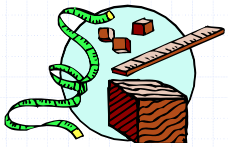
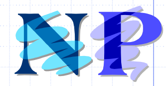
Approximation Algorithms

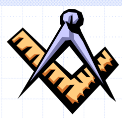


Outline and Reading



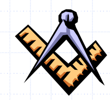
- Approximation Algorithms for NP-Complete Problems (§13.4)
  - Approximation ratios
  - Polynomial-Time Approximation Schemes (§13.4.1)
  - 2-Approximation for Vertex Cover (§13.4.2)
  - 2-Approximation for TSP special case (§13.4.3)
  - Log n-Approximation for Set Cover (§13.4.4)

Approximation Ratios



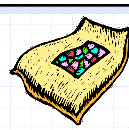
- Optimization Problems
  - We have some problem instance x that has many feasible "solutions".
  - We are trying to minimize (or maximize) some cost function c(S) for a "solution" S to x. For example,
    - Finding a minimum spanning tree of a graph
    - Finding a smallest vertex cover of a graph
    - 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)

Polynomial-Time Approximation Schemes

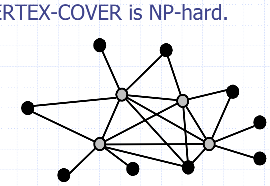


- A problem L has a **polynomial-time approximation scheme (PTAS)** if it has a polynomial-time $(1+\varepsilon)$-approximation algorithm, for any fixed $\varepsilon > 0$ (this value can appear in the running time).
- 0/1 Knapsack has a PTAS, with a running time that is $O(n^3/\varepsilon)$. Please see §13.4.1 in Goodrich-Tamassia for details.

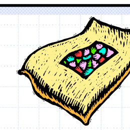
Vertex Cover



- 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

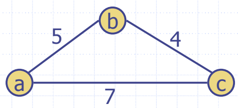


- Every chosen edge e has both ends in C
- But e must be covered by an optimal cover; hence, one end of e must be in OPT
- Thus, there is at most twice as many vertices in C as in OPT.
- That is, C is a 2-approx. of OPT
- Running time: O(m)

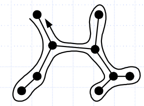
```
Algorithm VertexCoverApprox(G)
  Input graph G
  Output a vertex cover C for G
  C ← empty set
  H ← G
  while H has edges
    e ← H.removeEdge(H.anEdge())
    v ← H.origin(e)
    w ← H.destination(e)
    C.add(v)
    C.add(w)
    for each f incident to v or w
      H.removeEdge(f)
  return C
```

Special Case of the Traveling Salesperson Problem

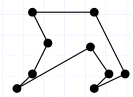
- OPT-TSP: Given a complete, weighted graph, find a cycle of minimum cost that visits each vertex.
  - OPT-TSP is NP-hard
  - Special case: edge weights satisfy the triangle inequality (which is common in many applications):
    - w(a,b) + w(b,c) $\geq$ w(a,c)



A 2-Approximation for TSP Special Case



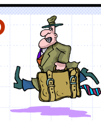
Euler tour P of MST M



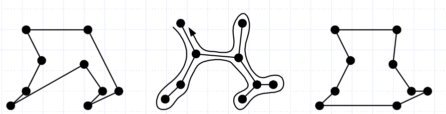
Output tour T

Algorithm *TSPApprox*(*G*)
  Input weighted complete graph G, satisfying the triangle inequality
  Output a TSP tour T for G
  $M \leftarrow$ a minimum spanning tree for G
  $P \leftarrow$ an Euler tour traversal of M, starting at some vertex s
  $T \leftarrow$ 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

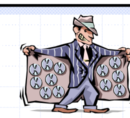


- The optimal tour is a spanning tour; hence |M|$\leq$|OPT|.
- The Euler tour P visits each edge of M twice; hence |P|=2|M|
- Each time we shortcut a vertex in the Euler Tour we will not increase the total length, by the triangle inequality (w(a,b) + w(b,c) $\geq$ w(a,c)); hence, |T|$\leq$|P|.
- Therefore,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

Set Cover



- 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. See §13.4.4 for details.

Algorithm *SetCoverApprox*(*G*)
  Input a collection of sets $S_1 \ldots S_m$
  Output a subcollection C with same union
  $F \leftarrow \{S_1, S_2, \ldots, S_m\}$
  $C \leftarrow$ empty set
  $U \leftarrow$ union of $S_1 \ldots S_m$
  while U is not empty
    $S_i \leftarrow$ set in F with most elements in U
    $F.remove(S_i)$
    $C.add(S_i)$
    Remove all elements in S_i from U
  return C