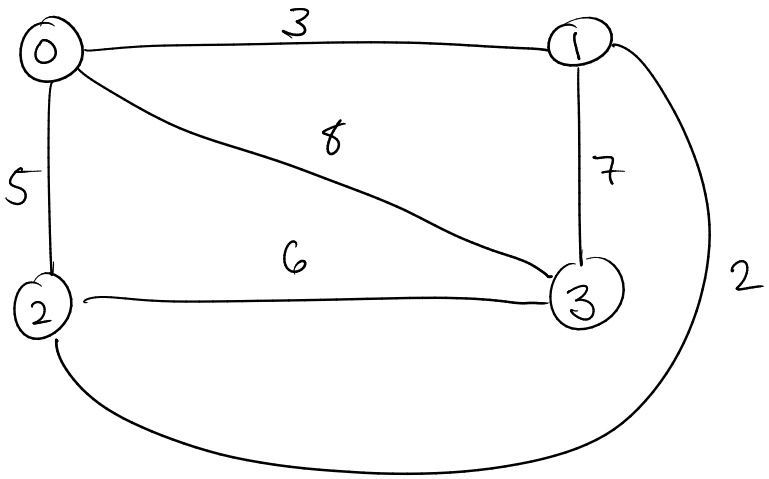
Math 343, Lecture 21

① A better bound for the traveling salesman problem

Def Given an instance of the traveling salesman problem let M be

eg Continuing the example from last class we have the following graph and cost



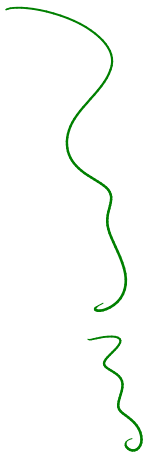
Then

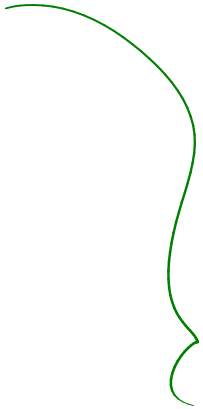
$M =$

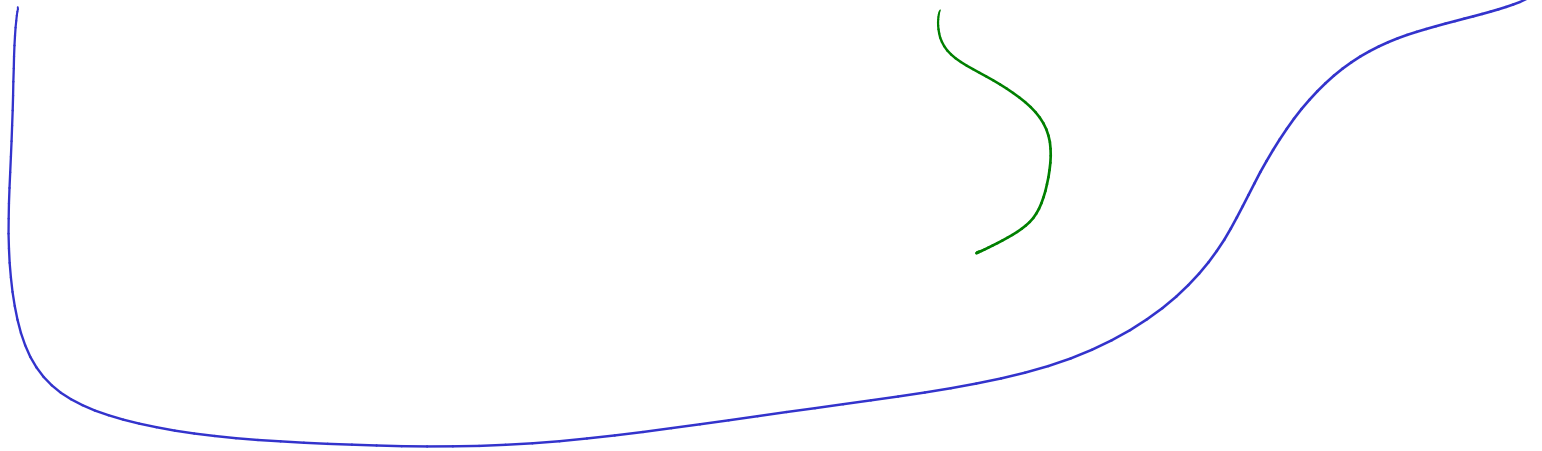
Def the cost of M is the output of the following algorithm

Algorithm Reduce

input M (mxm matrix)







eg

$$M = \begin{bmatrix} \infty & 3 & 5 & 8 \\ 3 & \infty & 2 & 7 \\ 5 & 2 & \infty & 6 \\ 8 & 7 & 6 & \infty \end{bmatrix}$$ the cost of M is:

prop let M be the matrix for an instance of the traveling salesman problem then the cost of M is $\leq$ the cost of any Hamiltonian cycle of the problem

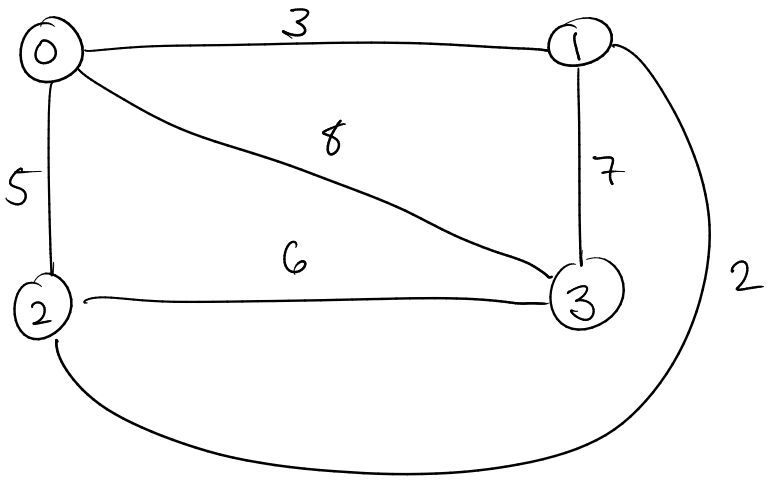
pf let G be the graph of the problem
let $X = [x_0, \ldots, x_{n-1}]$ be a Hamiltonian cycle of G
let $x_n = x_0$. Then

$\text{cost}(X) =$

let $r_i =$
$c_j =$

eg in the running eg

$G =$



we calculated

$$\text{Reduce}(M) = 18$$

The Hamiltonian cycles are

This gives us a bounding function for the Traveling salesman problem

Algorithm ReduceBound

Input M (mxm matrix), $X = [x_0, .., x_{m-1}]$, V vertices of G

If m=n

$M'(0,0) = \infty$

$Y = V - \{x_0, .., x_{m-1}\}$

return ans

And so we have the following backtracking algorithm

Algorithm ReduceTravelingSalesman

```
global X, OptC, OptX, C_l, M
input l
if l = n
    C = cost(x_0, x_1) + ... + cost(x_{n-2}, x_{n-1}) + cost(x_{n-1}, x_0)
    if C < OptC
        OptC = C
        OptX = [x_0, ..., x_{n-1}]
if l = 0
    C_l = {0}
else if l = 1
    C_l = {1, ..., n-1}
else
    C_l = C_{l-1} \ {x_{l-1}}
B = ReduceBound(M, X, V)
for x ∈ C_l
    if B ≥ OptC
        return
    ReduceTravelingSalesman(l+1)
```

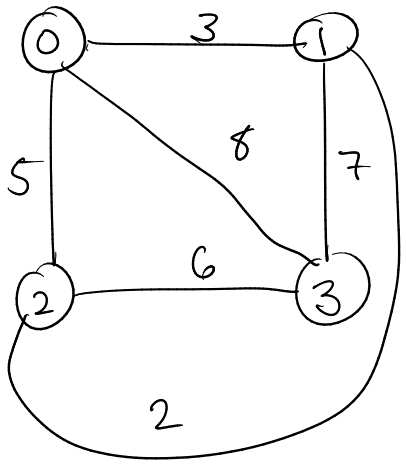
How fast is this in practice? This table is from Kreher + Stinson p134. The edge costs in each problem were random integers between 0 and 100

TABLE 4.3
Size of state space trees for Algorithms 4.10 and 4.13, on random instances of the Traveling Salesman problem with n vertices. Algorithm 4.13 is applied with bounding functions MINCOSTBOUND and REDUCEBOUND

n	Optimal Cost	Algorithm 4.10	Algorithm 4.13	
			MINCOSTBOUND	REDUCEBOUND
5	137	65	45	18
10	160	986,410	5,199	1,287
15	234	236,975,164,805	1,538,773	53,486
20	173	$\approx 3.3 \cdot 10^{17}$	64,259,127	1,326,640

naive TSP

Now lets try our example using the reduce bound.



$$\begin{bmatrix} \infty & 3 & 5 & 8 \\ 3 & \infty & 2 & 7 \\ 5 & 2 & \infty & 6 \\ 8 & 7 & 6 & \infty \end{bmatrix}$$

If $m=n$
return cost(X)
$M'(0,0) = \infty$
$Y = V - \{x_0, \ldots, x_{m-1}\}$
$j=1$
for $y \in Y$
$M'(0,j) = M(x_{m-1}, y)$
$j = j+1$

ϕ

```
i = 1
for x ∈ Y
        M'(i, 0) = M(x, x_0)
        i = i + 1
        j = 1
        for y ∈ Y
                M'(i, j) = M(x, y)
                j = j + 1
ans = Reduce(M')
for i from 1 to n-1
        ans = ans + M(x_{i-1}, x_i)
return ans
```

② Branch and Bound

As we noticed with the knapsack problem the order in which we take the elements of C_l can make a big difference

We can use the bounding function to choose

.

.

.

So for the Traveling salesman problem the algorithm will look like

Algorithm Branch and Bound Traveling Salesman

This table is from Kreher and Stinson p143. It compares the branch and bound versions with both bounding functions to the usual bounded backtrack versions. The instances are the same ones as the other table

TABLE 4.6
Size of state space trees for Algorithms 4.13 and 4.23, on random instances with n vertices. Algorithms 4.13 and 4.23 are shown with both bounding functions MINCOSTBOUND and REDUCEBOUND

n	Algorithm 4.13		Algorithm 4.23	
	MINCOSTBOUND	REDUCEBOUND	MINCOSTBOUND	REDUCEBOUND
5	45	18	25	9
10	5,199	1,287	490	102
15	1,538,773	53,486	128,167	5,078
20	64,259,127	1,326,640	6,105,089	39,035

standard bounded backtrack

branch and bound

(3) Heuristic search

Even with good bounding backtrack algorithms can be quite slow.

Sometimes its good enough to have a solution which is close to optimal.

How can we explore the state space tree so as to quickly get to a near optimal solution?

One approach is heuristic search

Vocabulary

The universe X is

$x \in X$ is feasible if

$P(x)$ is

A neighbourhood function is

$\rightsquigarrow$

Given N, a neighbourhood search is

eg For the Knapsack problem we could take

$X =$

A reasonable neighbourhood function might be

$N(X) =$

Possible neighbourhood search strategies include

-
-
-
-

The simplest (but often too simple to be useful) is

Meta Algorithm Generic Hill Climbing

Select a feasible $X \in \mathcal{X}$

```
searching = true
while searching
    get a feasible solution Y ∈ N(X) with P(Y) > P(X)
        (randomly, or largest exhaustively)
    if Y ≠ fail
        X = Y
    else
        searching = false
return X
```

The problem with this algorithm is

4. Next time

better heuristic searches