**Reduction from FAP to TSP**

**clicker question:** what’s the cost from CCGCAA to AATCTG?

A. 4  
B. 5  
C. 6  
D. 7

**Reduction from FAP to TSP**

*summary of steps (1): creating nodes and links*

- add one node per fragment
  - add two directed links, one in each direction, between each pair of nodes
- add two extra nodes, labeled *start* and *end*
  - add a link from *start* to each fragment node
  - add a link from each fragment node to *end*
  - add a link from *end* to *start*

**Reduction from FAP to TSP**

*next choose edge costs so that the cost of a tour equals the length of the corresponding assembly*
Reduction from FAP to TSP

next choose edge costs so that the cost of a tour equals the length of the corresponding assembly

Reduction from FAP to TSP

summary of steps: adding link costs

- the cost of the link from
  - fragments node R to fragment node S is the length of the R→S-overhang***
  - start to fragment node S is the length of S
  - fragment node S to end is 0
  - end to start is 0

*** the R→S-overhang is the part of S that does not overlap R

Reduction from FAP to TSP

the cheapest tour of the network corresponds to the shortest assembly of the fragments***

Key Concept: Reduction

Solve FAP using a TSP procedure:

1. remove substrands from FAP input f (does not change the shortest assembly)
2. build a TSP input (network) t for the FAP input
3. run procedure TSP procedure T on TSP input t
4. read FAP output (assembly) from solution (tour) found in Step 3.

*** this is only true if no fragment is a sub-fragment of another
**Key Concept: Reduction**

**In general:** To solve problem $P$ using a procedure $X$ for solving another problem $Q$:

1. modify $P$ input to reduce computational burden (= pre-processing, not always possible)
2. translate $P$ input to $Q$ input (need translation procedure = encoding, like FAP -> TSP)
3. run procedure $X$ on $Q$ input from Step 2.
4. translate $Q$ solution from Step 3 into $P$ solution (need translation procedure = decoding)

**Why are reductions useful?**

1. make it easier to build algorithms for solving new problems
2. if different problems $P_1, \ldots, P_n$ are all solved by reduction to some problem $Q$ (e.g., TSP), improvements in $Q$ algorithms directly give improvements in solving $P_1, \ldots, P_n$

**Why Reduce FAP to TSP?**

- fragment assembly is an important step in sequencing a genome
- fragment assembly can be solved using algorithms for a seemingly unrelated network problem - the TSP
- the TSP is a famous problem and years of effort have produced good (while not optimal) algorithms for this problem

**Selected RQs**

How is it possible that many "interesting and important problems" are somehow related the TSP?

(asked by several students from 2011W1)
More on the TSP

- the TSP is an example of the famous so-called "NP-hard" problems for which no provably fast algorithms are known
- an efficient, optimal algorithm for TSP would yield efficient, optimal algorithms not only for FAP but also for thousands of computational problems of practical importance
- the TSP is famous because it “represents” the computational complexity of thousands of problems

Selected RQs

- If we add new variables to the Traveling Salesman Problem like airflight schedules (so that the salesman not only has to take the cheapest route, but also take into account that he has to complete all flights in a certain amount of time and the flights only leave in certain schedules), would the problem be harder to answer (because there are more possibilities) or easier to answer (because the new variables restrict the possibilities)?
  
  (submitted by Iselle, 2011W1 Student)

Selected RQs

- The reading kept referring to efficient algorithms. What exactly does it mean?

- Don't computers do brute force calculations and see if any of the answers match the predetermined criteria? Isn't that why we have computers? To do calculations and remember things that we simply cannot?
Selected RQs

It is said that there is a very big award for the person who either finds an efficient algorithm that is guaranteed to find the cheapest tour in any instance of the problem [or prove that none exists]. If someone actually discovers this algorithm, how do we know this is the most efficient one; just like how we know that the algorithm we have now is the best one?

Similarly, how do algorithms know that all the different possibilities to find the cheapest route is checked/tried?

(submitted by Adrienne, 2011W1 Student)

• In the reading, the author makes comparisons between TSP and FAP. TSP appears to be an intractable problem. However, is FAP considered an intractable problem even though the algorithm we have derived in class and in the readings seems to be an efficient solution to all instances of FAP?

• According to the [reading], many important and interesting problems in the world are actually similar to the TSP. However, what are some of these important problems that would warrant [paying] someone $1,000,000 for [their efficient] solution?

• Can creating daily schedule be treated as another TSP in disguise? If events are the nodes what should be the "cost"?
Recall Learning Goals

You should be able to

- define the Fragment Assembly Problem (FAP), explain why algorithms for solving the problem are useful in sequencing the human genome and trace through the execution of a “greedy” algorithm for FAP
- define the Traveling Salesperson Problem (TSP), reduce an instance of the Fragment Assembly Problem to an instance of the TSP and explain why an efficient algorithm for optimally solving the TSP would be a major breakthrough

Resources

- here is a report on the use of the code for the TSP in deriving the mouse genome
- there is a 1M USD prize for finding an efficient and optimal solution for the TSP:
  - www.claymath.org/millennium/P_vs_NP/
  or for proving that there is no efficient and optimal algorithm
- see also the lecture by Vijaya Ramachandran at
  - www.claymath.org//Popular_Lectures/U_Texas/