Outline

Announcements: PA3 is online, PA2 performance report due now

Today:

- Travelling Salesman Problem (TSP)
- Branch and Bound and Backtracking [Preiss 14.2]
- $k$-approximation and $k$-opt
Graph Algorithm Questions

See airline flight graph

Describe an algorithm to determine:

- all non-stop flights from JFK \(O( )\)
- if any non-stop flights from JFK exist \(O( )\)
- greatest distance covered by a non-stop flight from JFK \(O( )\)

Associate a price with each flight, describe an algorithm to determine:

- best “value” non-stop flight from JFK (max distance/price) \(O( )\)
- best “value” tour (non-stop or connecting flights) from JFK \(O( )\)
Graph Algorithm Questions

Describe an algorithm to determine:

- if a flight from JFK to SFO exists ($O(\cdot)$)
- minimal cost (distance or price) from JFK to SFO ($O(\cdot)$)

Suppose numbers represent cost (in billions USD) to build high-speed rail, describe an algorithm to determine least cost construction, such that any city can be reached from any other city ($O(\cdot)$)

Suppose you are planning a family reunion. Your family is spread out all over the US and you are paying for their travel. Describe an algorithm to find the city to host the reunion that minimizes total travel cost ($O(\cdot)$)
Hamiltonian Cycles and the Travelling Salesman Problem (TSP)

For the flight routes given in the airline flight graph is there a tour that takes us to each city exactly once and then takes us back to the starting city?

Such a tour is called a Hamiltonian Cycle, and a graph containing a Hamiltonian cycle is called Hamiltonian

Given a weighted graph, the Travelling Salesman Problem asks to find a Hamiltonion cycle with minimal weight (or of weight less than a given bound)
Example: the Travelling Professor Problem

Fig. 10.2.10 (J&S) lists some cities in India with the approximate road distances in miles

I’ve never been to India. I’d like to cover all of these cities in 10 days, going 450 miles a day

Is there a tour that would take me to all of the cities in less than 4500 miles?

I’ll be flying in and out of Chennai
A Tour of India

C
0

CN
650
A Tour of India

C 0

CN 650

CNA 1135
A Tour of India

C
0

CNA
1135

CNAK
1885

CN
650

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

C
0

CN
650

CNA
1135

CNAK
1885

CNK
1320

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

C
0

CN
650

CNA
1135

CNAK
1885

CNK
1320

CNKA
2070

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

C 0

CN 650

CNA 1135

CNAK 1885

CNK 1320

CNKA 2070

CNKAD 2190

CNKADS 2665

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India

Sugih Jamin (jamin@eecs.umich.edu)
A Tour of India
Finding a Tour for the Prof

If we avoid the long route from Chennai to Kolkata, we can try: Chennai-Nagpur-Kolkata-Agra-Delhi-Jaisalmer-Jaipur-Mumbai-Chennai for a total distance of 4525 miles

Or perhaps we should try to avoid the expensive routes to Nagpur from Chennai and Kolkata: Chennai-Kolkata-Agra-Nagpur-Jaipur-Delhi-Jaisalmer-Mumbai-Chennai for a total distance of 4905 miles

Chennai-Kolkata-Nagpur-Agra-Jaipur-Delhi-Jaisalmer-Mumbai-Chennai is only 4405 miles: tour found!

In general, how does one find a Hamiltonian cycle in a graph?
Problem Space vs. Solution Space

While you search for a solution in the **problem space**, you are generating “states” in the **solution space**

The “states” in the solution space form a search tree, with each internal node being a partial solution and the leaves potential solutions

For example:

- **problem space**: find a TSP tour starting from Chennai
- **solution space**: each step you take forms a partial path that is a unique “state” in the solution search tree
Solution Space Search

When we talk of searching for a solution, we’re talking about searching the solution space/search tree

Design patterns:

- brute force
- greedy, usually involving heuristics
- divide-and-conquer, usually involving recursion
- amortized
- randomized
- branch-and-bound and backtracking
- approximation
- local search
- dynamic programming
- steepest ascent hill climbing
- simulated annealing
- taboo search
- etc.

Sugih Jamin (jamin@eecs.umich.edu)
Finding TSP by Brute-Force (BF) Method

First find a Hamiltonian cycle using a randomized method

Then look for a tour shorter than the random one
Finding Hamiltonian Cycle by Randomized Method

Find a Hamiltonian cycle in $G$ by selecting random permutations of the nodes and sum up the weighted path lengths

What is the worst-case running time of the algorithm?

class Tour {
    int n; // number of cities
    int c[MAXCITIES];
    int l; // tour length
}

Tour hamiltonian(G)
{
    found = false; tour.n = n = |V|;
    while (!found) {
        for (i = 0 to n-1) {
            tour.c[i] = random_pick(V);
            V -= tour.c[i];
        }
        if ((c[n-1], c[0]) in E) {
            tour.l = weight(c[n-1], c[0]);
            found = true;
        } else { found = false; }
        for (i = 0 to n-1 && found) {
            if ((c[i], c[i+1]) in E) {
                tour.l += weight(c[i], c[i+1]);
            } else { found = false; }
        }
    }
    return tour;
}
Finding TSP by Brute-Force (BF) Method

Find a random Hamiltonian cycle, then find a shorter tour

Assume non-existent edges have weight $\infty$

What is the running time of this algorithm?

```
Tour tsp()
{
    best_tour = hamiltonian(G);
    tsp = { 1, [1], 0 };
    BFtsp(&tsp, &best_tour);
    return best_tour;
}

BFtsp(Tour *tsp, *best)
{
    n = tsp->n;
    if (n < best->n) {
        for each j neighbor of tsp->c[n-1] in (best->c - tsp->c) {
            // generate the full path to leaf
            ntsp = *tsp;
            ntsp.l += weight(ntsp.c[n-1], j);
            ntsp.c[n] = j;
            ntsp.n++;
            BFtsp(&ntsp, best);
        }
    } else {
        tsp->l +=
            weight(tsp->c[n-1], tsp->c[0]);
        if (tsp->l < best->l) *best = *tsp;
    }
}
```

Sugih Jamin (jamin@eecs.umich.edu)
Brute Force

When is Brute Force useful?

- no smarter algorithms are in sight (they may or may not exist)
- to evaluate correctness of heuristics (e.g., greedy)
- when smart algorithms are very complicated and results need to be verified
  - implement a simple brute-force algorithm
  - see if smarter alg. produces correct solution

Often relies on **enumeration**: produce all possible **permutations** of $N$ items (e.g., cities)

Takes $O(N \cdot N!)$ for TSP; $O(N \cdot 2^N)$ for coin change
Branch-and-Bound and Backtracking

Still brute force, but instead of following a solution branch all the way to the leaf, stop the enumeration once a constraint is violated

For example: find me a tour of 10 cities in 2500 miles
Tour of India (BnB)

Sugih Jamin (jamin@eecs.umich.edu)
Branch-and-Bound and Backtracking

**Branch**: enumerate all possible next steps from current partial solution

**Bound**: if a partial solution violates some constraint, e.g., an upper bound on cost, drop/prune the branch (don’t follow it further)

**Backtracking**: once a branched is pruned, move back to the previous partial solution and try another branch (depth-first branch-and-bound)

Branch-and-bound can also be done without backtracking if breadth-first search is done on the solution space
Finding TSP by Naive Branch-and-Bound (NBnB) Method

Assume non-existent edges have weight $\infty$

What is the worst-case running time of this algorithm?

Tour
tsp()
{
    best_tour = hamiltonian(G);
    tsp = { 1, [1], 0 };
    NBnBtsp(&tsp, &best_tour);
    return best_tour;
}

NBnBtsp(Tour *tsp, *best) {
    n = tsp->n;
    if (n < best->n) {
        for each j neighbor of tsp->c[n-1] in (best->c - tsp->c) {
            ntsp = *tsp;
            ntsp.l += weight(ntsp.c[n-1], j);
            if (ntsp.l < best->l) {
                ntsp.c[n] = j;
                ntsp->n++;
                NBnBtsp(&ntsp, best);
            } // otherwise bound and backtrack
        }
    } else {
        tsp->l += weight(tsp->c[n-1], tsp->c[0]);
        if (tsp->l < best->l) *best = *tsp;
    }

}
BnB Efficiency

Two factors effect the efficiency of BnB:

1. how soon (how high up in the solution search tree) can you prune away a partial solution: the sooner you can prune, the less time you need to spend on that branch

2. for optimization problem (e.g., finding the min cost tour) the tightness of your initial bound (e.g., the length of your random hamiltonian cycle) could be a huge factor in performance; so it may be worth spending extra effort to compute better bounds

Sugih Jamin (jamin@eecs.umich.edu)
About PA3

Tasks:
1. Input map (10%)
2. BnB (25%)
3. 2-Approx (25%)
4. $k$-Opt (20%)
5. Benchmarking and integration (20%)

Finding tighter bound for BnB is where you’d probably spent most of your time in PA3

Assume implementation of any given algorithm/heuristics has been optimized from a systems perspective (i.e., there will be no point given to implementing adjacency list instead of adjacency matrix)

Autograded, timing posted
More About PA3

Allowed to search the Internet for potential solutions. However:

- Do not consult previous terms’ solutions to this programming assignment
- **MUST** cite sources (or solution will **not** be graded)
- Don’t go off tangent reading more and more minute incremental improvements
- Improvements outside BnB won’t be accepted, in particular, the following improvements will not be accepted:
  - random walks
  - simulated annealing
  - taboo search
  - Lin-Kernighan
- Why? So that you won’t go off tangent spending too much time. Save them for EECS 477
Euler Cycle

An Euler cycle in a graph $G$ is a path from $v$ to $v$ that contains all of the edges and all of the vertices of $G$, but no repeated edges.

The Königsberg-bridge problem: the first graph theoretic problem!

A graph $G$ has an Euler cycle iff $G$ is connected and the degree of every vertex is even.

Given an MST, to construct the Euler cycle:

- consider each edge as 2 edges in separate directions, so each vertex has an even number of edges
- do a pre-order traversal
- running time: there’s one edge per node, a tree has at most $|V| - 1$ edges, each non-leaf node is visited 3 times: $O(|V|)$
A 2-Approximation for a Special Case of TSP

If distances satisfy the **triangle inequality**, i.e., for edges $(u, v), (v, w), (u, w)$ in $G: C(u, v) + C(v, w) \geq C(u, w)$, and $G$ is a **complete graph**, we can build a 2-approximate TSP:

1. construct an MST of the graph (using Prim’s for example)
2. construct an Euler-cycle of the MST
3. replace all edges $(u, v)$ and $(v, w)$ for which $v$ is visited more than once with edge $(u, w)$

$k$-approximate means that the result is off by a factor of $k$ from the optimal

Running time:
$O(|E| \log |V|)$ for MST + $O(|V|)$ for Euler = $O(|E| \log |V|)$
2-Approximate Proof

Let:
- $M$ be the MST
- $E$ the Euler tour, visit every edge in $M$ twice (once per direction)
- $T$ the non-optimal TSP obtained from $E$
- and $T'$ the optimal TSP minus an edge, which must be a spanning tree

Then:
- $C(T) \leq C(E)$ by triangle inequality
  
  \[ C(u, w) \leq C(u, v) + C(v, w) \]
- $C(E) = 2C(M)$
- $C(M) \leq C(T')$ by definition of MST

So $C(T) \leq 2C(T')$
$k$-Opt

$k$-Opt is a greedy, local search heuristic:

- start with an initial tour, e.g., the randomly generated one or the MST
- consider two cities $v_i$ and $v_{i+1}$ or any two cities $u$ and $v$
- check if visiting $v_{i+1}$ before visiting $v_i$ decreases total tour length
- repeat with different $v_i$ until there is no further improvement (or up to time limit)

The above is 2-opt, you can also consider other $k$ values.
Tour of India ($k$-Opt)

$2$-Opt

Sugih Jamin (jamin@eecs.umich.edu)