Billy Saysavath

Duy Do

Phung Tran

CPSC 335 – Section 3

Fall 2015

Assignment #3

# **Exhaustive Optimization Algorithm:**

* 1. Pseudocode:

#Clock starts at this point

Dist = farthest(n,P)

bestDist [:n]= Dist

A [n]

for i in range(0, n-1):

A[i] = i;

def print\_perm( n, A[], sizeA, P, bestSet, bestDist ):

i, dist

if (n == 1):

dist = sqrt( pow( P[A[i]].x – P[A[sizeA-1]].x, 2 ) + pow( P[A[i]].y – P[A[sizeA-1]].y, 2 ))

if( disk < bestDist ):

bestDist = dist

else

for i in range (0, sizeA):

print\_perm(n – 1, A, sizeA, P, bestSet, bestDist)

if ( n % 2 == 0 ):

A[i], A[n-1] = A[n-1], A[i]

else:

A[0], A[n-1] = A[n-1], A[0]

print\_perm(n – 1, A, sizeA, P, bestSet, bestDist)

def farthest(n, P):

max\_dist=0;

i, j;

dist;

for i in range(0, n):

for j in range(0, n):

dist = (P[i].x - P[j].x)\*(P[i].x - P[j].x) + (P[i].y - P[j].y)\*(P[i].y - P[j].y)

if (max\_dist < dist):

max\_dist = dist

return sqrt(max\_dist)

* 1. Analyze:

- Line 2 calls farthest function, and it takes O(n2) steps.

- Line 3 takes n steps. Line 4 takes a constant step, so we say 1.

- Line 6 takes n steps to fill the values in a permutation array A.

- Then, the program calls print\_perm. In print\_perm, line 10 takes 2 steps.

- In an if-else statement, line 12 takes 1 step. Line 13 takes 1 + max(1, 0) step.

- Line 16, the for-loop will take sizeA steps, which equals to number of points m.

- Multiply the recursive call to print\_perm at line 17 which, as we learned in a class, takes O(n!), and 1 + max(1, 1) for if-else statement from line 18 to 21.

- Lastly, we add another O(n!) for a recursive call to print\_perm. We have the following:

We have, where m = n; therefore,

1. **Approximation algorithms**
   1. Pseudocode:

#clock starts counting at this point

1. Visited [n] = false
2. A = farthest\_point(n, P)
3. for i in range[0,n]
4. M[i] = i
5. M[0]= A
6. Visited[A] = true
7. dist = 0
8. #calculate the nearest unvisited neighbor from node A
9. for i in range[1, n]:
10. B = nearest(n, P, A, Visited)
11. A = B
12. M[i] = A
13. Visited[A]=true
14. #cal calculate the length of Hamiltonian cycle
15. for i in range [0, n-1]:
16. dist += sqrt((P[M[i]].x - P[M[i+1]].x)\*(P[M[i]].x - P[M[i+1]].x) +

(P[M[i]].y - P[M[i+1]].y)\*(P[M[i]].y - P[M[i+1]].y))

1. dist += sqrt((P[M[0]].x - P[M[n-1]].x)\*(P[M[0]].x - P[M[n-1]].x) +

(P[M[0]].y - P[M[n-1]].y)\*(P[M[0]].y - P[M[n-1]].y))

#return an index of farthest point from P[0]

1. def farthest\_point(n, P):
2. farthest\_point = 0
3. max\_dist, dist
4. i, j
5. max\_dist = sqrt( (P[0].x - P[n-1].x)\*(P[0].x - P[n-1].x) + (P[0].y - P[n-1].y)\*(P[0].y - P[n-1].y) )
6. for i in range[0, n-1]:
7. for j in range[0, n-1]:
8. dist = sqrt( (P[i].x - P[j].x)\*(P[i].x - P[j].x) + (P[i].y - P[j].y)\*(P[i].y - P[j].y) )
9. if (max\_dist < dist) :
10. max\_dist = dist;
11. farthest\_point = i
12. return farthest\_point
13. #return an index of a nearest unvisited neighboring point
14. def nearest(n, P, A, Visited):
15. min\_dist, dist
16. nearest, i
17. for i in range[0, n]:
18. if ( !Visited[i] ):
19. min\_dist = sqrt( (P[A].x - P[i].x)\*(P[A].x - P[i].x) + (P[A].y - P[i].y)\*(P[A].y - P[i].y) )
20. nearest = i
21. for i in range[0, n]:
22. if ( !Visited[i] ):
23. dist = sqrt( (P[A].x - P[i].x)\*(P[A].x - P[i].x) + (P[A].y - P[i].y)\*(P[A].y - P[i].y) );
24. if (min\_dist > dist):
25. min\_dist = dist
26. nearest = i
28. return nearest;
    1. Analyze:

- Line 1, 5, 6, 7 each takes 1 step

- Line 3 iterates i from 0 to n-1 and line 4 takes 1 step so overall, these lines take n step

- Line 15 iterates i from 0 to n-2 and lines 16-17 each takes 1 step, so overall these lines take

2(n-1) steps

- Line 2 calls farthest function, line 18

* Lines 19-22 and line 31 each takes 4 steps
* Line 24, outer loop, iterates i from 0 to n-2 which takes n-1 steps
  + Line 25, inner loop, iterates i from 0 to n-2 which steps n-1 steps
    - Line 26 takes 1 step
    - Line 27-29 takes 1 + max(2, 0) which is 2 steps

- Line 9 iterates i from 1 to n-1 steps which takes n-1 steps

* lines 11-13 each takes 1 step, and line 10 calls nearest, line 33
* Lines 34,35,47 each takes 1 step
* Line 36 iterates from 0 to n-1 takes n steps
  + Lines 37 to 39 takes 1 + max(2,0) which is 3 steps
* Line 40 iterates from 0 to n-1 takes n steps
  + Lines 41 to 45 takes 1 + max(1+1+max(2,0),0) which is 5 steps

Altogether, we have:

T(n) = ­­­4 + n + 2(n – 1) + 4 + (n – 1) \* (n – 1)\*(1 + 2) + (n – 1)\*(3 + 3 + 3n + 5n)

= 8 + n + 2n – 2 + 6– 6n + 6 + (n – 1) \* (6 + 8n)

= 6– 3n + 12 + 6n + 8– 6 – 8n

= 14– 5n + 6

* T(n) ∈ O(