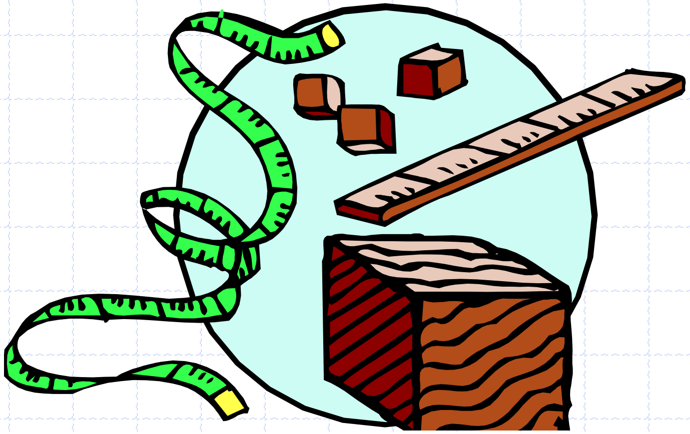
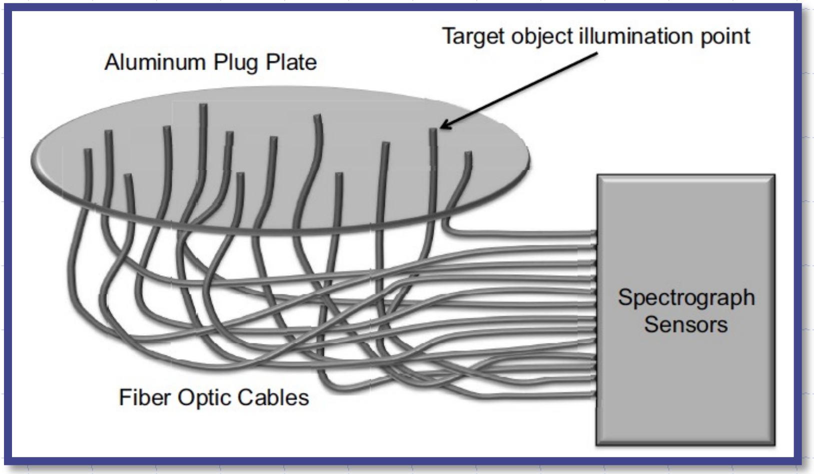
Approximation Algorithms

Michael T. Goodrich



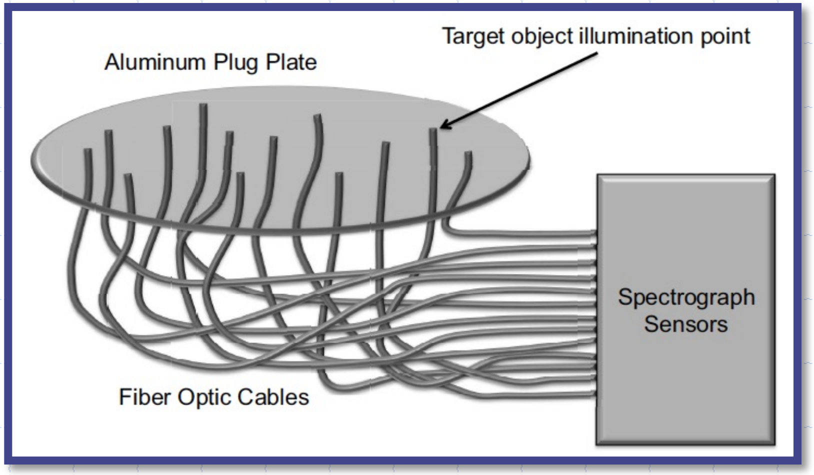
Applications

- One of the most time-consuming parts of astronomy involves collecting the light from the galaxy or star over a given period of time.
- To do this with a telescope, a large aluminum disk the size of the diameter of the telescope is used.
- This disk is placed in the focal plane of the telescope, so that the light from each stellar objects in an observation falls in a specific spot on the disk.
- The astronomers use robotic drilling equipment to drill a hole in each spot of interest and they insert a fiber-optic cable into each such hole and connect it to a spectrograph.



Application to TSP

- Drilling the holes in the fastest way is an instance of the **traveling salesperson problem (TSP)**.



- According to this formulation of TSP, each of the hole locations is a "city" and the time it takes to move a robot drill from one hole to another corresponds to the distance between the "citie" for these two holes.
- Thus, a minimum-distance tour of the cities that starts and ends at the resting position for the robot drill is one that will drill the holes the fastest.
- Unfortunately, TSP is NP-complete.
- So it would be ideal if we could at least approximate this problem.

Application to Set Cover

- Another optimization problem is to minimize the number of observations needed in order to collect the spectra of all the stellar objects of interest.
- In this case, we want to cover the map of objects with the minimum number of disks having the same diameter as the telescope.
- This optimization problem is an instance of the **set cover problem**.
- Each of the distinct sets of objects that can be included in a single observation is given as an input set and the optimization problem is to minimize the number of sets whose union includes all the objects of interest.
- This problem is also NP-complete, but it is a problem for which an approximation to the optimum might be sufficient.

Set Cover Example

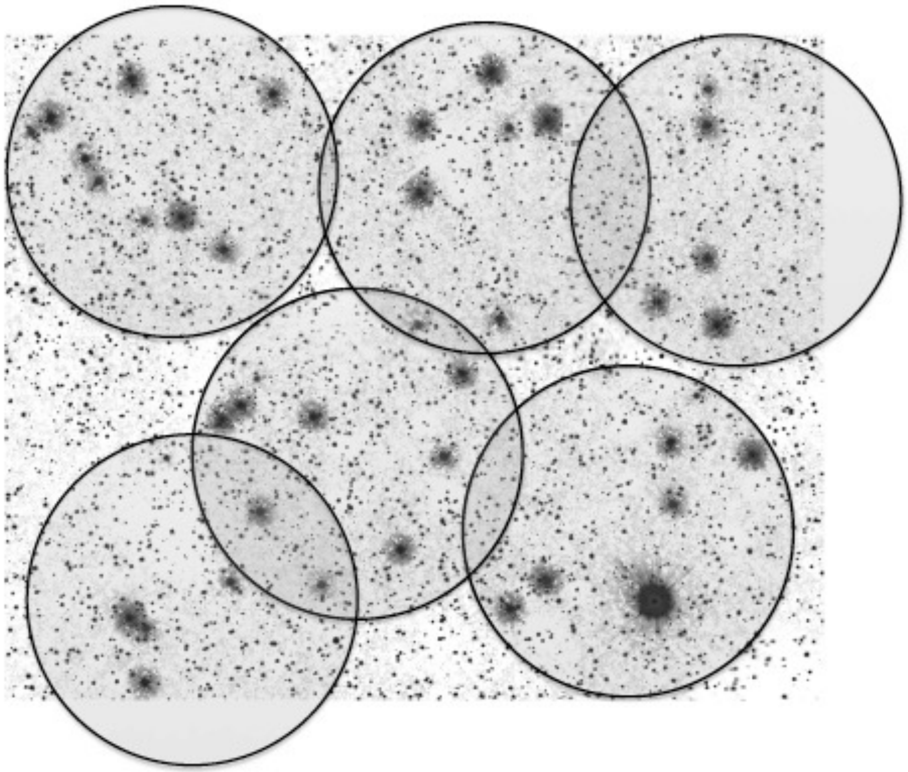
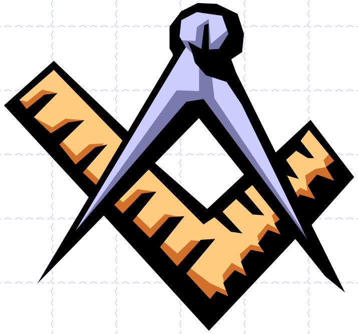


Figure 18.2: An example disk cover for a set of significant stellar objects (smaller objects are not included). Background image is from Omega Centauri, 2009. U.S. government image. Credit: NASA, ESA, and the Hubble SM4 ERO team.

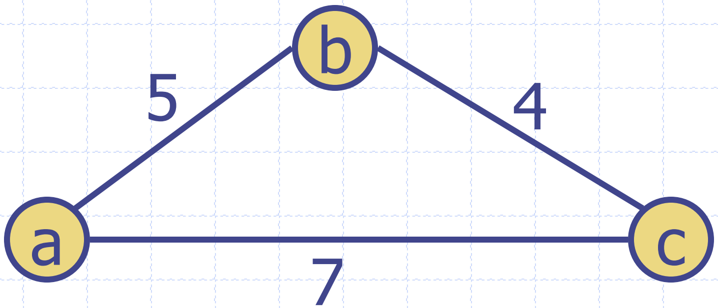
Approximation Ratios



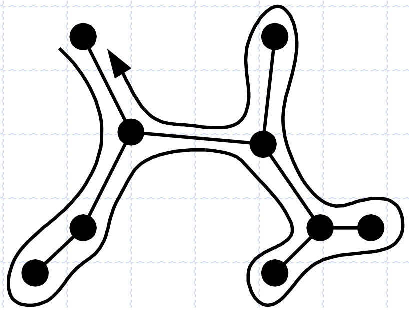
- Optimization Problems
  - We have some problem instance x that has many feasible "solutions".
  - We are trying to minimize (or maximize) some cost function c(S) for a "solution" S to x. For example,
    - Finding a minimum spanning tree of a graph
    - Finding a smallest vertex cover of a graph
    - Finding a smallest traveling salesperson tour in a graph
- An approximation produces a solution T
  - T is a **k-approximation** to the optimal solution OPT if $c(T)/c(OPT) \leq k$ (assuming a min. prob.; a maximization approximation would be the reverse)

Special Case of the Traveling Salesperson Problem

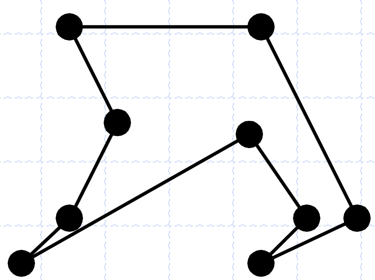
- OPT-TSP: Given a complete, weighted graph, find a cycle of minimum cost that visits each vertex.
  - OPT-TSP is NP-hard
  - Special case: edge weights satisfy the triangle inequality (which is common in many applications):
    - w(a,b) + w(b,c) $\geq$ w(a,c)



A 2-Approximation for TSP Special Case



Euler tour P of MST M



Output tour T

Algorithm ***TSPApprox***(***G***)

Input weighted complete graph ***G***, satisfying the triangle inequality

Output a TSP tour ***T*** for ***G***

M ← a minimum spanning tree for ***G***

P ← an Euler tour traversal of ***M***, starting at some vertex ***s***

T ← empty list

for each vertex ***v*** in ***P*** (in traversal order)

if this is ***v***' s first appearance in ***P*** **then**

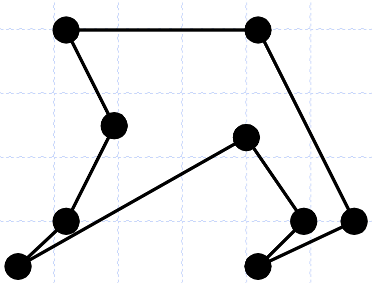
T.insertLast(***v***)

T.insertLast(***s***)

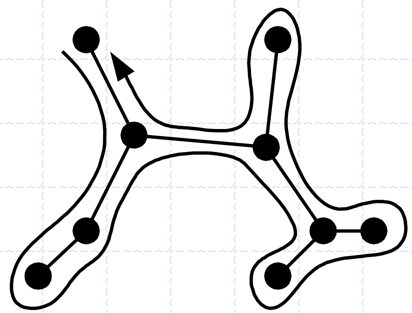
return ***T***

A 2-Approximation for TSP Special Case - Proof

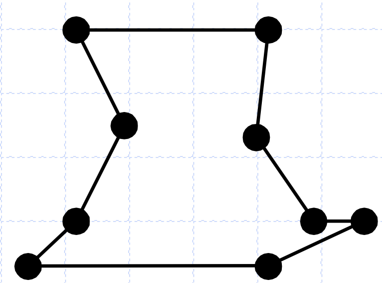
- The optimal tour is a spanning tour; hence $|M| \leq |OPT|$.
- The Euler tour P visits each edge of M twice; hence $|P|=2|M|$
- Each time we shortcut a vertex in the Euler Tour we will not increase the total length, by the triangle inequality ($w(a,b) + w(b,c) \geq w(a,c)$); hence, $|T| \leq |P|$.
- Therefore, $|T| \leq |P| = 2|M| \leq 2|OPT|$



Output tour T
(at most the cost of P)

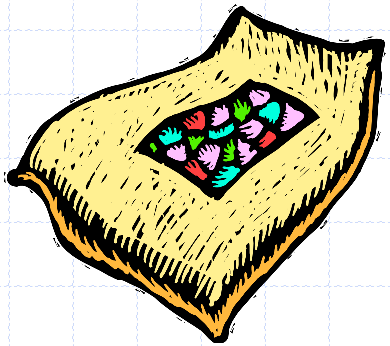


Euler tour P of MST M
(twice the cost of M)

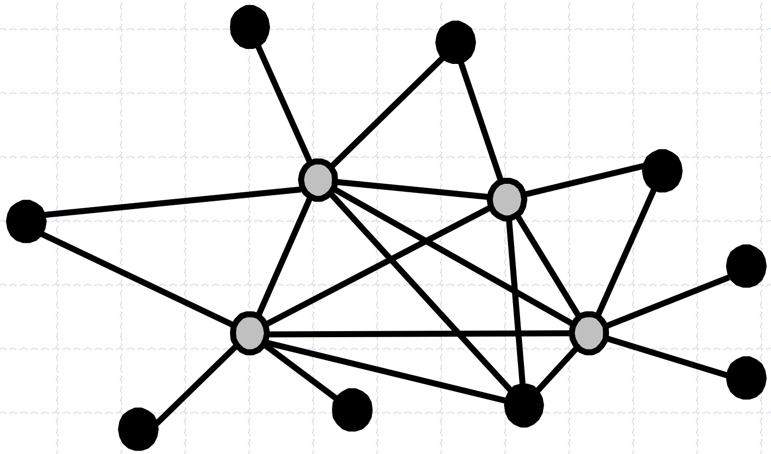


Optimal tour OPT
(at least the cost of MST M)

Vertex Cover



- A **vertex cover** of graph G=(V,E) is a subset W of V, such that, for every (a,b) in E, a is in W or b is in W.
- OPT-VERTEX-COVER: Given an graph G, find a vertex cover of G with smallest size.
- OPT-VERTEX-COVER is NP-hard.



A 2-Approximation for Vertex Cover

- Every chosen edge e has both ends in C
- But e must be covered by an optimal cover; hence, one end of e must be in OPT
- Thus, there is at most twice as many vertices in C as in OPT.
- That is, C is a 2-approx. of OPT
- Running time: O(n+m)

Algorithm VertexCoverApprox(G):
Input: A graph G
Output: A small vertex cover C for G

$C \leftarrow \emptyset$
while G still has edges **do**
  select an edge $e = (v, w)$ of G
  add vertices v and w to C
  for each edge f incident to v or w **do**
    remove f from G
return C

Set Cover (Greedy Algorithm)

- OPT-SET-COVER: Given a collection of m sets, find the smallest number of them whose union is the same as the whole collection of m sets?
  - OPT-SET-COVER is NP-hard
- Greedy approach produces an O(log n)-approximation algorithm.

Algorithm SetCoverApprox(S):
 Input: A collection S of sets $S_1, S_2, \ldots, S_m$ whose union is U
 Output: A small set cover C for S
 $C \leftarrow \emptyset$ // The set cover built so far
 $E \leftarrow \emptyset$ // The elements from U currently covered by C
 while $E \neq U$ **do**
  select a set S_i that has the maximum number of uncovered elements
  add S_i to C
  $E \leftarrow E \cup S_i$
 Return C.

Greedy Set Cover Analysis

- Consider the moment in our algorithm when a set S_j is added to C, and let k be the number of previously uncovered elements in S_j.
- We pay a total charge of 1 to add this set to C, so we charge each previously uncovered element i of S_j a charge of $c(i) = 1/k$.
- Thus, the total size of our cover is equal to the total charges made.
- To prove an approximation bound, we will consider the charges made to the elements in each subset S_j that belongs to an optimal cover, C ′. So, suppose that S_j belongs to C ′.
- Let us write $S_j = \{x_1, x_2, . . . , x_{nj}\}$ so that S_j's elements are listed in the order in which they are covered by our algorithm.

Greedy Set Cover Analysis, cont.

Now, consider the iteration in which x_1 is first covered. At that moment, S_j has not yet been selected; hence, whichever set is selected must have at least n_j uncovered elements. Thus, x_1 is charged at most $1/n_j$. So let us consider, then, the moment our algorithm charges an element x_l of S_j. In the worst case, we will have not yet chosen S_j (indeed, our algorithm may never choose this S_j). Whichever set is chosen in this iteration has, in the worst case, at least $n_j - l + 1$ uncovered elements; hence, x_l is charged at most $1/(n_j - l + 1)$. Therefore, the total amount charged to all the elements of S_j is at most

$$\sum_{l=1}^{n_j} \frac{1}{n_l - l + 1} = \sum_{l=1}^{n_j} \frac{1}{l},$$

which is the familiar ***harmonic number***, H_{n_i}. It is well known (for example, see the Appendix) that H_{n_j} is $O(\log n_j)$. Let $c(S_j)$ denote the total charges given to all the elements of a set S_j that belongs to the optimal cover C'. Our charging scheme implies that $c(S_j)$ is $O(\log n_j)$. Thus, summing over the sets of C', we obtain

$$\begin{aligned}\sum_{S_j \in C'} c(S_j) &\leq \sum_{S_j \in C'} b \log n_j \\ &\leq b|C'| \log n,\end{aligned}$$

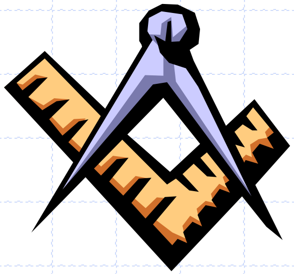
for some constant $b \geq 1$. But, since C' is a set cover,

$$\sum_{i \in U} c(i) \leq \sum_{S_j \in C'} c(S_j).$$

Therefore,

$$|C| \leq b|C'| \log n.$$

Polynomial-Time Approximation Schemes



- A problem L has a **polynomial-time approximation scheme (PTAS)** if it has a polynomial-time $(1+\varepsilon)$-approximation algorithm, for any fixed $\varepsilon > 0$ (this value can appear in the running time).
- 0/1 Knapsack has a PTAS, with a running time that is $O(n^3/\varepsilon)$.