)

9.2 Consider the Greedy Knapsack algorithm from week 6:

Algorithm Greedy Knapsack

```
Sort item_list by value/weight ratio
for all item in item_list do
  if current_capacity + weight(item) < max_capacity then
    add item to knapsack
  end if
end for
```

Apply the Greedy Knapsack algorithm to each of the following Knapsack instances:

(i) Number of items: 4 Knapsack size: 12

item_list	A	B	C	D	Total
weight	7	1	1	11	20
value	3	1	3	11	18

(ii) Number of items: 4 Knapsack size:6

item_list	A	B	C	D	Total
weight	2	3	5	1	11
value	17	4	20	11	52

(iii) Number of items: 5 Knapsack size: 14

item_list	A	B	C	D	E	Total
weight	9	1	1	4	13	28
value	2	2	2	2	18	26

Find the *optimal* knapsack selection (by any method) and calculate the *approximation ratio* for each of the above instances.

SOLUTION:

(i) First we order the items by their *value*/*weight* ratio:

item_list	C	B	D	A
weight	1	1	11	7
value	3	1	11	3
value/weight	3	1	1	0.43

We then add the items into the knapsack from left-to-right, ommitting any that will not fit. We first add C and then B. The next in order is D but this will exceed the maximum capacity of the knapsack. However, there is still room at add A, without exceeding the maximum capacity.

However, the optimal selection would be D and C.

Greedy Solution	**Optimal Solution**
$\{C, B, A\}$	$\{D, C\}$
$value = 7$	$value = 14$

The Approximation ratio is therefore:

$$\text{Approximation Ratio} = 14/7 = 2$$

(ii) Ordered by *value/weight* ratio, we have

item_list	D	A	C	B
weight	1	2	5	3
value	11	17	20	4
value/weight	11	8.5	4	1.3

Applying the Greedy Knapsack algorithm we would first add D then A. We would omit C since adding this would exceed the maximum capacity of the knapsack. Finlly we can still add B without exceeding the maximum capacity.

This turns out that this is also the optimal selection!

Greedy Solution	**Optimal Solution**
$\{D, A, B\}$	$\{D, A, B\}$
$value = 32$	$value = 32$

The Approximation ratio is therefore:

$$\text{Approximation Ratio} = 32/32 = 1$$

(iii) Ordered by *value/weight* ratio, we have

`item_list`	B	C	E	D	A
weight	1	1	13	4	9
value	2	2	18	2	2
value/weight	2	2	1.38	0.5	0.2

Applying the Greedy Knapsack algorithm we would first add B then C. We would omit E since adding this would exceed the maximum capacity of the knapsack. We would add D, and then omit A, as this would again exceed the maximum capacity. However, the optimal selection would be to take B and E.

Greedy Solution	**Optimal Solution**
$\{B, C, E\}$	$\{B, E\}$
$value = 6$	$value = 20$

The Approximation ratio is therefore:

$$\text{Approximation Ratio} = 20/6 \approx 3.33$$

What we can say from these instances is that the Greedy Algorithm cannot be 3-approximable since we have seen instances whose approximation ratio exceeds 3.

9.3 *(tricky!)* What is the *worst case* approximation ratio for this implementation of the Greedy Knapsack algorithm?

SOLUTION: This implementation of the Greedy Knapsack algorithm is not a good approximation. Consider the following instance of Knapsack, consisting of just two items:

`item_list`	`light`	`heavy`	Total
weight	1	100	101
value	1	99	100

It is not possible to accomodate *both* items since their total exceeds the capacity of the Knapsack. However, the Greedy Algorithm is favour

`light` with the higher *value/weight* ratio. This is produce an 'aprroximate' solution with a value of 1, compared with the optimal solution of taking just `heavy` with value 99.

Greedy Solution
{`light`}
$value = 1$

Optimal Solution
{`heavy`}
$value = 99$

Hence the approximation ratio for this instance is 99. However, by varying the numbers we used we can force the approximation ratio to be as large as we like. For example, we could have

`item_list`	`light`	`heavy`	Total
weight	1	$K+1$	$K+2$
value	1	K	$K+1$

and a Knapsack capacity of $(K+1)$.

Here, the Greedy algorithm still favours `light` with *value/weight* ratio of 1, compared with the *value/weight* ratio of $K-1/K$ for B. However the optimal solution is to take only `heavy` whose value is K times that of `light`.

Greedy Solution
{`light`}
$value = 1$

Optimal Solution
{`heavy`}
$value = K$

Hence the approximation ration can be made to exceeds K, for any chosen value of K.