Solving LP with MS Excel

LP example to help a farmer

We have following linear constraints:

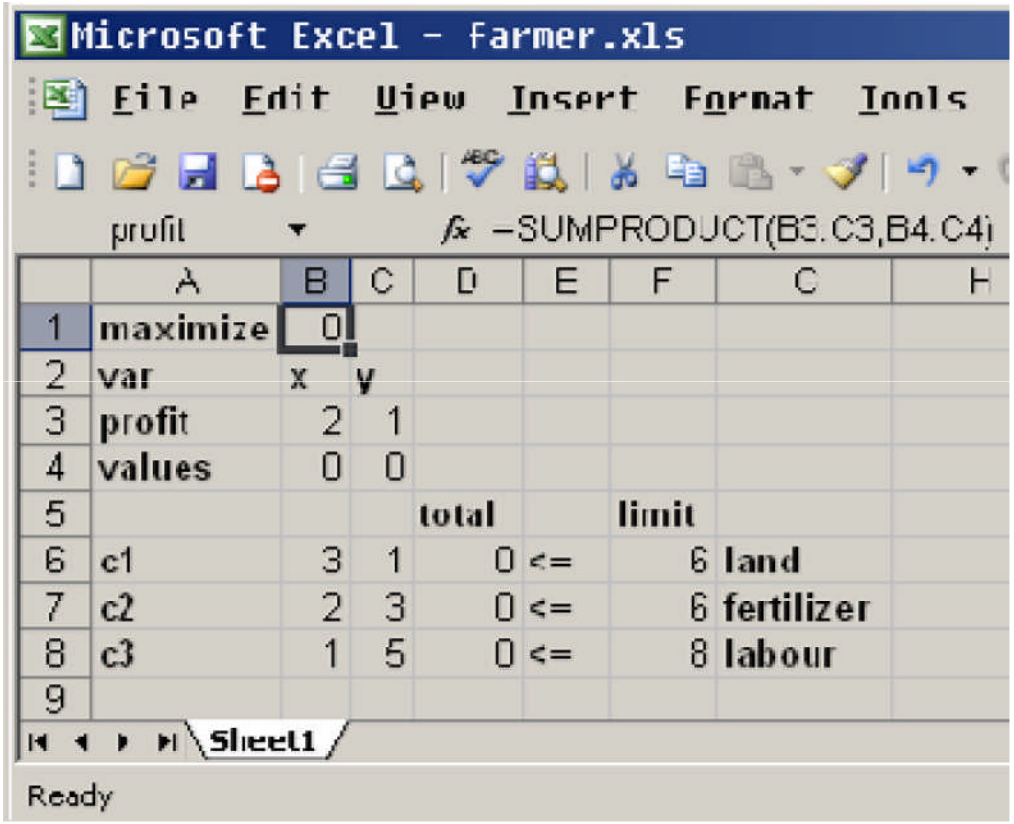
- 6 acres of land: $3x + y \leq 6$
- 6 tons of fertilizer: $2x + 3y \leq 6$
- 8 hour work day: $x + 5y \leq 8$
- Apples sell for twice as much as oranges
- We want to maximize profit: $z = 2x + y$
- Production is positive: $x \geq 0$, $y \geq 0$

Enter the data into Excel

x : apple
y : orange
Price apple = 2 price orange
maximize profit: $z = 2x + y$

Constraints:
6 acres of land: $3x + y \leq 6$
6 tons of fertilizer: $2x + 3y \leq 6$
8 hour work day: $x + 5y \leq 8$

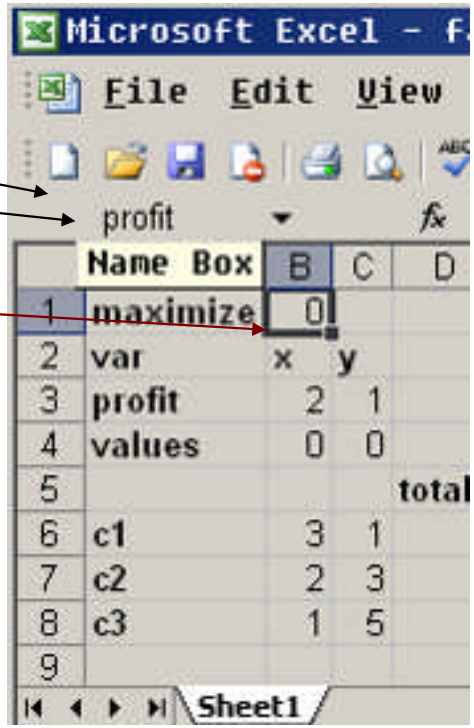
Production is positive: $x \geq 0$, $y \geq 0$



	A	B	C	D	E	F	G	H
1	maximize	0						
2	var	x	y					
3	profit	2	1					
4	values	0	0					
5				total		limit		
6	c1	3	1	0	<=	6	land	
7	c2	2	3	0	<=	6	fertilizer	
8	c3	1	5	0	<=	8	labour	
9								

Naming a cell

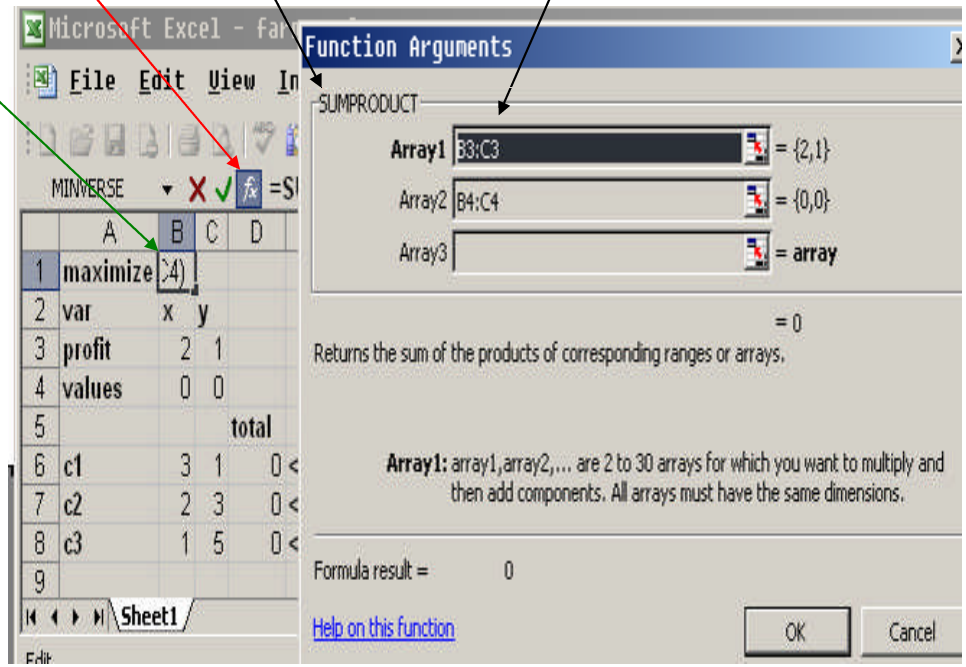
- Select B1
- Click on "*name box*",
- Type *"profit"*,
- Press *"Control-Shift-Enter"* to name B1 profit.



Input the objective function, (equation for profit)

Select cell B1, Click on ***fx***, Select *sumproduct*, Array1=B3:C3
Array2=B4:C4

So profit is
z = 2 * x + 1 * y
B1 = B3 * B4 + C3 * C4

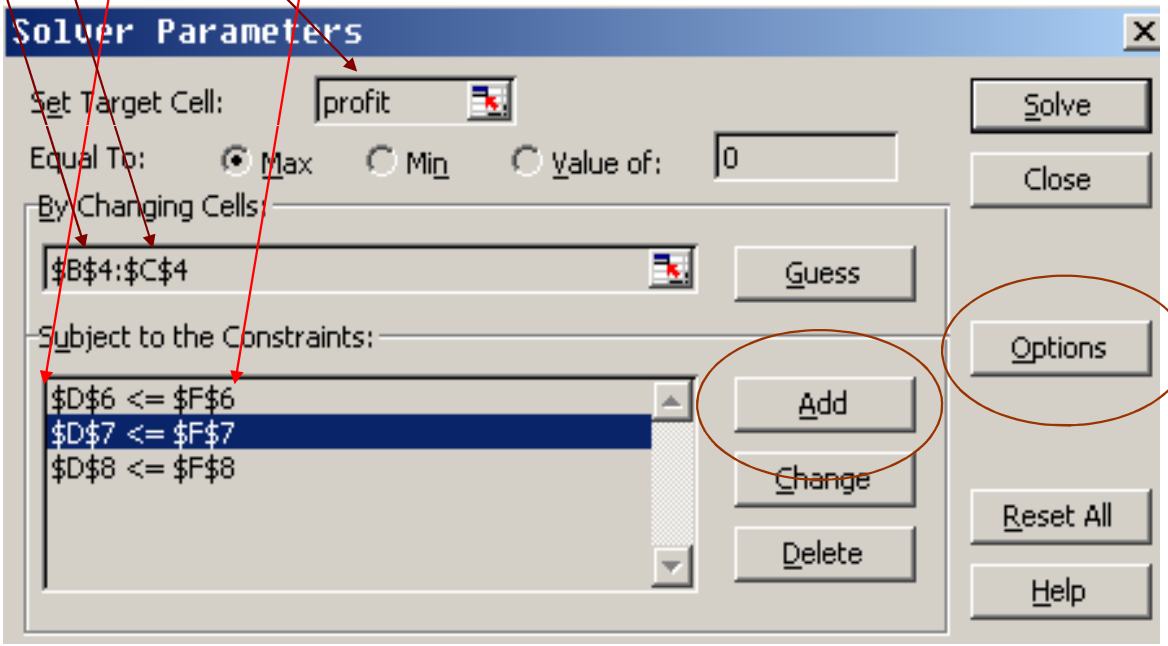


Enter the formula for constraints

- D6=SUMPRODUCT(B6:C6,$B$4:$C$4)
- D7=SUMPRODUCT(B7:C7,$B$4:$C$4)
- D8=SUMPRODUCT(B8:C8,$B$4:$C$4)

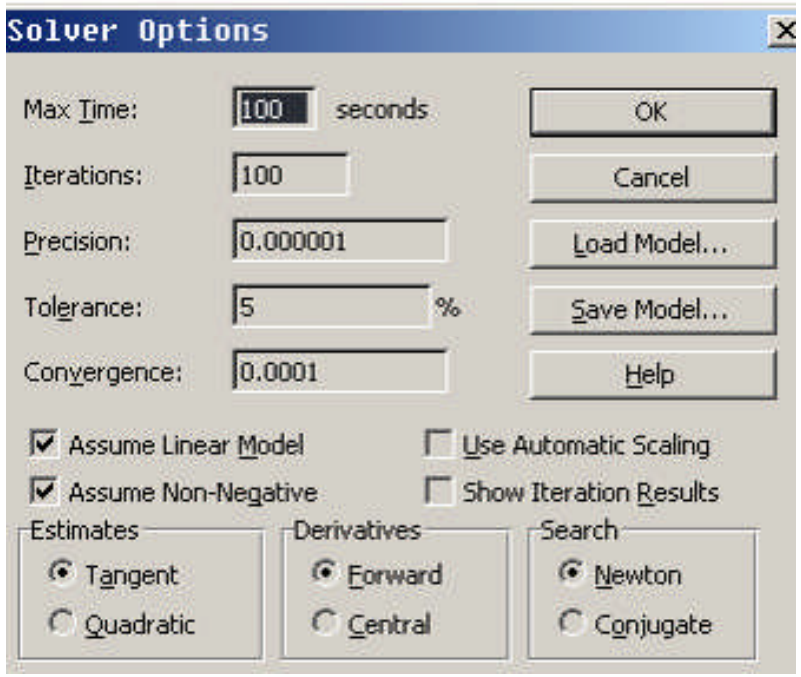
Tools > Solver

	A	B	C	D	E	F	G
1	maximize	0					
2	var	x	y				
3	profit	2	1				
4	values	0	0				
5				total		limit	
6	c1	3	1	0	<=	6	land
7	c2	2	3	0	<=	6	fertilizer
8	c3	1	5	0	<=	8	labour



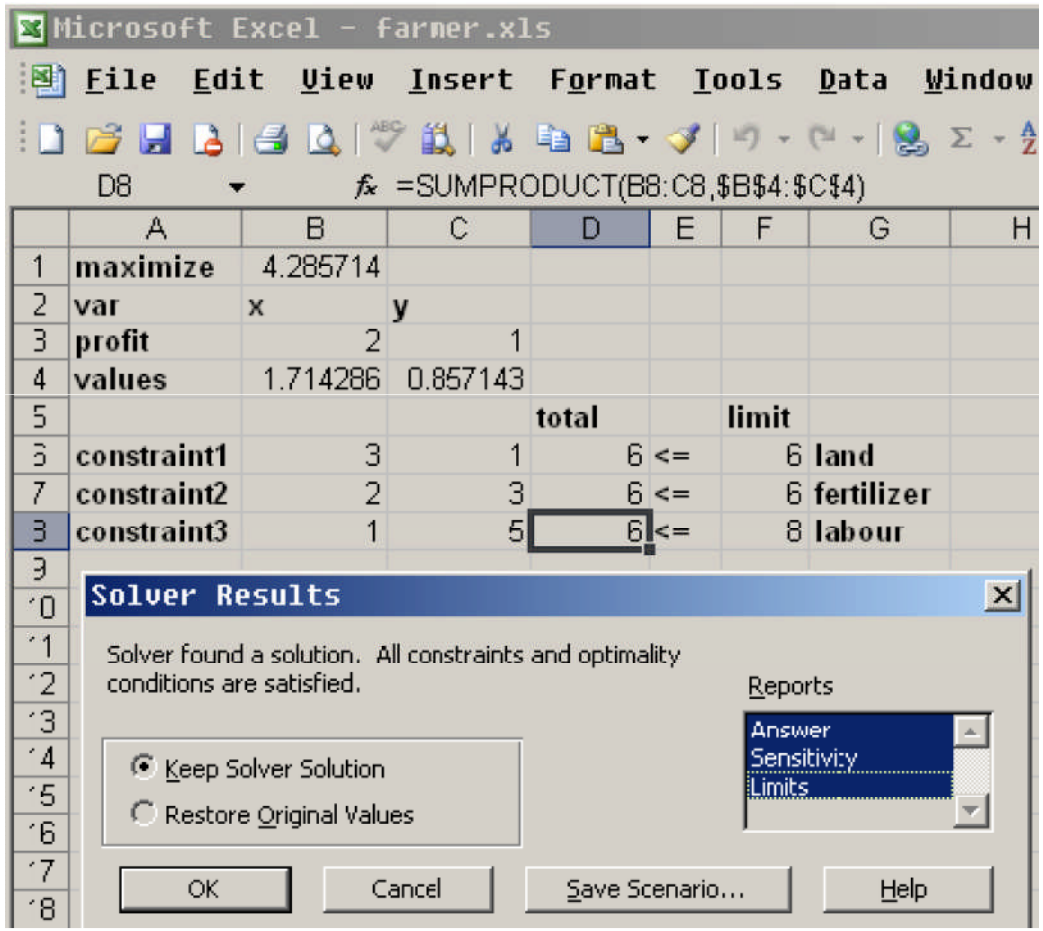
Solver > Options

- Defaults solver options are good.

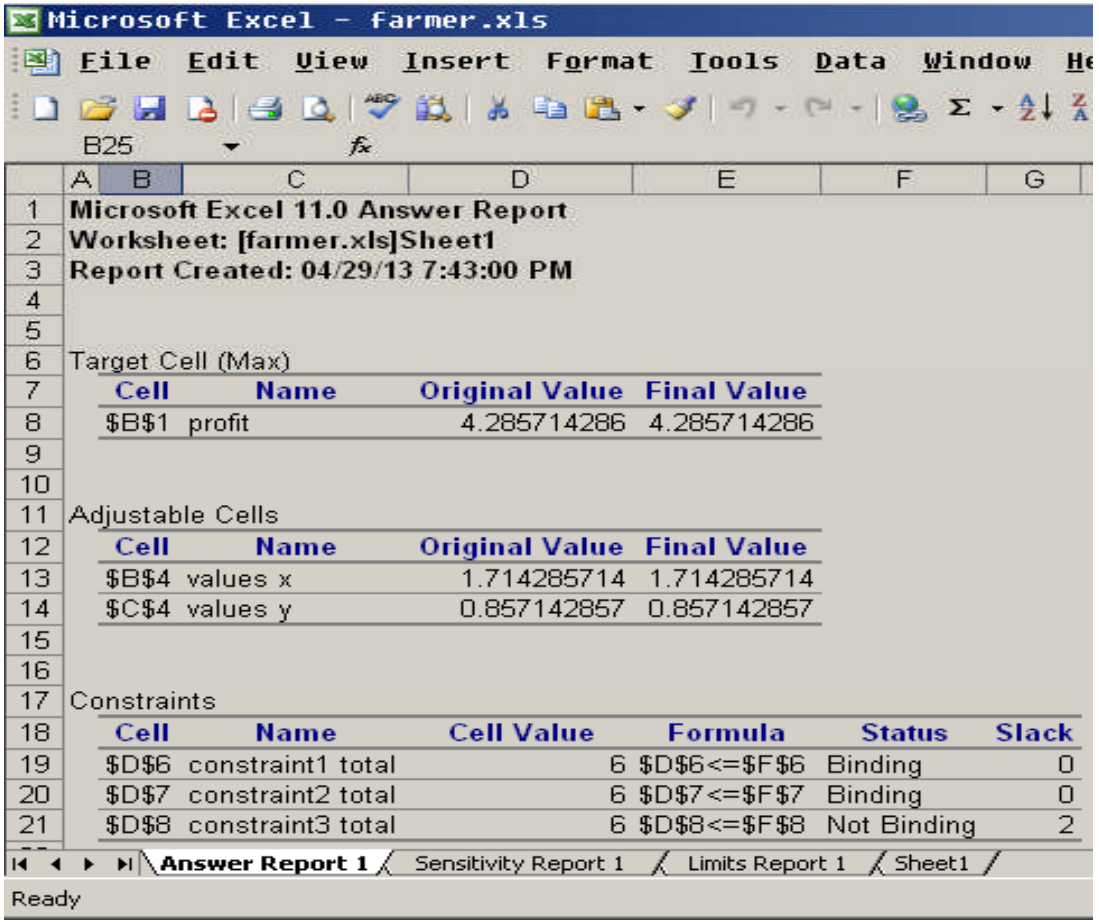


Solved

- x=1.714
- y=0.857
- z = 4.28



Answer report



Microsoft Excel - farmer.xls

File Edit View Insert Format Tools Data Window He

B25

Microsoft Excel 11.0 Answer Report
Worksheet: [farmer.xls]Sheet1
Report Created: 04/29/13 7:43:00 PM

Target Cell (Max)

Cell	Name	Original Value	Final Value
$B$1	profit	4.285714286	4.285714286

Adjustable Cells

Cell	Name	Original Value	Final Value
$B$4	values x	1.714285714	1.714285714
$C$4	values y	0.857142857	0.857142857

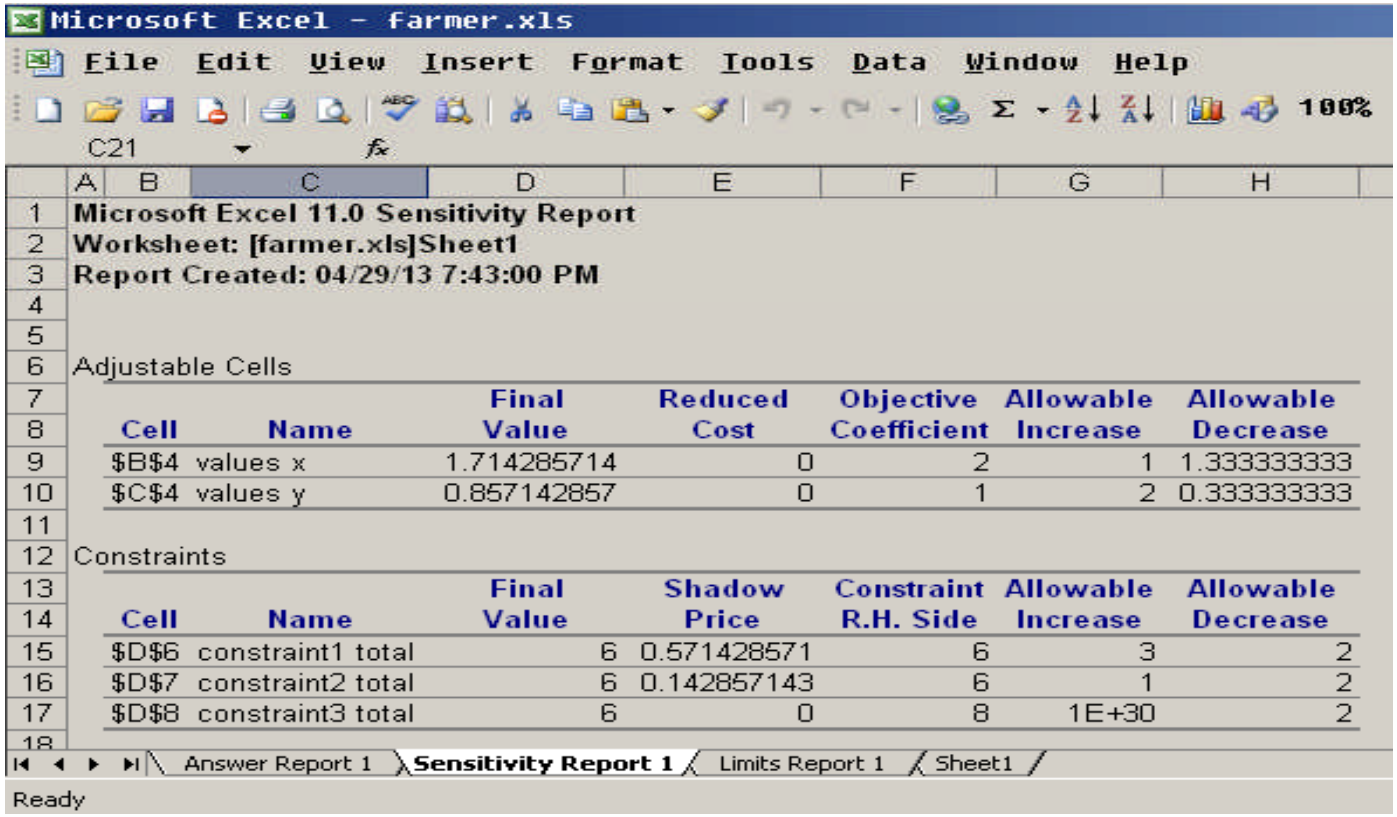
Constraints

Cell	Name	Cell Value	Formula	Status	Slack
$D$6	constraint1 total	6	$D$6<=$F$6	Binding	0
$D$7	constraint2 total	6	$D$7<=$F$7	Binding	0
$D$8	constraint3 total	6	$D$8<=$F$8	Not Binding	2

Answer Report 1 | Sensitivity Report 1 | Limits Report 1 | Sheet1

Ready

Sensitivity report



Microsoft Excel - farmer.xls

File Edit View Insert Format Tools Data Window Help

C21

Microsoft Excel 11.0 Sensitivity Report
Worksheet: [farmer.xls]Sheet1
Report Created: 04/29/13 7:43:00 PM

Adjustable Cells

Cell	Name	Final Value	Reduced Cost	Objective Coefficient	Allowable Increase	Allowable Decrease
$B$4	values x	1.714285714	0	2	1	1.333333333
$C$4	values y	0.857142857	0	1	2	0.333333333

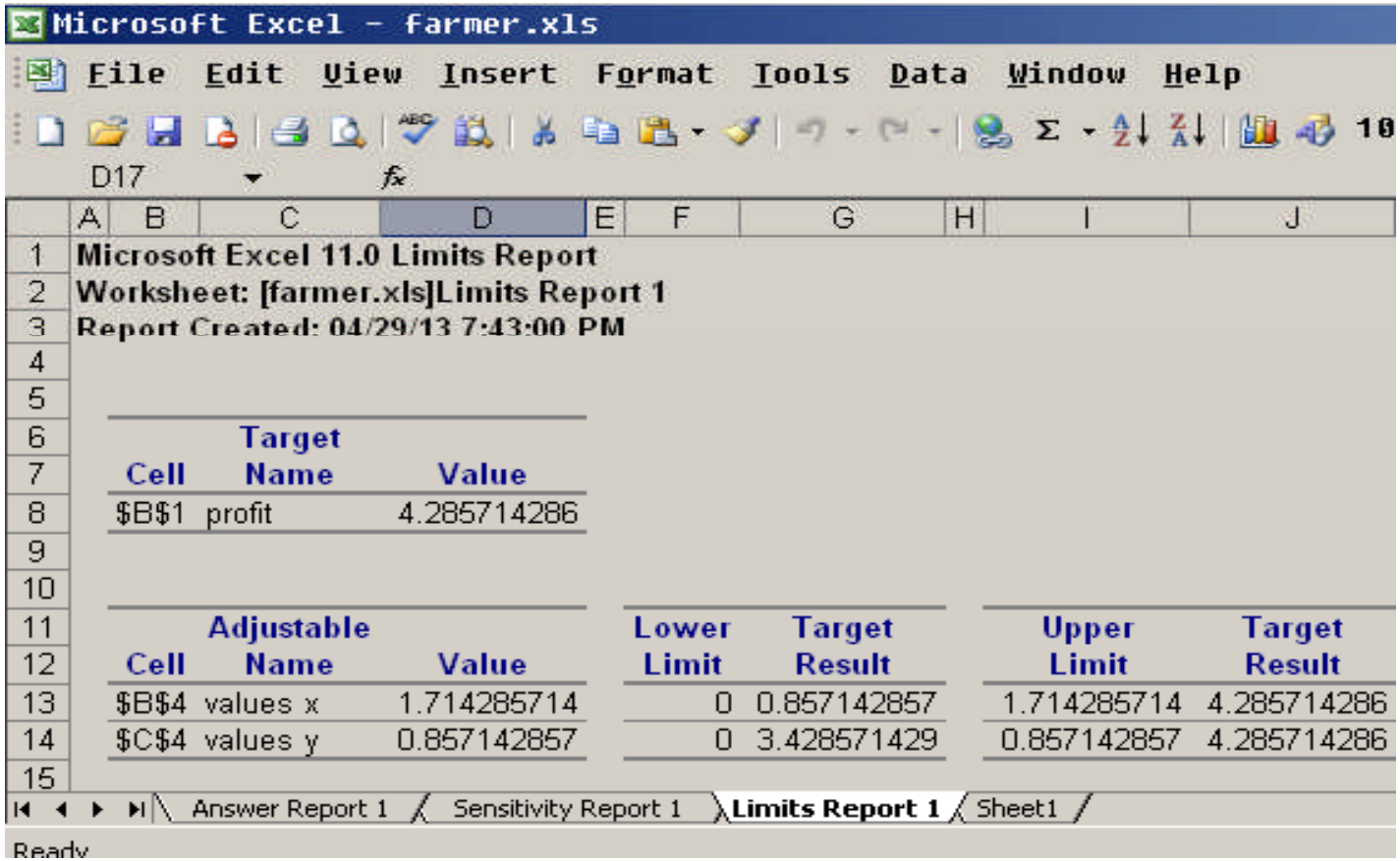
Constraints

Cell	Name	Final Value	Shadow Price	Constraint R.H. Side	Allowable Increase	Allowable Decrease
$D$6	constraint1 total	6	0.571428571	6	3	2
$D$7	constraint2 total	6	0.142857143	6	1	2
$D$8	constraint3 total	6	0	8	1E+30	2

Answer Report 1 | Sensitivity Report 1 | Limits Report 1 | Sheet1

Ready

Limits Report



Microsoft Excel - farmer.xls

File Edit View Insert Format Tools Data Window Help

D17

Microsoft Excel 11.0 Limits Report
Worksheet: [farmer.xls]Limits Report 1
Report Created: 04/29/13 7:43:00 PM

Cell	Target Name	Value
$B$1	profit	4.285714286

Cell	Adjustable Name	Value	Lower Limit	Target Result	Upper Limit	Target Result
$B$4	values x	1.714285714	0	0.857142857	1.714285714	4.285714286
$C$4	values y	0.857142857	0	3.428571429	0.857142857	4.285714286

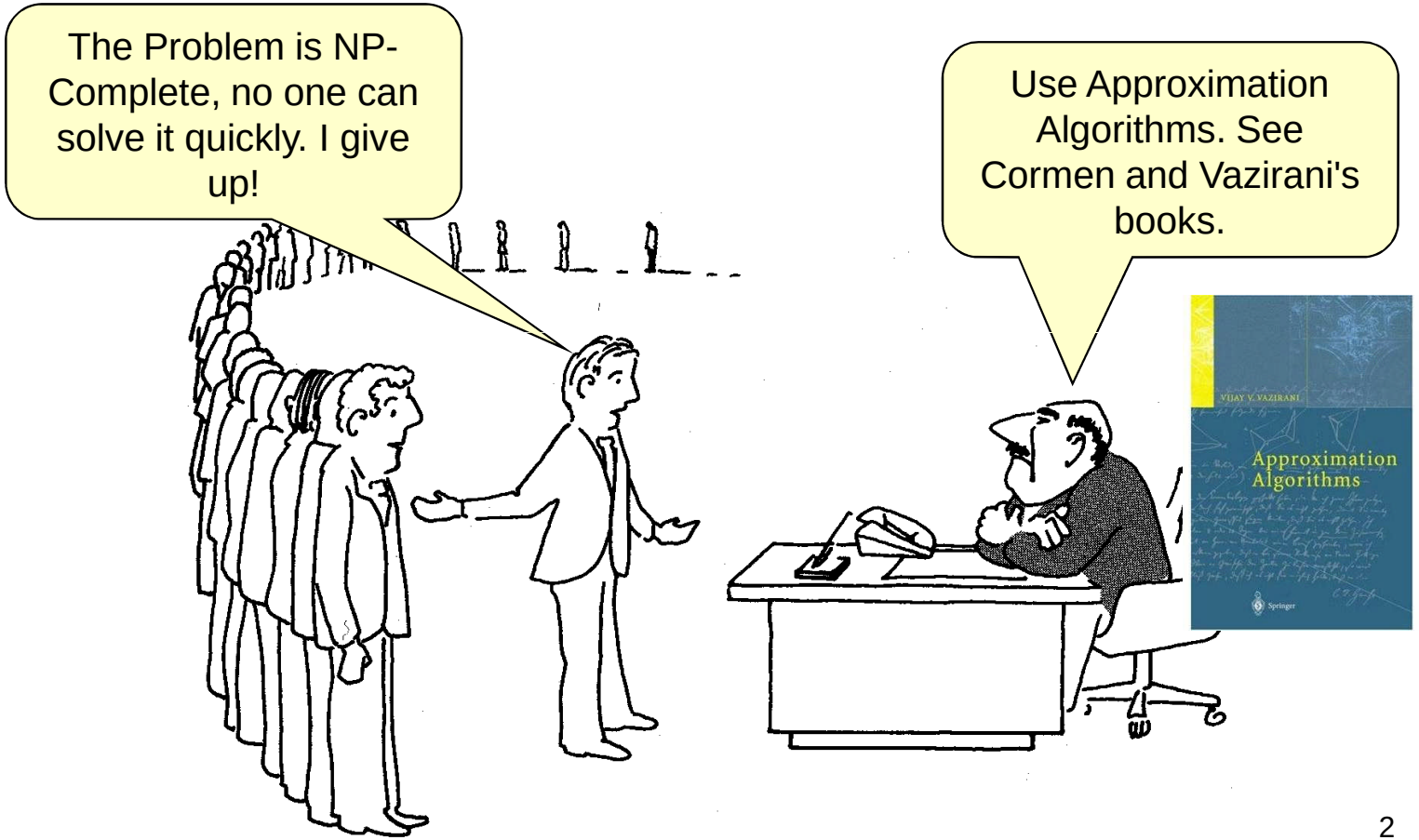
Answer Report 1 | Sensitivity Report 1 | Limits Report 1 | Sheet1

Ready

Aproximation Algorithms

Cormen chapter 35

NP-completeness is not the end



Coping With NP-Hardness

Brute-force algorithms.

- Develop clever enumeration strategies.
- Guaranteed to find optimal solution.
- No guarantees on running time.

Heuristics

Develop intuitive algorithms.
Guaranteed to give some correct solution in polynomial time.
No guarantees on quality of solution.

Approximation algorithms (AA)

- Guaranteed to run in polynomial time.
- Guaranteed to find "high quality" solution, say within 1% of optimum.
  How to prove a solution’s value is close to optimum,
  without even knowing what optimum value is?

Approaches

Special cases of graph may have faster algorithms
e.g. vertex cover for bipartite graphs.
Fixed parameter problems, may have faster algorithms
e.g. graph coloring, for k=2.
Average case
find an algorithm which works well on average,
e.g. simplex for LP.
Approximation algorithms
find an algorithm which return solutions that are
guaranteed to be close to an optimal solution.

Approx ratio ρ(n)

An AA is bounded by ρ(n) (Rho) if cost of the AA solution c (for input size n) is within a factor ρ(n) of the optimal c*, that is:

An AA has an **approximation ratio** ρ(n) if for any input of size n

$$\max(C/C^*, C^*/C) \leq \rho(n)$$

where C = cost of solution produced by the AA.

C^* = cost of optimal solution > 0

for both minimization and maximization problems.

Definitions: PTAS, FPTAS

Approximate Scheme: AA that takes $\varepsilon > 0$ as input and is a $(1 + \varepsilon)$-approximation algorithm.
That is, we can get as near to the optimal as we wish.

PTAS (Polynomial-time approximate scheme):
For **fixed** ε, AA scheme runs in polynomial time for input size n. That is, a constant decrease in ε, will result in a constant increase in running time.

Fully PTAS: polynomial time in n and $1/\varepsilon$.
E.g. of FPTAS: $O((1/\varepsilon)^2 * n^3)$

AA examples

- Vertex cover, using greedy
- TSP⇔, using MST.
- TSP, Hamiltionian path, NO AA.
- Set Cover,
- 3CNF, randomized 7/8 algorithm.
- Weighted vertex cover, using LP.
- Subset sum (partition), FPTAS divide and conquer.

Aproximation Algorithm for Vertex Cover using LP

Cormen chapter 35

- **Approximating weighted vertex cover using linear programming.**
- Minimum-weight VC problem:

  Given an undirected graph G = (V, E), where each $v \in V$ has an associated positive weight $w(v)$. For any vertex cover V', $w(V') = \sum_{v \in V'} w(v)$ The goal is to find a vertex cover of minimum weight.
- Associate a variable $x(v)$ with each $v \in V$, and let $x(v) \in \{0, 1\}$.

  $x(v) = 1$ means v is in the VC; 0 0/w.
- For each edge (u, v), at least one of u and v must be in VC.

  Thus $x(u) + x(v) \geq 1$.
- 0-1 linear program:

$$\text{min} \sum_{v \in V} w(v)x(v)$$

$$\text{Subject to} \quad x(u) + x(v) \geq 1 \quad \text{for each } (u, v) \in E$$

$$x(v) \in \{0, 1\} \quad \text{for each } v$$

- Linear-programming relaxation:

$$\text{min} \sum_{v \in V} w(v)x(v)$$

$$\text{s.t.} \quad x(u) + x(v) \geq 1 \quad \text{for each } (u, v) \in E$$

$$x(v) \leq 1 \quad \text{for each } v \in V$$

$$x(v) \geq 0 \quad \text{for each } v \in V$$

- **Approximating weighted vertex cover using linear programming.**
- Minimum-weight VC problem:

  $\sum_{v \in V'} w(v)$

  Given an undirected graph G = (V, E), where each $v \in$ V has an associated positive weight w(v). For any vertex cover V', w(V') = The goal is to find a vertex cover of minimum weight.

  $\sum_{v \in V} w(v)x(v)$
- Associate a variable $x(v)$ with each $v \in$ V, and let $x(v) \in \{0, 1\}$.

  $x(v)$ = 1 means v is in the VC; 0 0/w.

  $\sum_{v \in V} w(v)x(v)$
- For each edge (u, v), at least one of u and v must be in VC.

  Thus $x(u) + x(v) \geq 1$.

- 0-1 linear program:

  min $\sum_{v \in V'} w(v)$

  Subject to $x(u) + x(v) \geq 1$ for each $(u, v) \in E$

  $x(v) \in \{0, 1\}$ for each v

- Linear-programming relaxation:

  min $\sum_{v \in V} w(v)x(v)$

  s.t. $x(u) + x(v) \geq 1$ for each $(u, v) \in E$

  $x(v) \leq 1$ for each $v \in V$

  $x(v) \geq 0$ for each $v \in V$

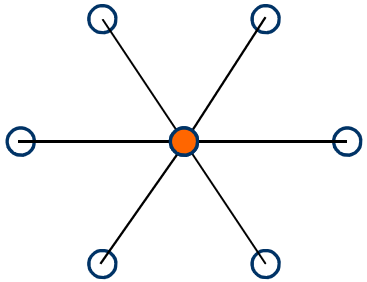
Aproximation Algorithms for Vertex Cover

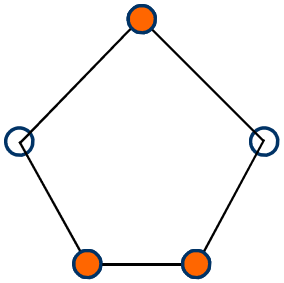
Cormen chapter 35

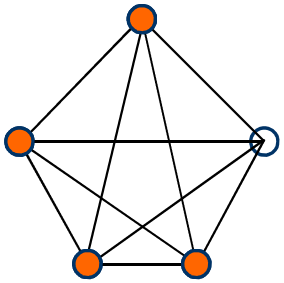
Vertex Cover

Vertex cover: a subset of vertices which "covers" every edge.
An edge is covered if one of its endpoint is chosen.

The Minimum Vertex Cover Problem:
Find a vertex cover with minimum number of vertices.



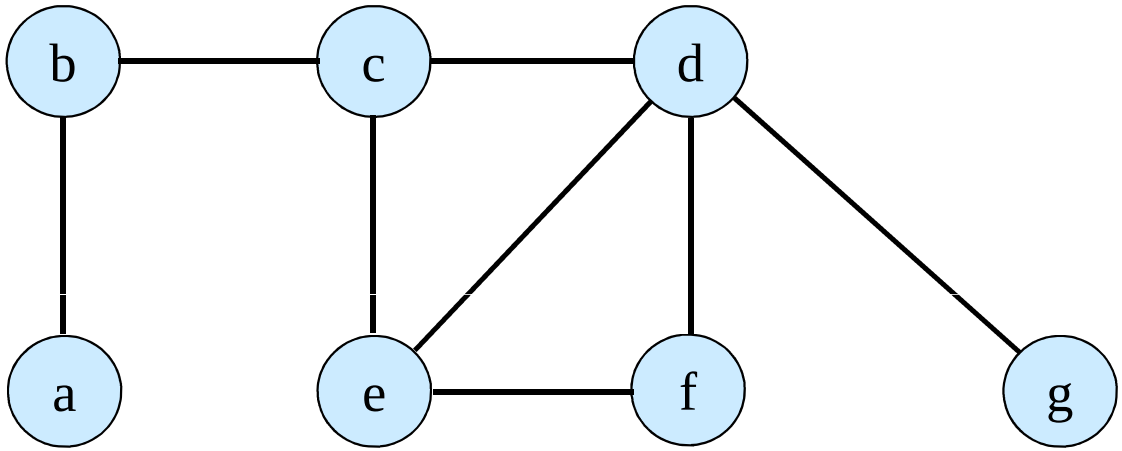




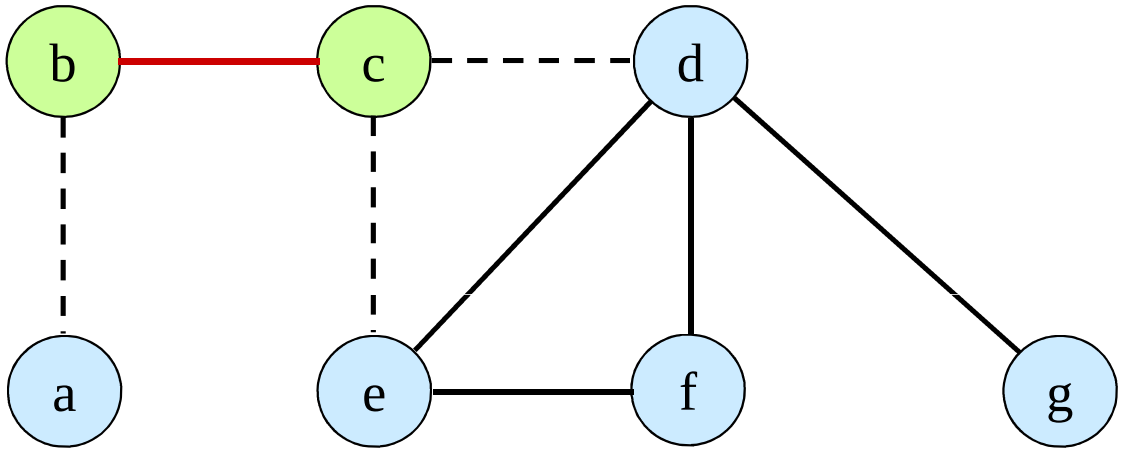
Vertex Cover AA

```
Approx-VC(G)
    C = { }
    E′ = E[G]
    while E′ ≠ { } do
        pick edge (u,v) in E′
        C = C ∪ {u,v};
        remove from E′ every edge incident on either u or v
    return C // cover.
```

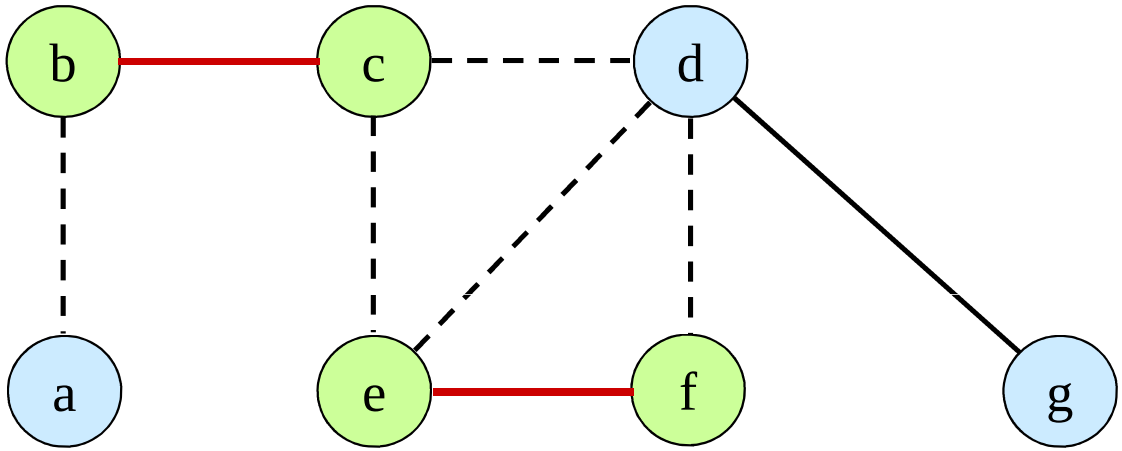
Example: Vertex cover AA



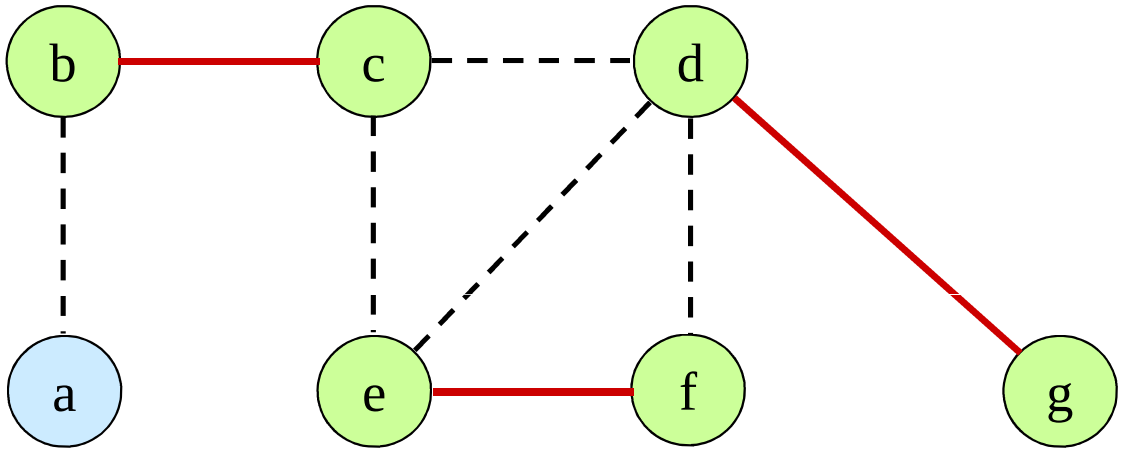
Example



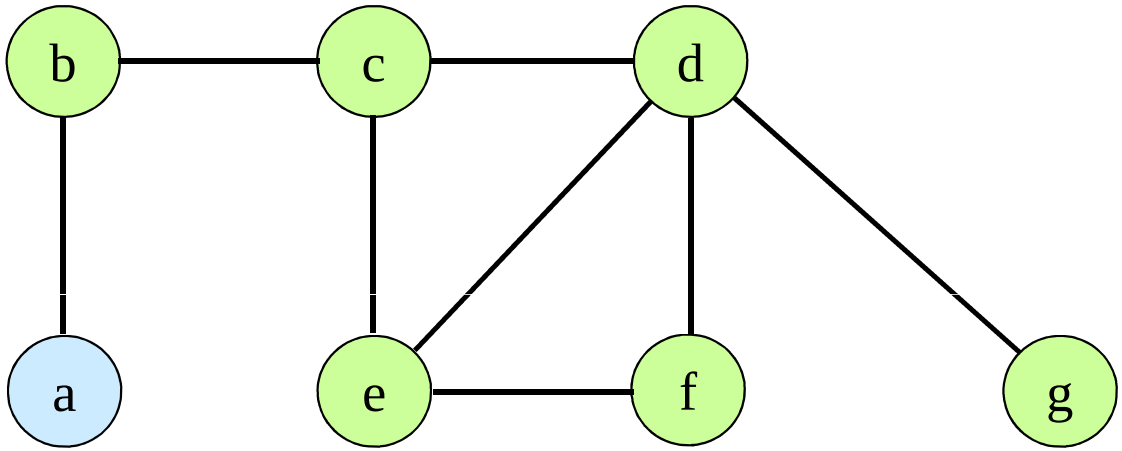
Example



Example

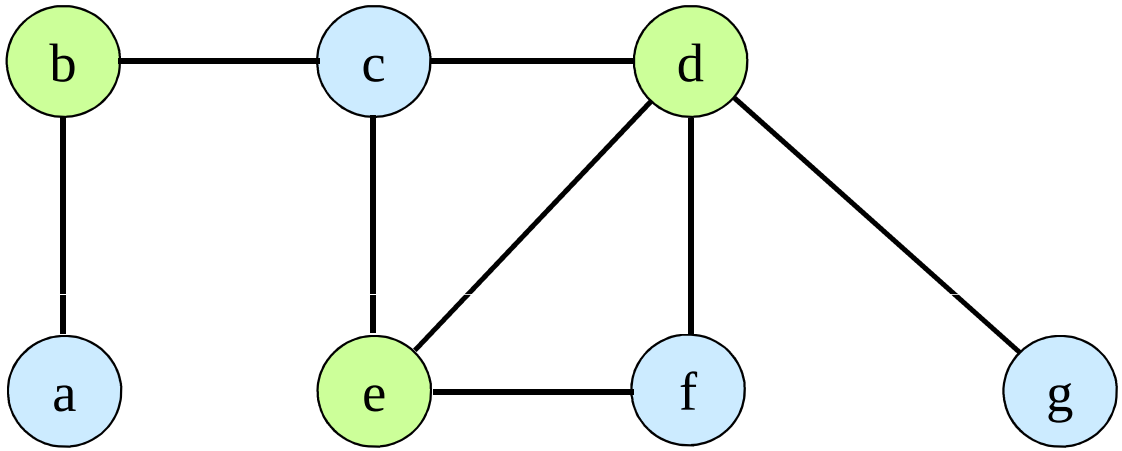


Example



Vertex cover produced by the algorithm
(6 vertices in VC)

Example



Optimal vertex cover
(3 vertices in VC)

VC AA Ratio is 2

VC AA finds upto twice the optimal number of vertices.

Proof:

Let A = set of edges selected by AA.

Notice: That no two edges in A have a common endpoint,

So our solution size is $|C| = 2|A|$.

The optimal cover C* must include one the two endpoints.

hence $|A| \leq |C^*|$.

Combining: $|C| = 2|A| \leq 2|C^*|$.

So our solution is at most twice the optimal. QED

Note: We don't know the optimal value, but we do know lower bound on C*.

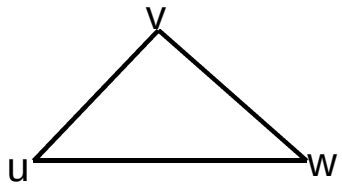
Aproximation Algorithm for TSP

Cormen chapter 35

⇔ Triangle Inequality

Given complete, undirected graph.
For all vertices, u,v,w, require:

$$c(u,w) \leq c(u,v) + c(v,w) \ldots \Leftrightarrow \text{inequality}$$



Distances on the euclidean plane obey the triangle inequality.

TSP⇔ (with Triangle Inequality)

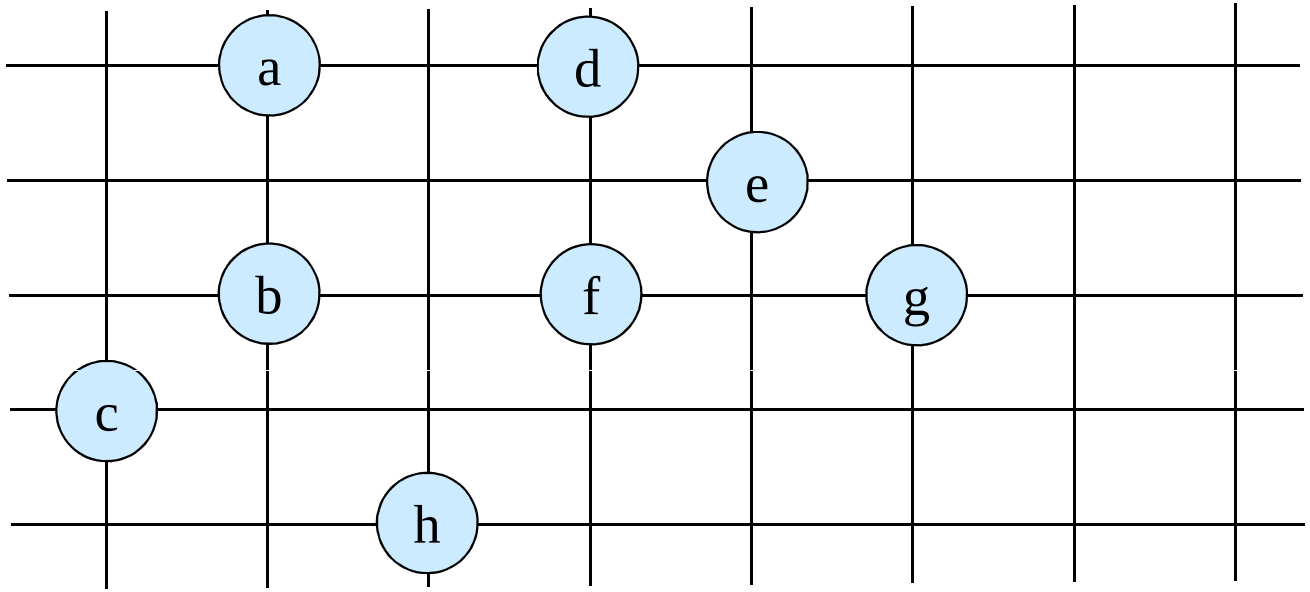
TSP⇔ remains NP-complete even with ⇔inequality.

Note that a TSP tour with any one edge removed is a MST of the graph

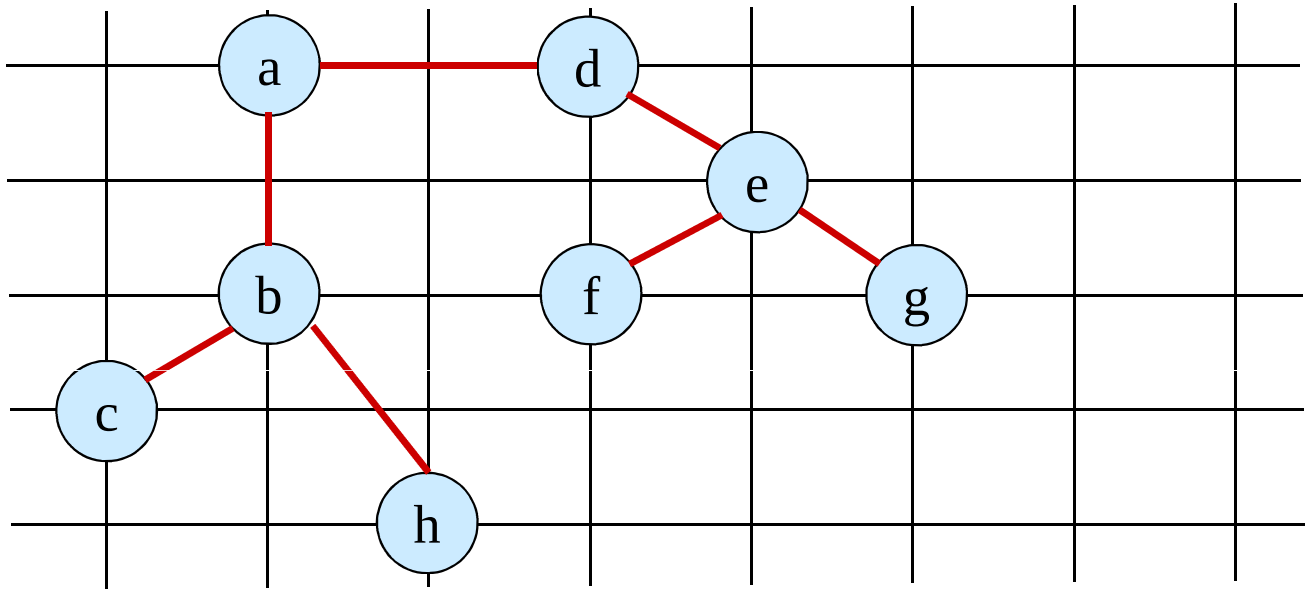
Approx-TSP is O(V^2), solution is within twice the optimal.

```
Approx- TSP(G,c)
    select r ∈ v[G]
    call MST-Prim(G,c,r) to construct MST with root r
    let  L = vertices on preorder walk of MST
    // This step needs triangle inequality to skip duplicate visits.
    let H = cycle that visits vertices in the order L
    return H
```

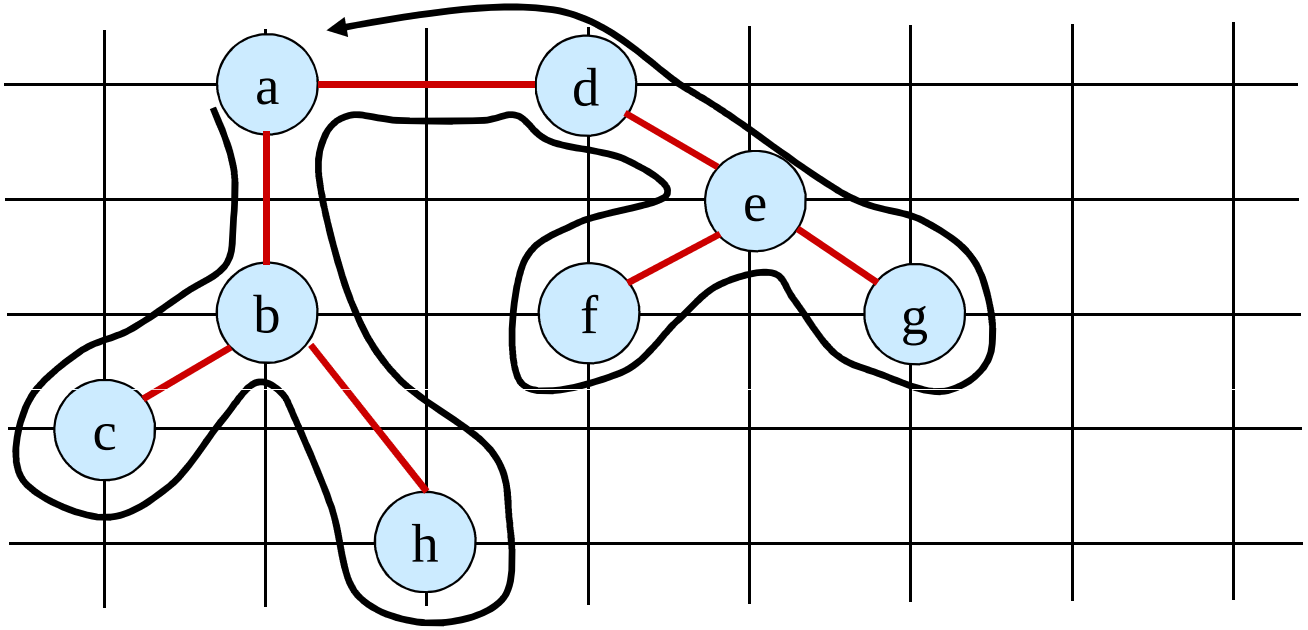
Example TSP AA



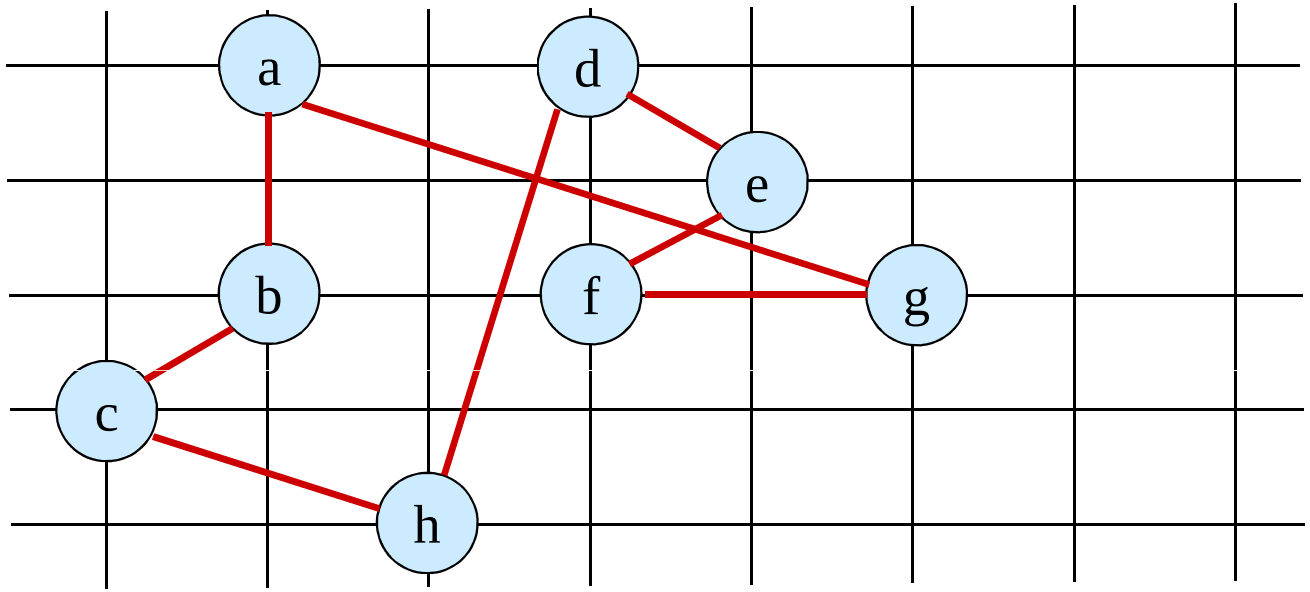
Example TSP MST prim



Example: Pre-order walk of MST

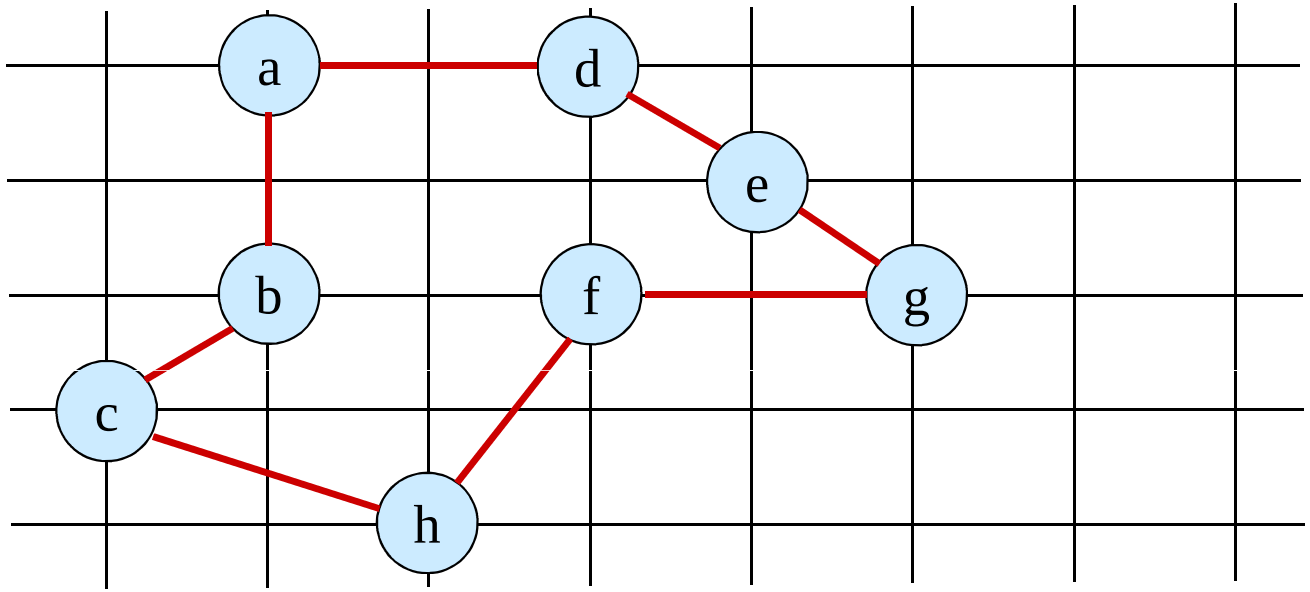


Example: TSP solution



Computed Tour

Example: TSP optimal tour



The Optimal tour is 23% shorter, than the AA

Claim: Approximation Ratio is 2

Proof:

Let H^* = optimal tour, T = MST.
Deleting an edge of H^* yields a spanning tree.
So, $c(T) \leq c(H^*)$.

Consider **full walk** W of T.
Ex: a,b,c,b,h,b,a,d,e,f,e,g,e,d,a.
Visits each edge twice

$\Rightarrow c(W) = 2c(T)$.

Hence, $c(W) \leq 2c(H^*)$.

Triangle inequality $\Rightarrow$ can delete any vertex from W and cost doesn't increase.

Deleting all but first appearance of each vertex yields **preorder walk**.
Ex: a,b,c,h,d,e,f,g.

Corresponding cycle H = W with some edges removed.
Ex: a,b,c,h,d,e,f,g,a.

$c(H) \leq c(W)$.

Implies $c(H) \leq 2c(H^*)$.

General TSP has no AA

Theorem 35.3: If $P \neq NP$ and approximation ratio $\rho \geq 1$,
There is no polynomial-time AA with ρ for the general TSP.

Proof:

Suppose $\exists$ polynomial-time a.a. A with approximation ratio ρ.

WLOG, assume ρ is an integer.

We use A to solve the Hamiltonian Circuit (HC) Problem
in polynomial time.

Proof: solving HC using TSP AA.

Given G =(V,E) to find a HCP,
Create a new complete graph G' of G
$G' = (V,E')$ be the complete graph on V.
For each $(u,v) \in E'$, the cost is:

$$c(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ \rho|V|+1 & \text{otherwise (very large)} \end{cases}$$

We run TSP AA on instance (G',c).

Proof Continued

If G has a H.C., then G′ has a tour of cost |V|.

If G does not have a H.C., then any tour in G′ uses an edge not in E, and costs at least $(\rho|V|+1)+ (|V| – 1) > \rho|V|$.

Now, consider running AA on (G′,c), to find
a tour of cost at most ρ times cost of optimal.

Thus, if G has a HC, A must return it.
If G has no HC, then A returns a tour of cost > ρ|V|.

Therefore, can use A to solve HC in polynomial time.
But HC is NPC, [Contradiction] so this is a impossible,
so TSP AA is also impossible.

Randomization: MAX-3CNF

A randomized algorithm has an approximation ratio of $\rho(n)$ as before, but C is interpreted as an expected cost.

Example: MAX-3CNF Satisfiability. Given an expression in 3CNF, want an assignment that makes as many clauses as possible true.

Assume: Each clause consists of exactly three distinct literals, and no clause contains both a variable and its negation.

Approximation Algorithm: Randomly (with equal probability of 0 and 1) assign values to variables.

Claim: Approximation Ratio is 8/7

Proof:

First, note that

P[clause i satisfied]
= 1 – P[all 3 literals are 0]
= 1 – $(1/2)^3$ = 7/8.

Define the indicator random variable Y_i = I{clause i is satisfied}.

Then, $E[Y_i]$ = P[clause i satisfied]
= 7/8.

Let $Y = \sum_i Y_i$.

Expected number of satisfied clauses is:

$$E[Y] = E\left[\sum_{i=1}^{m} Y_i\right]$$

$$= \sum_{i=1}^{m} E[Y_i]$$

$$= \sum_{i=1}^{m} 7/8$$

$$= 7m/8$$

Optimal solution is upper bounded by m. So, approx. ratio is at most m/(7m/8) = 8/7.

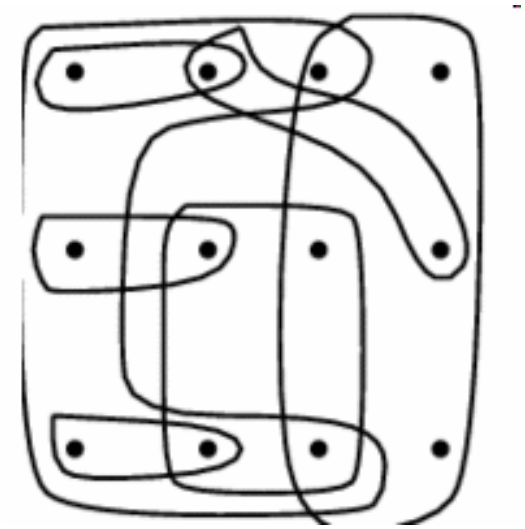
Approx Algorithm for Set covering

SC Problem and example

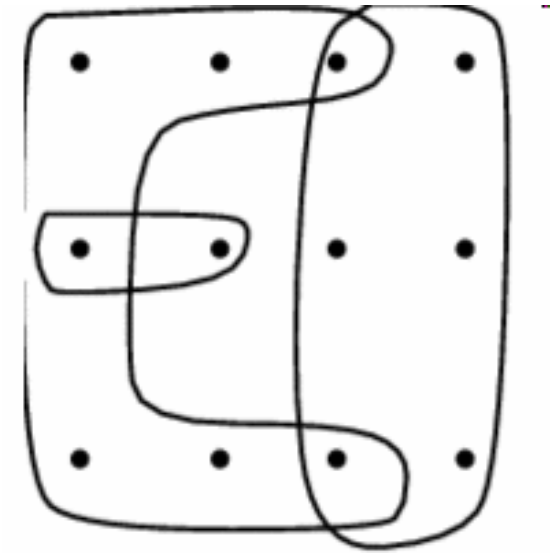
Given a set X and a F family of subset of X.
F = {f1,f2,...}, Union(F) = X
Find C, the minimal subset of F : Union(C) = X.
This problem is **NP-Complete**.

Example: Given: X = {1,2,3,4}
F = { {1,2},{1,2,3},{2,3},{2,4}, {1,4} }
One solution: C = { {1,2,3}, {2,4} }
Size(C)=2, Size(F) = 4.
Note: Union(F) = Union(C) = X.
F is also a solution but not the smallest.

SC Example



Input: X, F



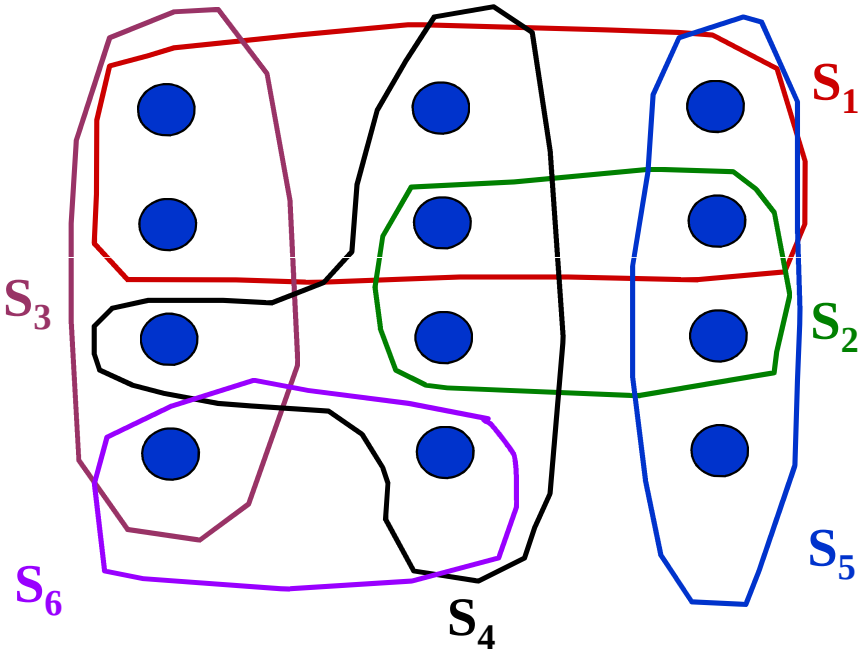
Output: C set cover of X

Set Cover: Greedy AA algorithm

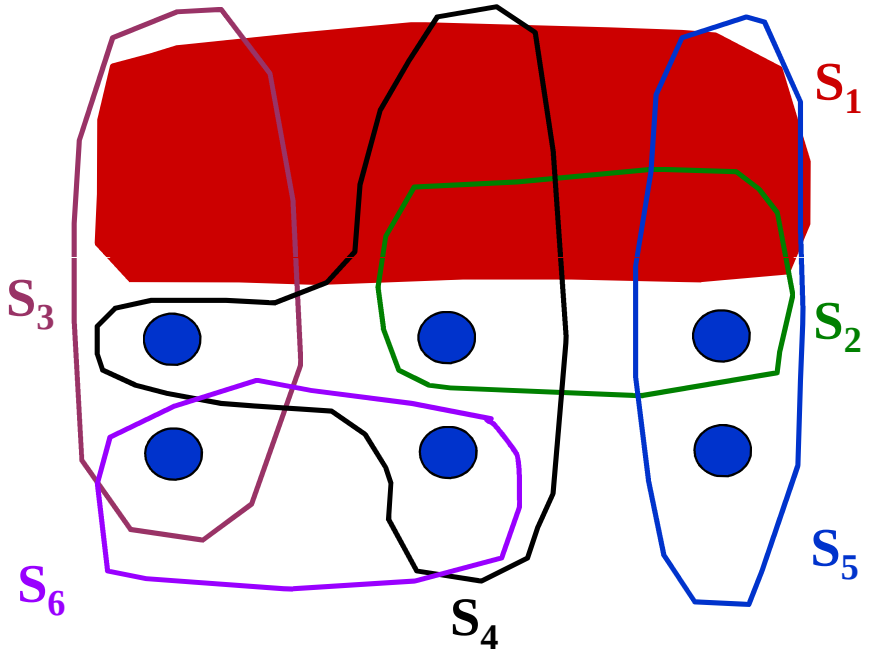
Family of Sets.
Each set consists of elements from X.

```
Greedy-SC( X, F)
    U = X
    C = { }
    while U ≠ { } do
        select S ∈F that maximizes |S ∩ U|;
        U = U – S;
        C = C ∪ {S}
    do;
    return C
```

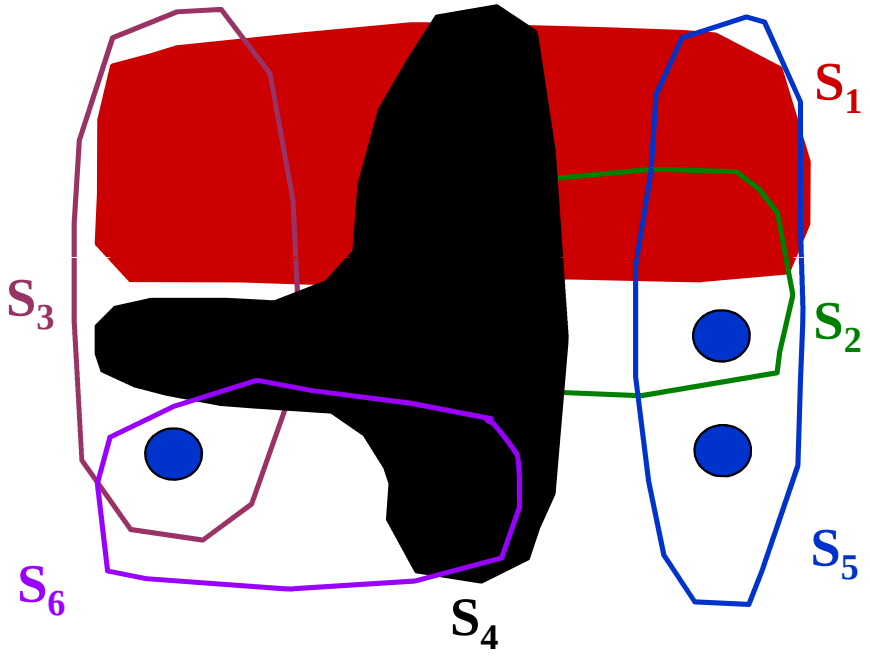
Example



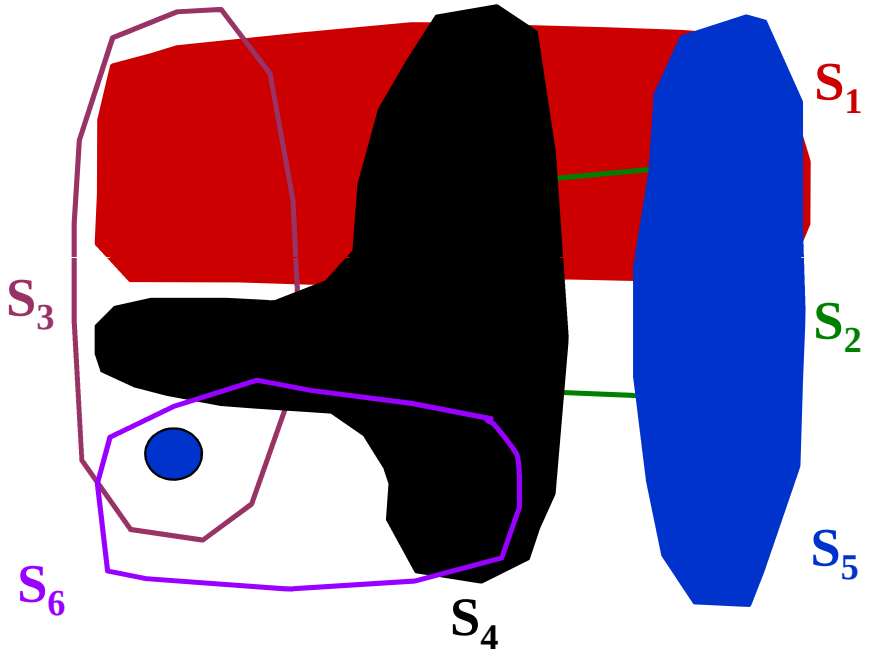
Example



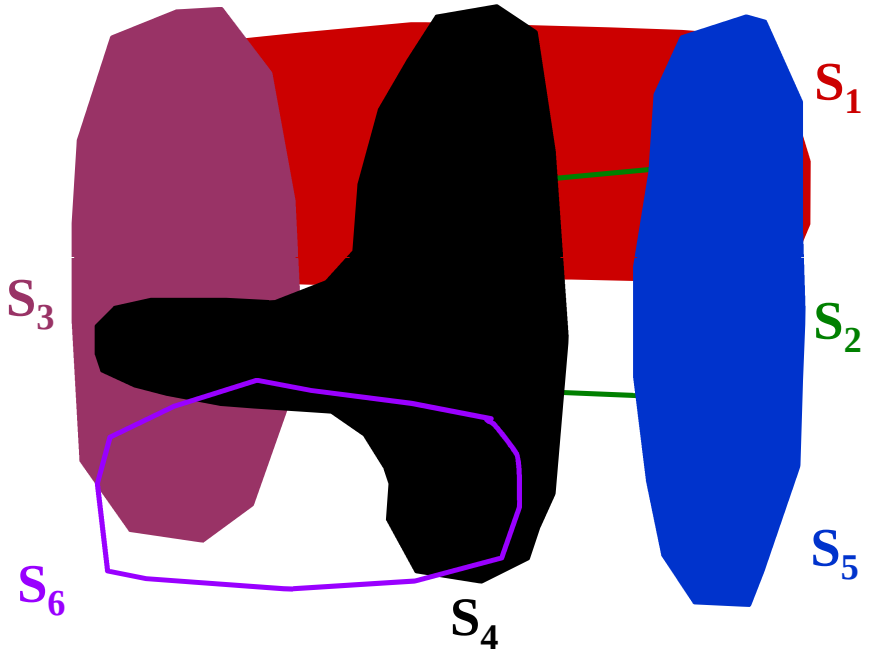
Example



Example

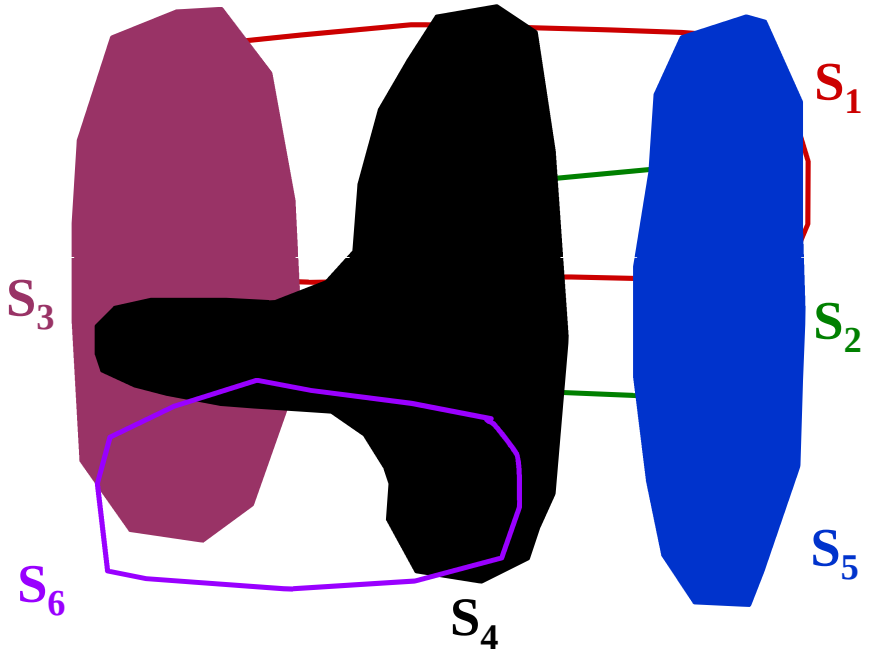


Example



Computed cover consists of 4 sets.

Example



Optimal cover consists of **3 sets**.

Claim: Approx. Ratio is Logarithmic

Let $H(d)$ be d^{th} harmonic number

$$\sum_{i=1,\ldots,d} 1/i = 1 + 1/2 + 1/3 + \ldots + 1/d = O(\ln(d))$$

Claim: Greedy-SC has an approx. ratio of $H(\max\{|S| : S \in F\})$.
Note: approx. ratio is $\ln|X|+1$

Proof: Let
C = set cover returned by the algorithm.
C* = optimal S.C.
S_i = i^{th} set selected by the algorithm.

Claim: Approx. Ratio is Logarithmic

Let cost of C be |C|, i.e., **charge** 1 for each S_i.

Distribute this cost to elements of X.

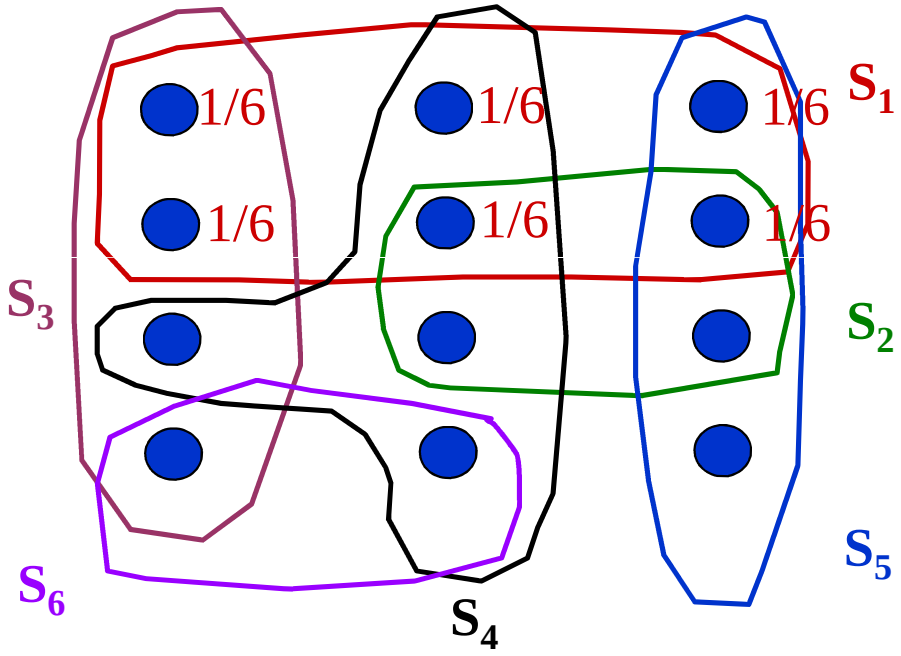
If $x \in X$ is covered for the first time by S_i, then

$$c_x = \frac{1}{|S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})\}|}$$

cost of x — no. of elements covered for the 1st time by S_i

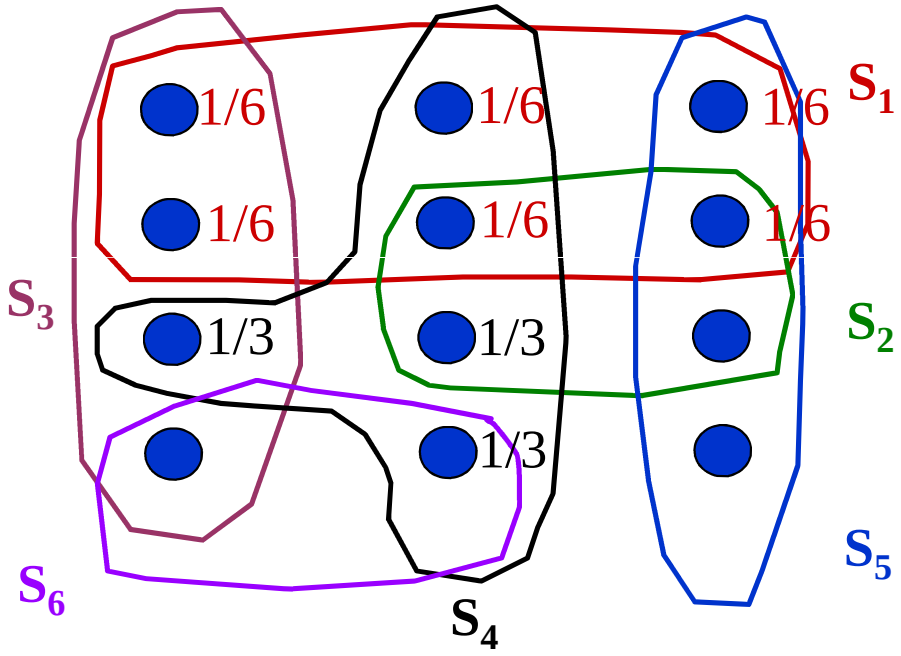
no of elements covered before step i.

Example: S1 has cost 1/6 per 6 elements covered:



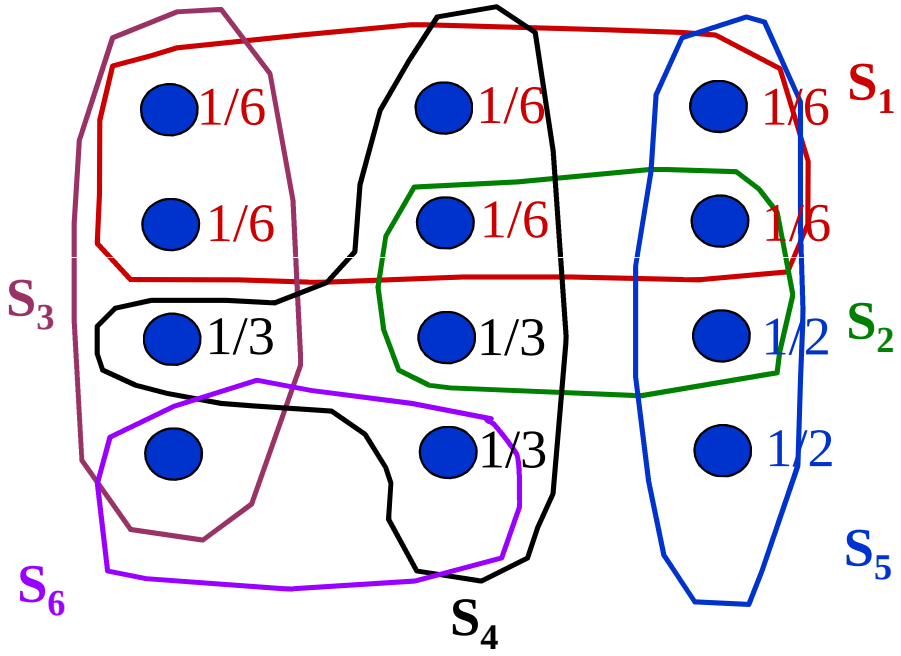
Computed cover consists of 4 sets.

Example: S3 has cost 1/3 per 3 new elements covered



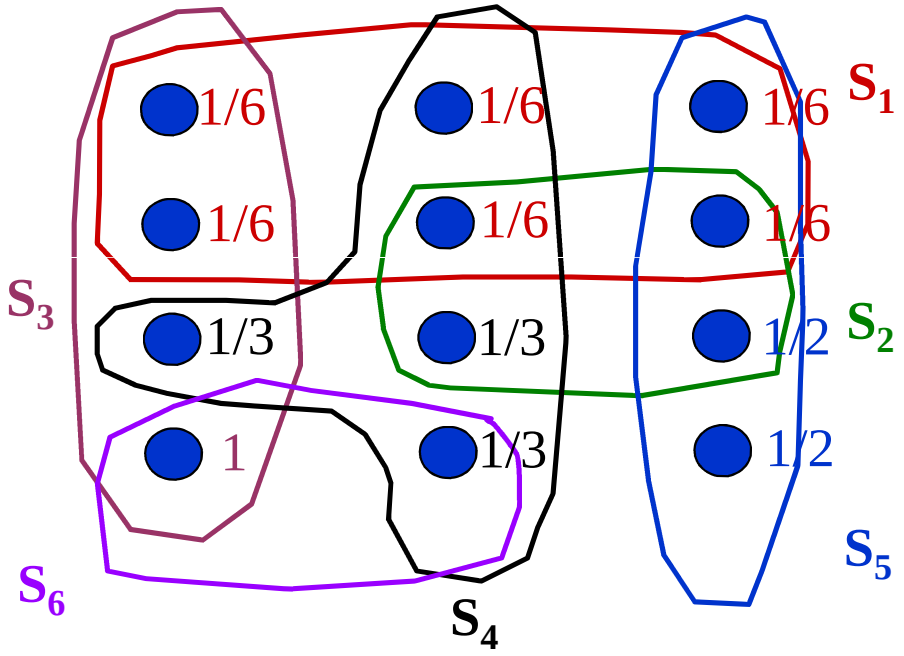
Computed cover consists of 4 sets.

Example S2 has cost 1/2 for 2 new elements covered



Computed cover consists of 4 sets.

Example: S4 has cost 1 for the last element covered



Computed cover consists of 4 sets = {S1,S3,S2,S4}

Proof continued

We have the following bound on |C|:

$$|C| = \sum_{x \in X} c_x$$

$$\leq \sum_{S \in C^*} \sum_{x \in S} c_x$$

This follows because $x \in X$ may appear in multiple sets in C*.
And each such x appears in at least one such set.(See example.)

Proof Continued

Claim: For any S in F, total cost

$$\sum_{x \in S} c_x \leq H(|S|)$$

By claim:

$$|C| \leq \sum_{S \in C^*} H(|S|)$$

$$\leq |C^*| \cdot H(\max\{|s|: s \in F\})$$

So, approx. ratio is as claimed.

Proof of Claim

Consider any $S \in F$.

For $i = 1,2, \ldots, |C|$, let

$u_i = | S - (S_1 \cup S_2 \cup \ldots \cup S_i)|$
= no. of uncovered elements in S after $S_1,\ldots,S_i$ have been selected.

Let $u_0 = |S|$.

Let k be s.t. S completely covered after S_k.

$u_{i-1} - u_i$ = no. of elements of S covered for the first time by S_i.

Proof of Claim

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{|S_i - (S_1 \cup \cdots \cup S_{i-1})|}$$

Greedy Choice $\Rightarrow$

$$|S_i - (S_1 \cup \cdots \cup S_{i-1})| \geq |S - (S_1 \cup \cdots \cup S_{i-1})| = u_{i-1}$$

Thus,

$$\sum_{x \in S} c_x \leq \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{u_{i-1}}$$

For $a < b$,

$$H(b) - H(a) = \sum_{i=a+1}^{b} \frac{1}{i} \geq (b-a)\frac{1}{b}$$

(For $a = b > 0$, $H(b) - H(a) = (b-a)(1/b)$.)

Thus,

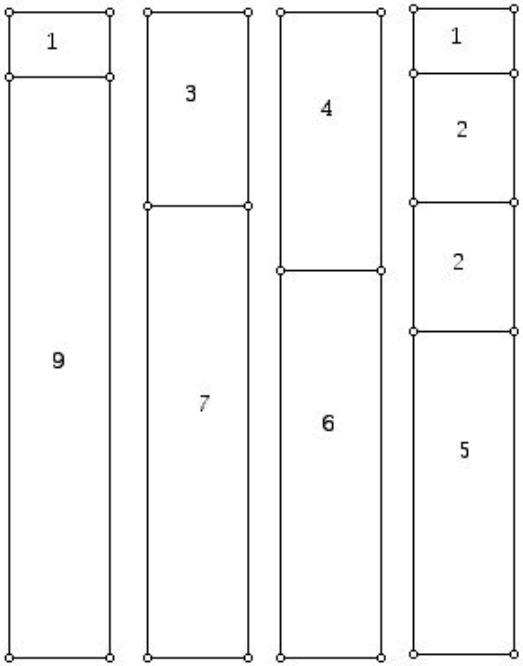
$$\begin{aligned} \sum_{x \in S} c_x &\leq \sum_{i=1}^{k} (H(u_{i-1}) - H(u_i)) \\ &= H(u_0) - H(u_k) \\ &= H(u_0) - H(0) \\ &= H(u_0) \\ &= H(|S|) \end{aligned}$$

Aproximation Algorithm for Subset sum

Cormen chapter 35

Recall: Bin packing problem is NPC

Can k bins of capacity t each, store a finite set of weights S = {x1,..,xn}?



Example:
We have k=4 bins with capacity t=10 each.
and weights S = {1,9,3,7,6,4,1,2,2,5}.
How to pack S into k bins?

Subset-Sum Problem

Problem: Pack as many of S into box of size t, without overflow.

ie. Find one subset of $S=\{x_1, \ldots, x_n\}$ (positive integers) whose sum is as large as possible but not larger than the capacity t.

Subset-Sum Brute force Exp Algorithm

Notation:

If L = [y1,…,yk], then L + x means [y1 + x, … ,yk + x].

If S is a set, then S + x means { s + x: s in S}.

```
Exact-SubsetSum(S, t)   // Exp because Length(L) can be 2^i.
    n = |S|
    L_0 = [0]
    for i = 1 to n do
        L_i = Merge (L_i-1, L_i-1 + x_i );   /* like Merge Sort */
        Remove from L_i every element  > t
    return largest element in L_n.
```

Example

L_i = Merge (L_{i-1}, L_{i-1} + x_i); /* like Merge Sort */

E.g. if L1 = [0, 1, 2] and x2 = 10 then

L2 = Merge(L1, L1 + 10)

L2 = Merge([0,1,2], [10,11,12])

L2 = [0, 1, 2, 10, 11, 12]

So the list size can double in each step

A Fully-PT-Approximation-Scheme

Goal: Modify exponential algorithm to make it a fully polynomial-time approximation scheme.

Idea: "Trim" elements from L_i that are near other elements.
Given: trim parameter δ, $0 < \delta < 1$. (will depend on ε.)
Remove y from list if there is z nearby it: $\exists\ z : y/(1+\delta) \le z \le y$.

Example: If $\delta = 0.1$ and L = [10,11,12,15,20,21,22,23,24,29],
trim(L) = L′ = [10,12,15,20,23,29].

Intuition: Deleted values 21 & 22 are approximated by 20.

Trim the list L of nearby numbers

```
Trim (L, δ)          // L = [y1, y2, …, ym]
    m =|L|;
    L′  = [y1];
    last = y1;
    for i = 2 to m do
        if  yi > last·(1 + δ) then
            append yi onto end of L′;
            last = yi
        fi
    od
    return L′
```

Approx Subset Sum

Example:

S = {104,102,201,101}, $\varepsilon = 0.40$, $\delta = 0.05$

t = 308

line 2: L_0 = <0>

line 4: L_1 = <0, 104>
line 5: L_1 = <0, 104>
line 6: L_1 = <0, 104>

line 4: L_2 = <0, 102, 104, 206>
line 5: L_2 = <0, 102, 206>
line 6: L_2 = <0, 102, 206>

line 4: L_3 = <0, 102, 201, 206, 303, 407>
line 5: L_3 = <0, 102, 201, 303, 407>
line 6: L_3 = <0, 102, 201, 303>

line 4: L_4 = <0, 101, 102, 201, 203, 302, 303, 404>
line 5: L_4 = <0, 101, 201, 302, 404>
line 6: L_4 = <0, 101, 201, 302>

Algorithm returns 302.

Optimal answer is 307 = 104 + 102 + 101.

```
Approx-Subset-Sum(S, t, ε)
1   n = |S|;
2   L0 = [0];
3   for I =1 to n do
4       Li = Merge (Li-1, Li-1+xi);
5       Li = Trim (Li, ε/2n);
6       remove from Li every element > t
    od;
7   return largest element in Ln
```

Theorem: Approx-SS is a fully-PTAS.

Proof: We consider the solution space Pi at step i.
Let P_i = set of all values that can be obtained by selecting a (possibly empty) subset of $\{x_1, x_2, \ldots, x_i\}$ and summing its members.

In Exact-SS, $L_i = P_i$ (with items > t removed).

In Approx-SS, ✂ $y \in P_i$ (y ✪ t),
$\exists\, z \in L_i$: $y/(1+ \varepsilon/2n)^i \leq z \leq y$.

Proof: By induction on i.

Proof continued

Let $y^* \in P_n$ denote the optimal solution, and let z^* denote the solution returned by SSAA. Then, $z^* \leq y^*$.
TPT. $\mathbf{y^*/z^* \leq 1 + \varepsilon}$.

By previous observation, $\exists\ z \in L_n$ s.t.
$y^*/(1+ \varepsilon/2n)^n \leq\ z\ \leq\ y^*$, and hence, $y^*/z \leq (1+ \varepsilon/2n)^n$.
Because z^* is the largest value in L_n, we have $y^*/z^* \leq (1+ \varepsilon/2n)^n$.

$d/dn\ (1+ \varepsilon/2n)^n > 0$, so $(1+ \varepsilon/2n)^n$ increases with n. Its limit is $e^{\varepsilon/2}$.
Thus, $(1+ \varepsilon/2n)^n \leq e^{\varepsilon/2}$

$\leq 1 + \varepsilon/2 + (\varepsilon/2)^2$ $\quad \{e^x \leq 1 + x + x^2$ — see Ch. 3$\}$

$\leq 1 + \varepsilon$ $\quad \{0 < \varepsilon < 1\}$

Proof: Time complexity

Show time complexity is polynomial in length[I] and $1/\varepsilon$.
Want to bound the length of L_i.
After trimming, successive z and z′ satisfy $z'/z > 1 + \varepsilon/2n$.
Thus, each list contains [0 to $\lfloor \log_{1+\varepsilon/2n} t \rfloor$] other values.
So, time complexity is polynomial in length(I) and $1/\varepsilon$.
Because:

$$\log_{1+\varepsilon/2n} t = \frac{\ln t}{\ln(1+\varepsilon/2n)}$$

$$\leq \frac{2n(1+\varepsilon/2n)\ln t}{\varepsilon} \qquad \{x/(1+x) \leq \ln(1+x) \text{ -- see Ch. 3}\}$$

$$\leq \frac{4n \ln t}{\varepsilon} \qquad \{0 < \varepsilon < 1\}$$

TSP: Travelling salesman problem

TSP

Given:

Set of N cities = $\{c_1, c_2, \ldots, c_N\}$.

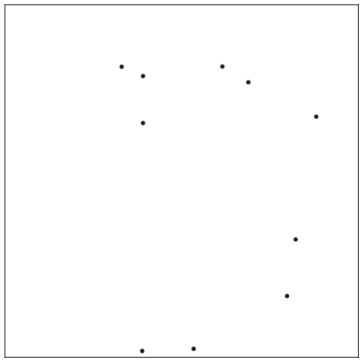
For each pair of cities $\{c_i, c_j\}$, a distance $d(c_i, c_j)$.

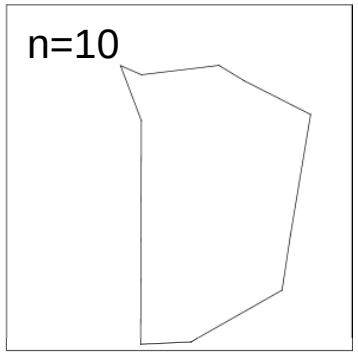
Find: Permutation Pi of {1..n}

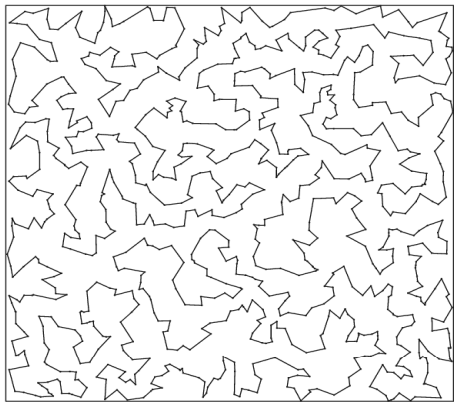
that minimizes the total

$$\sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(N)}, c_{\pi(1)})$$

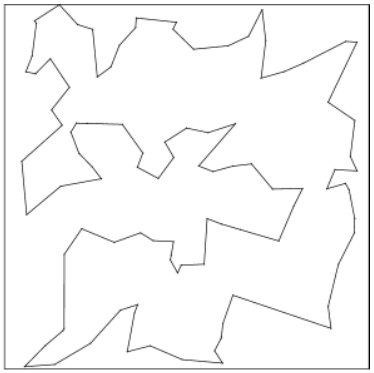
Example of TSP instances



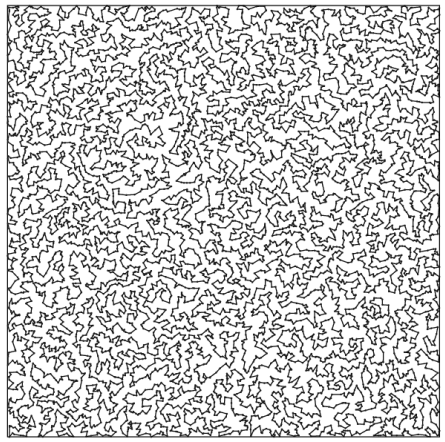




n=1000



n=100



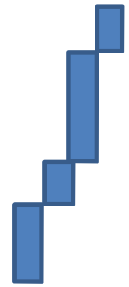
n=10000

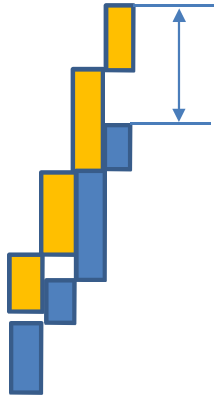
Other Instances of TSP

- X-ray crystallography
  - Cities: orientations of a crystal
  - Distances: time for motors to rotate the crystal from one orientation to the other
- High-definition video compression
  - Cities: binary vectors of length 64 identifying the summands for a particular function
  - Distances: Hamming distance (the number of terms that need to be added/subtracted to get the next sum)

No-Wait Floor shop Scheduling

- Cities: Length-4 vectors $<c_1,c_2,c_3,c_4>$ of integer task lengths for a given job that consists of tasks that require 4 processors that must be used in order, where the task on processor i+1 must start as soon as the task on processor i is done).
- Distances: d(c,c') = Increase in the finish time of the 4^{th} processor if c' is run immediately after c.
- Note: Not necessarily symmetric: may have $d(c,c') \neq d(c',c)$.





How Hard is TSP?

- NP-Hard for all the above applications and many more
  - [Karp, 1972]
  - [Papadimitriou & Steiglitz, 1976]
  - [Garey, Graham, & Johnson, 1976]
  - [Papadimitriou & Kanellakis, 1978]

How Hard?

Number of possible tours:

$N! = 1 \times 2 \times 3 \times \cdots \times (N-1) \times N = \Theta(2^{N \log N})$

$10! = 3{,}628{,}200$

$20! \sim 10^{18}$ (quadrillion)

Dynamic Programming Solution:

$O(N^2 * 2^N) = o(2\text{^}(N*\log N))$

Dynamic Programming Algorithm

- For each subset C' of the cities containing c_1, and each city $c \in C'$,
- let f(C',c) = Length of shortest path from c1 to c that is a permutation of C'. So $f(\{c_1\}, c_1) = 0$
- For $x \notin C'$, $f(C' \cup \{x\},x) = \text{Min}_{c \in C'} f(C',c) + d(c,x)$.
- Optimal tour length $= \text{Min}_{c \in C} f(C,c) + d(c, c_1)$.
- Running time: ~(N-1)2^(N-1) items to be computed, at time N for each = $O(N^2 2^N)$

DP is how Hard?

Number of possible tours:

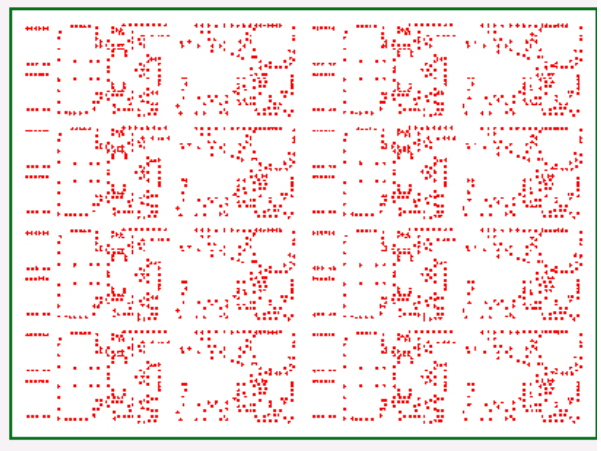
- $N! = 1 \times 2 \times 3 \times \cdots \times (N-1) \times N = \Theta(2^{N \log N})$
- $10! = 3{,}628{,}200$
- $20! \sim 2.43 \times 10^{18}$ (2.43 quadrillion)

Dynamic Programming Solution:

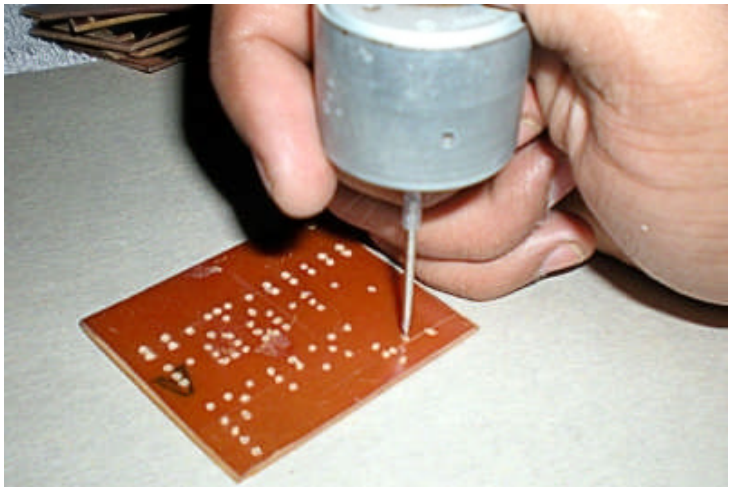
- $O(N^2 2^N)$
- $10^2 2^{10} = 102{,}400$
- $20^2 2^{20} = 419{,}430{,}400$

Planar Euclidean Application #1

- Cities: Holes to be drilled in printed circuit boards

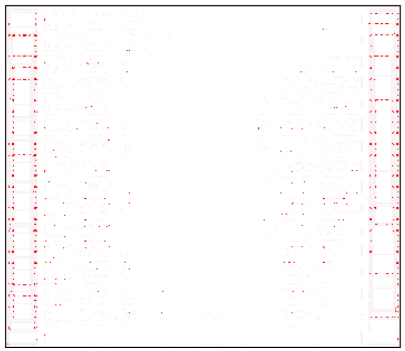


N = 2392

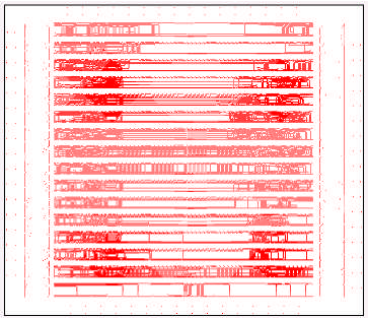


Planar Euclidean Application #2

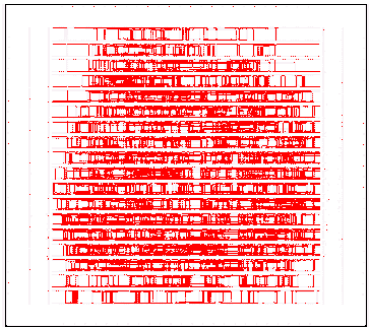
Cities: Wires to be cut in a “Laser Logic” programmable circuit



N = 7397



N = 33,810



N = 85,900

Standard Approach to Coping with NP-Hardness:

- Approximation Algorithms
  - Run quickly (polynomial-time for theory, low-order polynomial time for practice)
  - Obtain solutions that are guaranteed to be close to optimal
  - Euclidean TSP, the triangle inequality holds: $d(a,c) \leq d(a,b) + d(b,c)$

Travelling Salesman Problem (TSP)

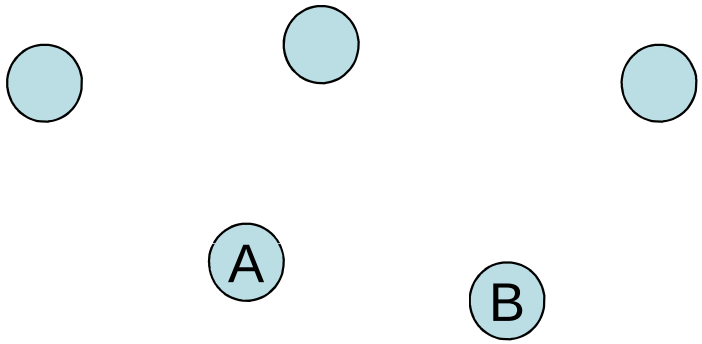
Animated slides

A Salesman wishes to travel around a given set of cities, and return to the beginning, covering the smallest total distance

Easy to State

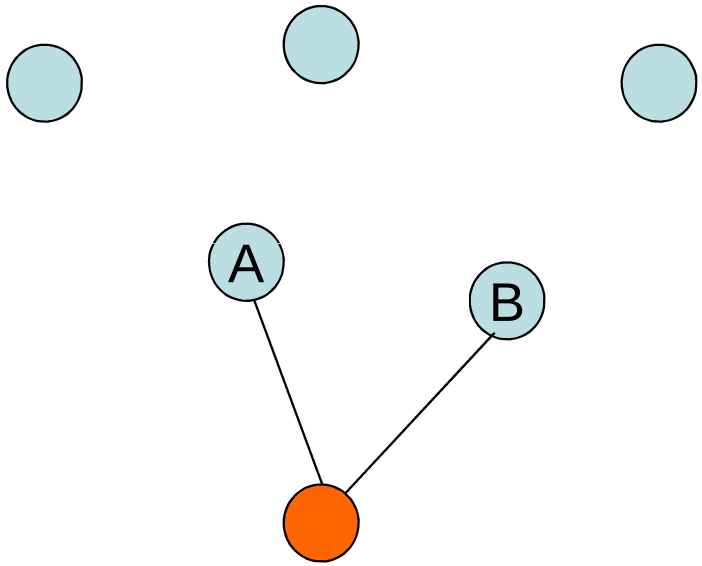
Difficult to Solve

If there is no condition to return to the beginning.
It can still be regarded as a TSP.
Suppose we wish to go from A to B visiting all cities.



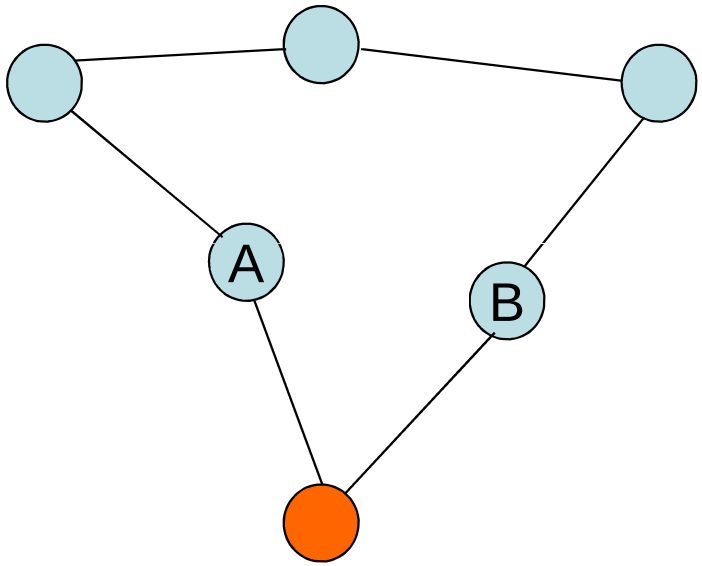
If there is no condition to return to the beginning. It can still be regarded as a TSP.

Connect A and B to a ‘dummy’ city at zero distance
(If no stipulation of start and finish cities connect all to dummy at zero distance)



If there is no condition to return to the beginning. It can still be regarded as a TSP.

Create a TSP Tour around all cities



A route returning to the beginning is known as a
Hamiltonian Circuit

A route not returning to the beginning is known as a
Hamiltonian Path

Essentially the same class of problem

Applications of the TSP

Routing around Cities

Computer Wiring - connecting together computer components using minimum wire length

Archaeological Seriation - ordering sites in time

Genome Sequencing - arranging DNA fragments in sequence

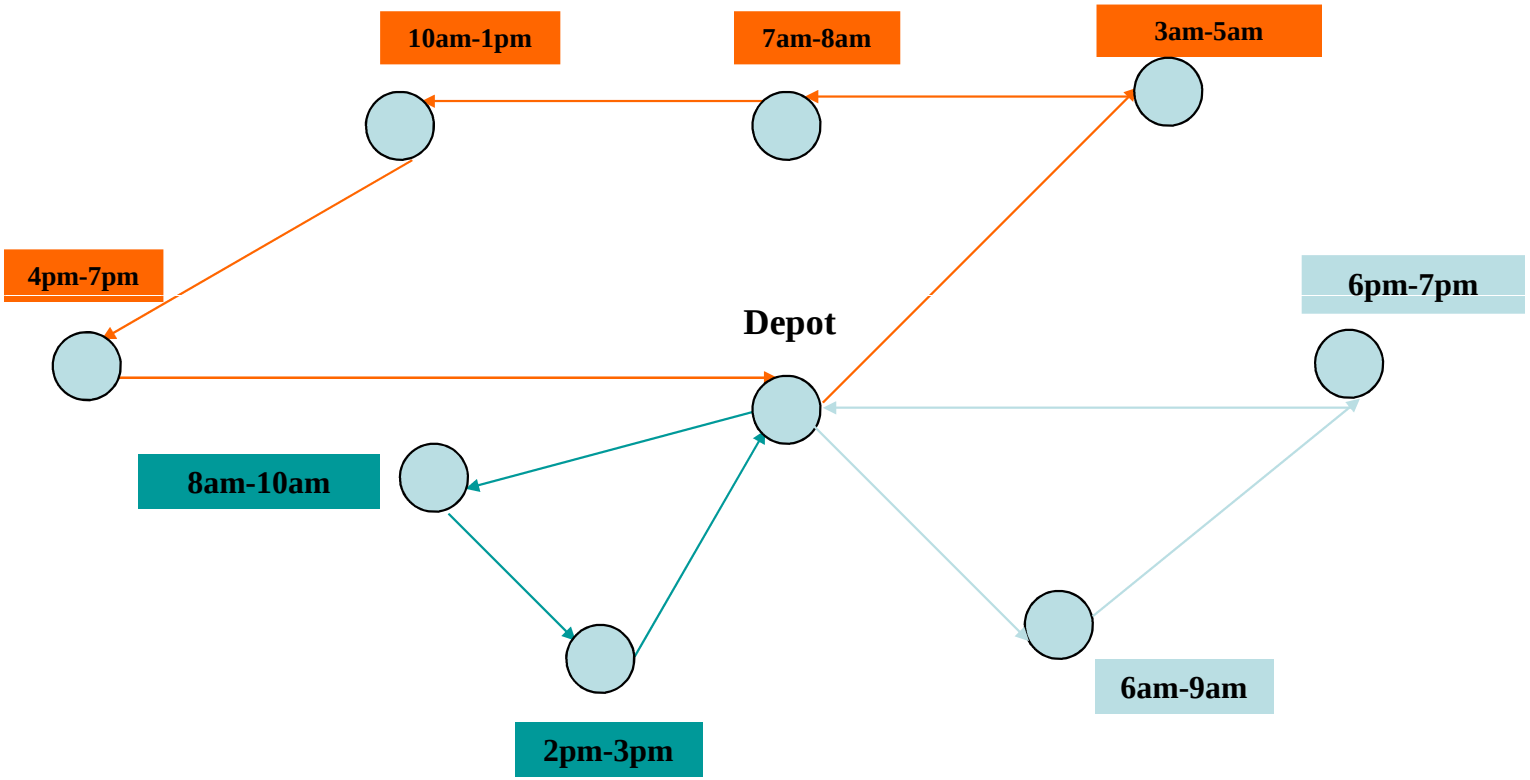
Job Sequencing - sequencing jobs in order to minimise total set-up time between jobs

Wallpapering to Minimise Waste

Note: First three applications are *symmetric*, Last three *asymmetric*

Major Practical Extension of the TSP

Vehicle Routing - Meet customers demands within given *time windows* using lorries of limited capacity



Much more difficult than TSP

History of TSP

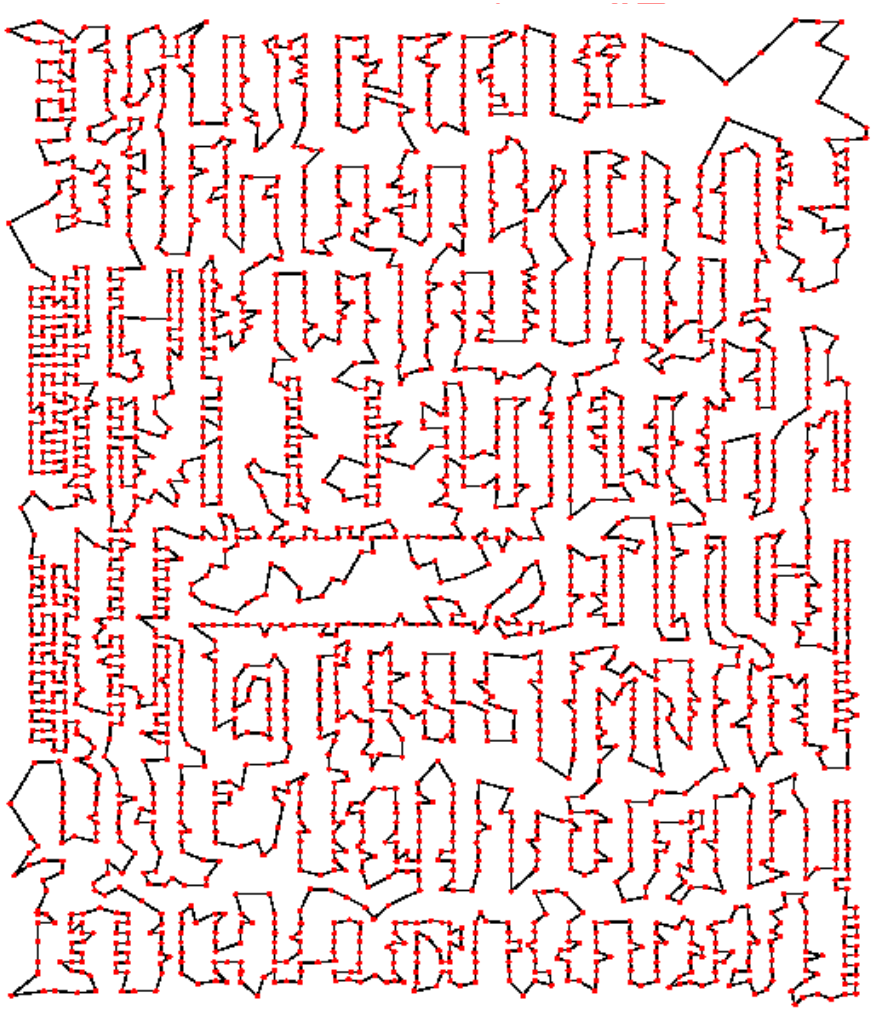
1800's	**Irish Mathematician, Sir William Rowan Hamilton**
1930's	**Studied by Mathematicians Menger, Whitney, Flood etc.**
1954	**Dantzig, Fulkerson, Johnson, 49 cities (capitals of USA states) problem solved**
1971	**64 Cities**
1975	**100 Cities**
1977	**120 Cities**
1980	**318 Cities**
1987	**666 Cities**
1987	**2392 Cities (Electronic Wiring Example)**
1994	**7397 Cities**
1998	**13509 Cities (all towns in the USA with population > 500)**
2001	**15112 Cities (towns in Germany)**
2004	**24978 Cities (places in Sweden)**

But many smaller instances not yet solved (to proven optimality)

Recent TSP Problems and Optimal Solutions

from William Cook, Georgia Tech

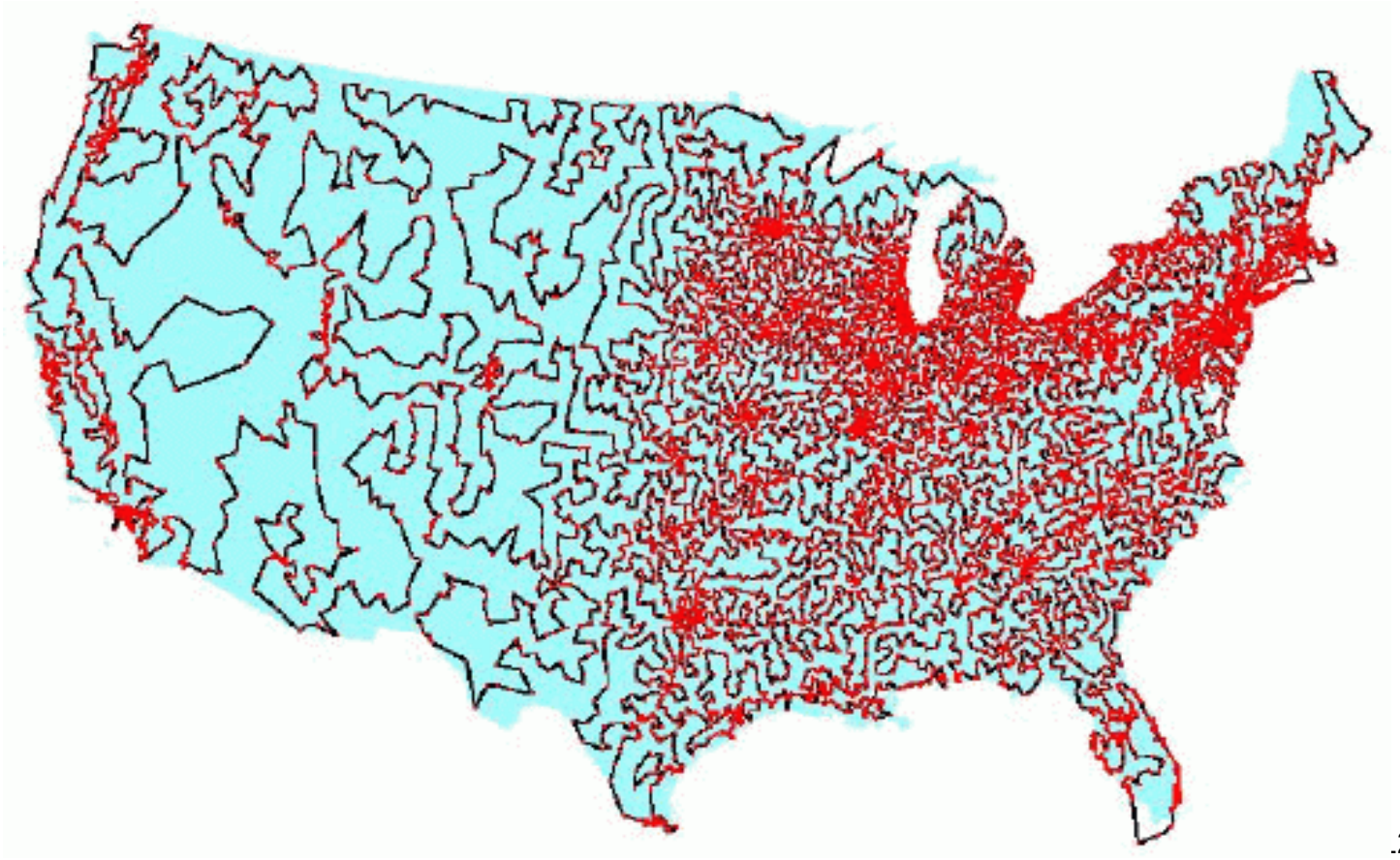
Printed Circuit Board 2392 cities **1987 Padberg and Rinaldi**



2004

n=24978

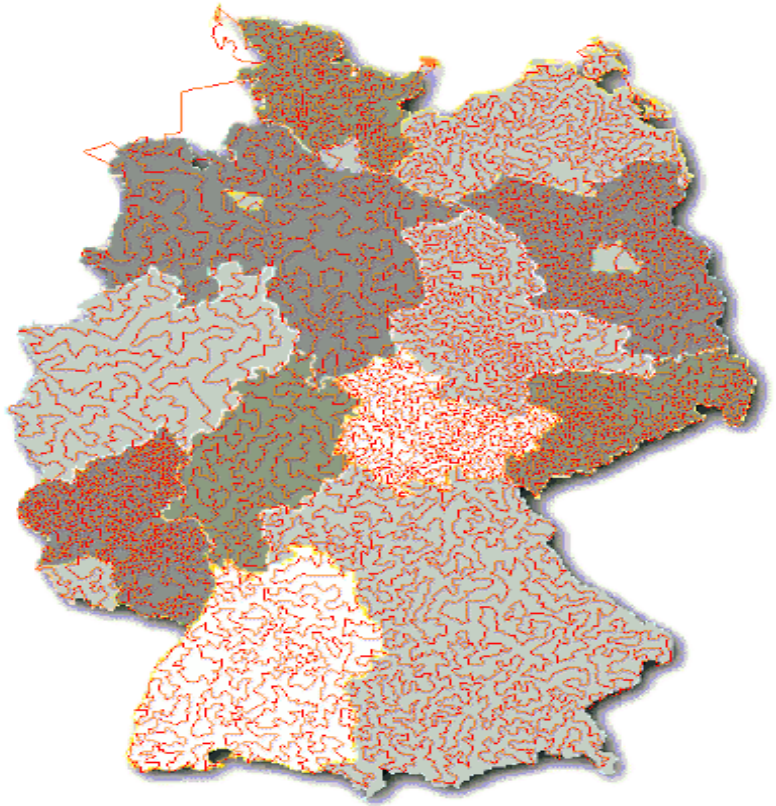
USA Towns of 500 or more population 13509 cities 1998 Applegate, Bixby, Chvátal and Cook



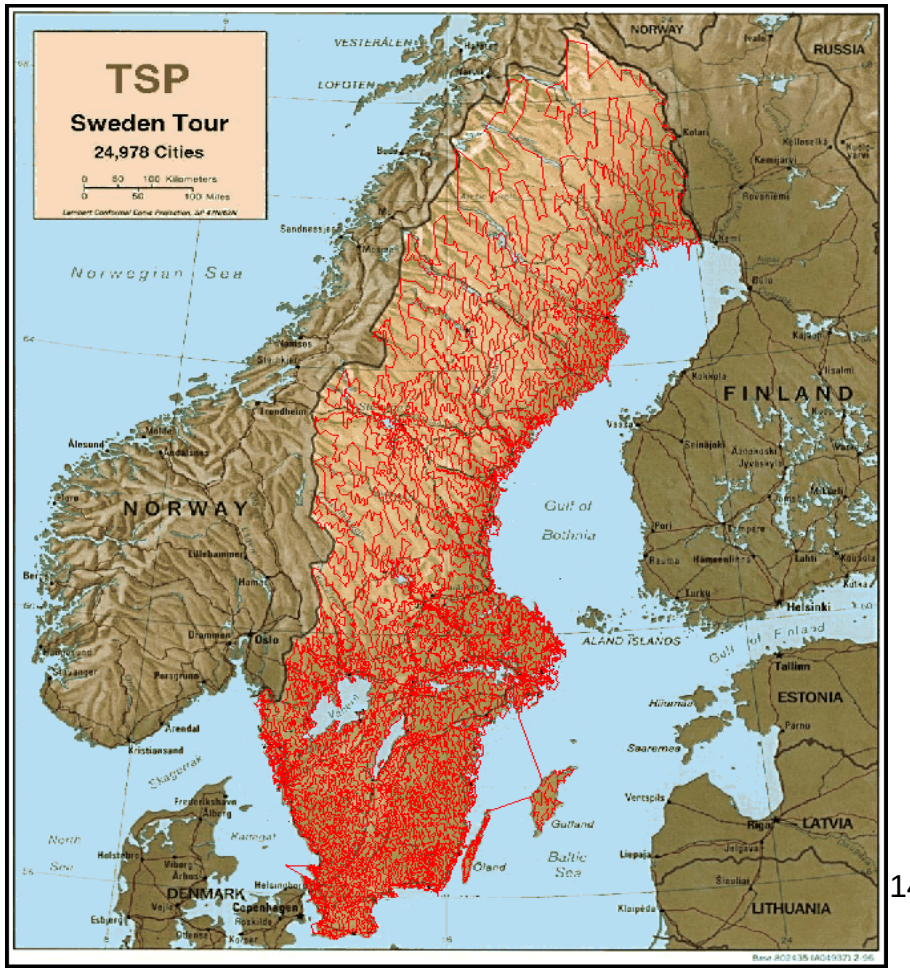
.2

Towns in Germany 15112 Cities

2001Applegate, Bixby, Chvátal and Cook



Sweden 24978 Cities 2004 Applegate, Bixby, Chvátal, Cook and Helsgaun



Solution Methods

I.	**Try every possibility**	(n-1)! possibilities – grows faster than exponentially

If it takes 1 micro-second / possibility, all possibilites for n=100 will take $10^{\wedge 140}$ centuries.

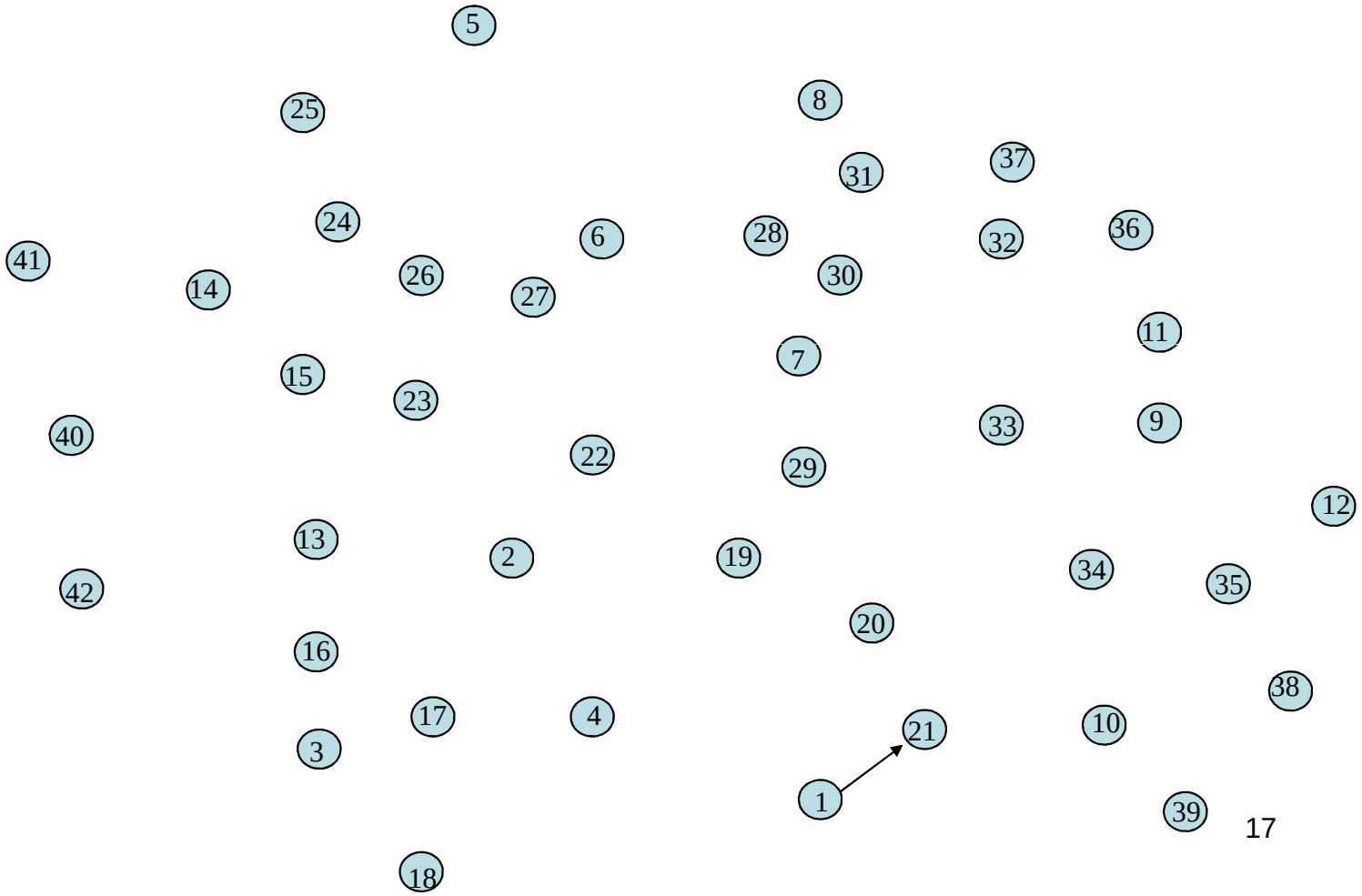
II.	**Optimising Methods**	obtain **guaranteed** optimal solution, but can take a very, very, long time
III.	**Heuristic Methods**	obtain ‘good’ solutions ‘quickly’ by intuitive methods. No guarantee of optimality

(Place problem in newspaper with cash prize)

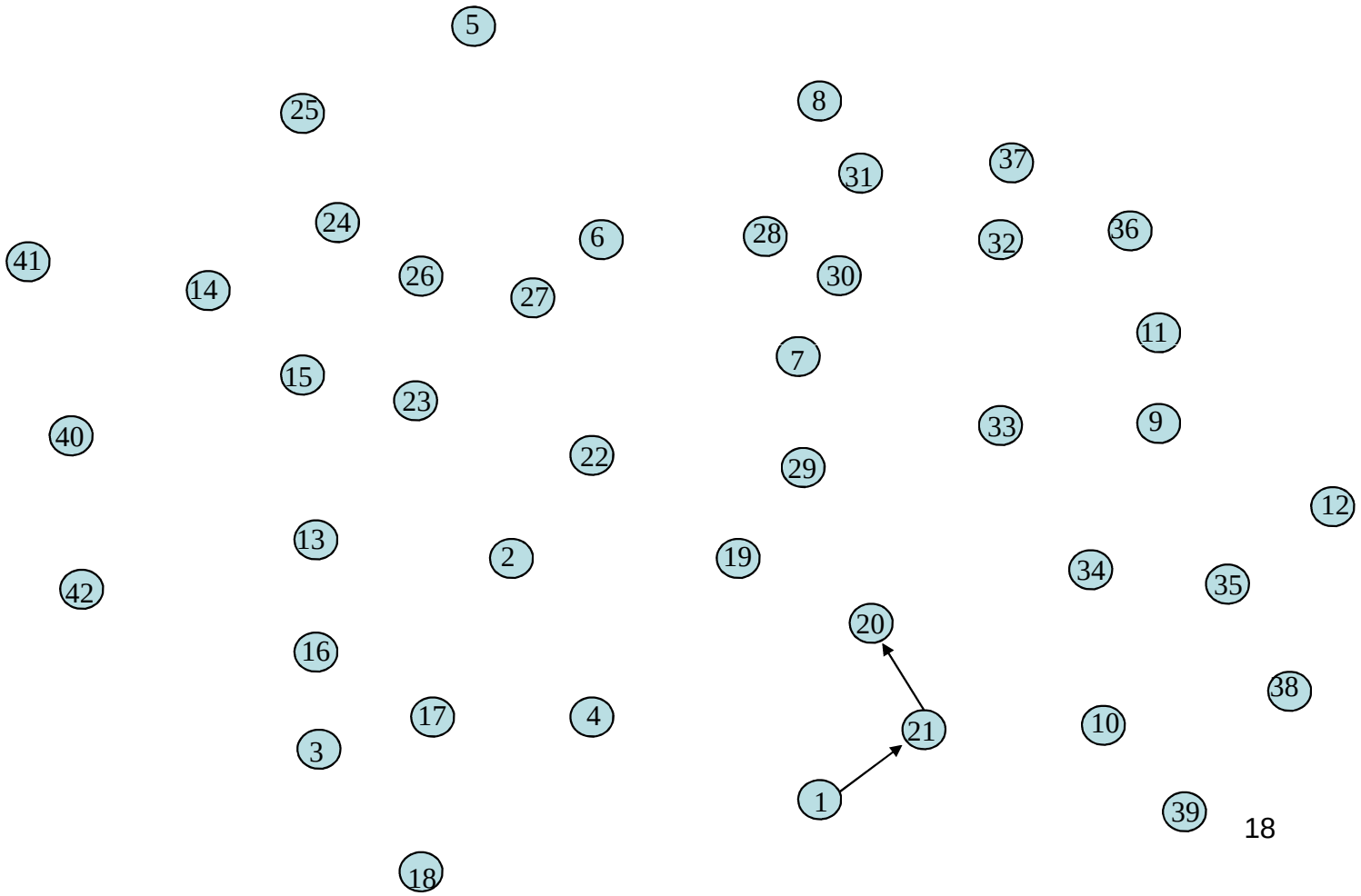
The Nearest Neighbour Method (Heuristic) – A ‘Greedy’ Method

1. Start Anywhere
2. Go to Nearest Unvisited City
3. Continue until all Cities visited
4. Return to Beginning

A 42-City Problem The Nearest Neighbour Method (Starting at City 1)

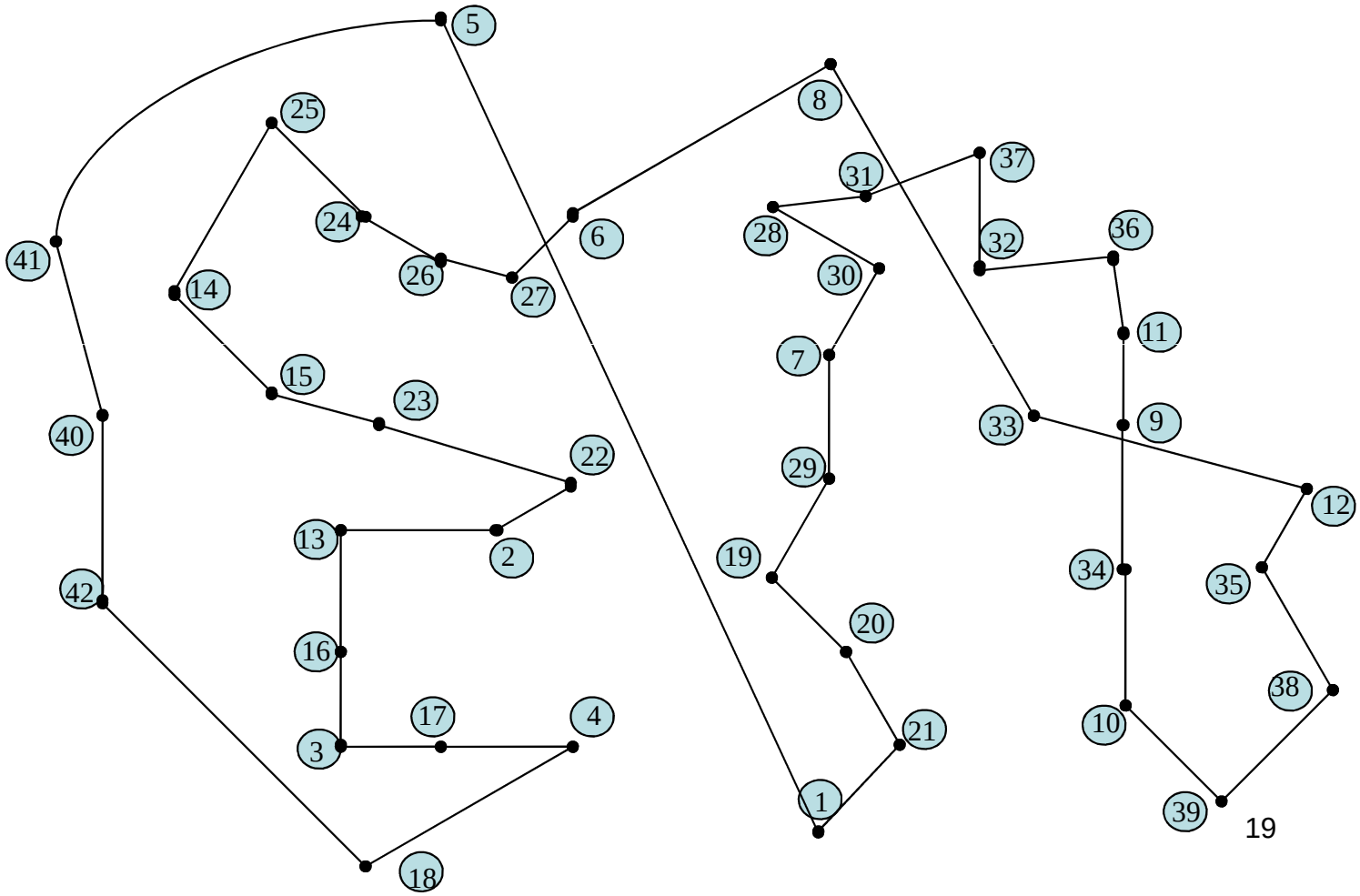


The Nearest Neighbour Method (Starting at City 1)

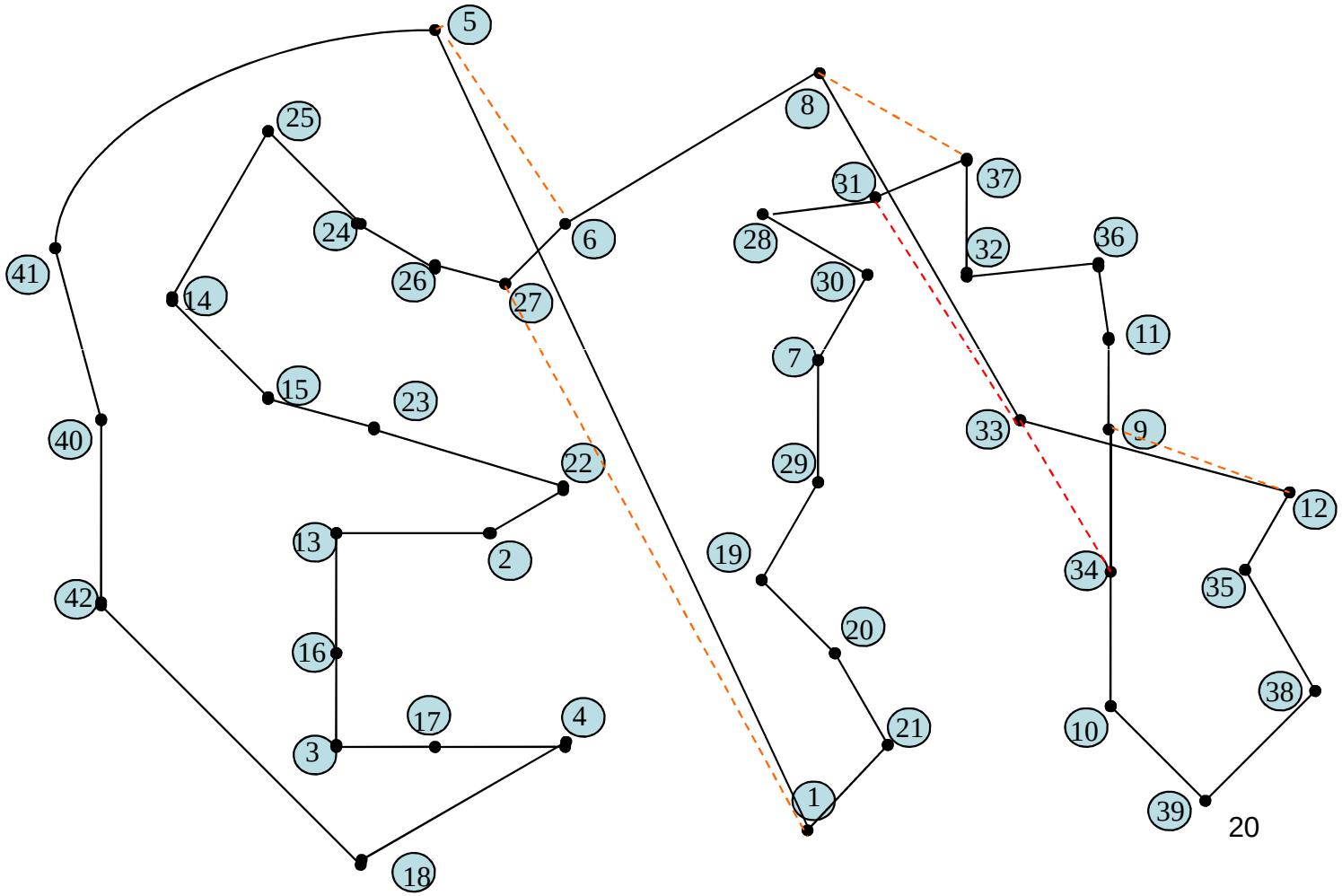


The Nearest Neighbour Method (Starting at City 1)

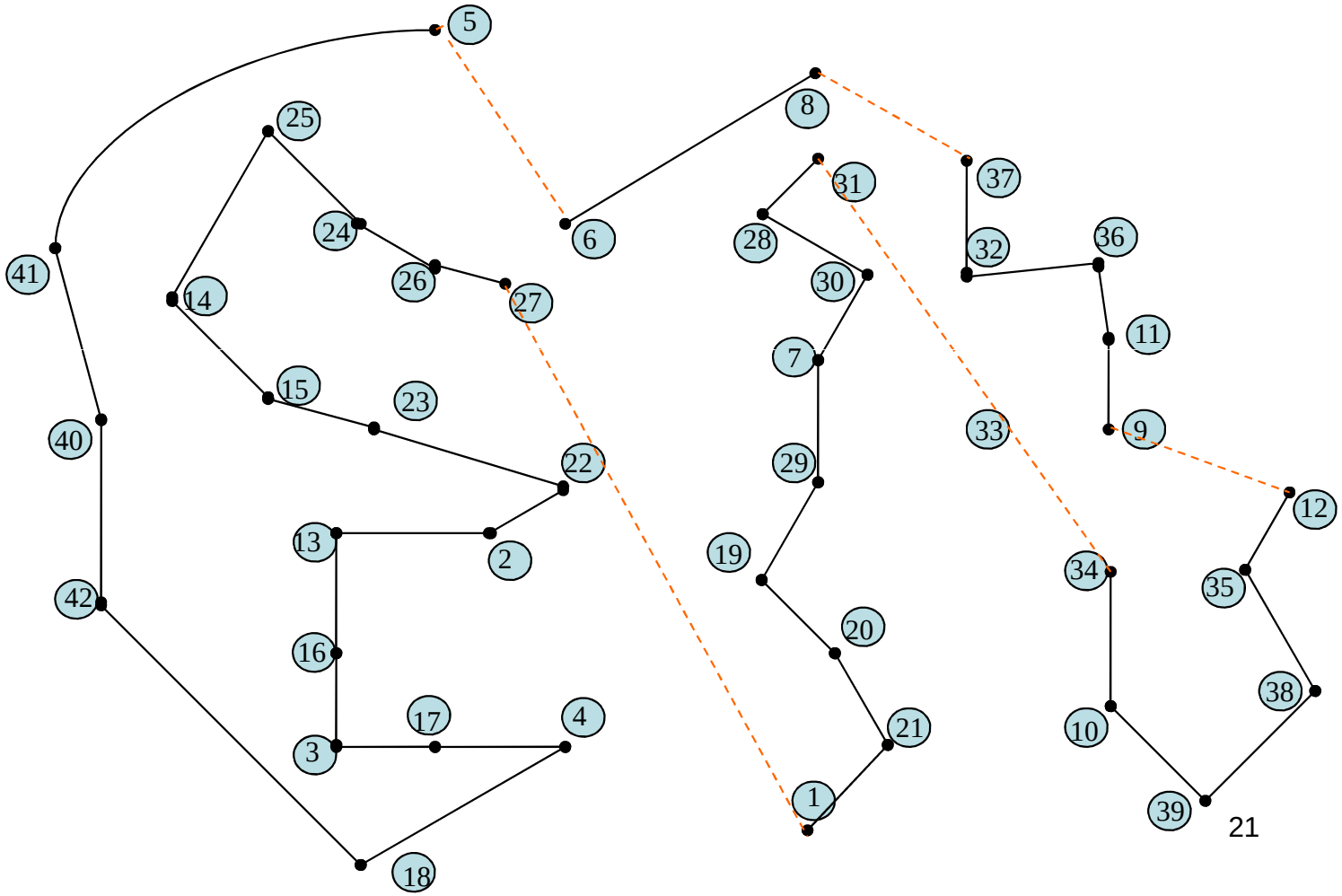
Length 1498



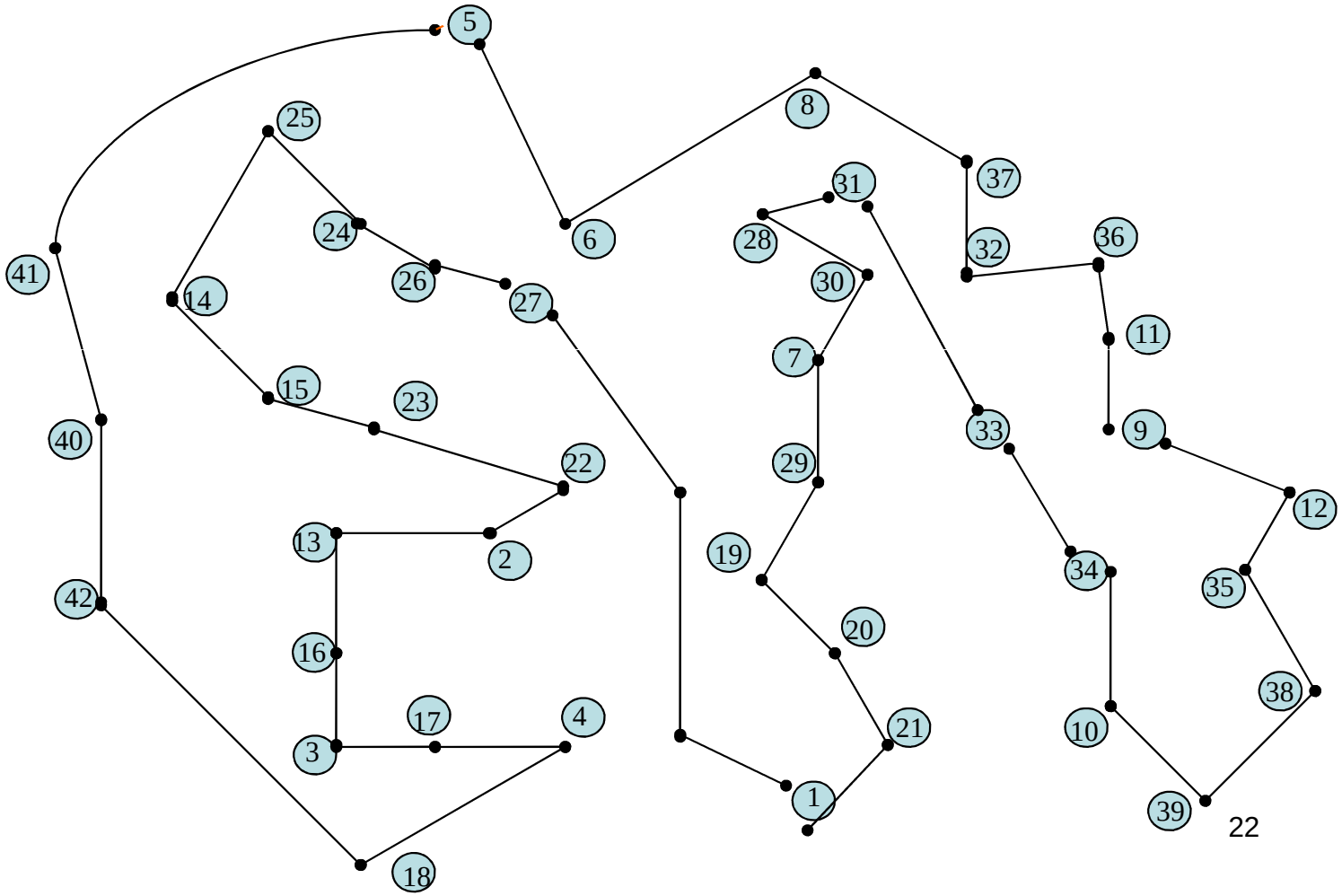
Remove Crossovers



Remove Crossovers



Remove Crossovers Length 1453



Christofides Method (Heuristic)

1. Create Minimum Cost Spanning Tree (Greedy Algorithm)

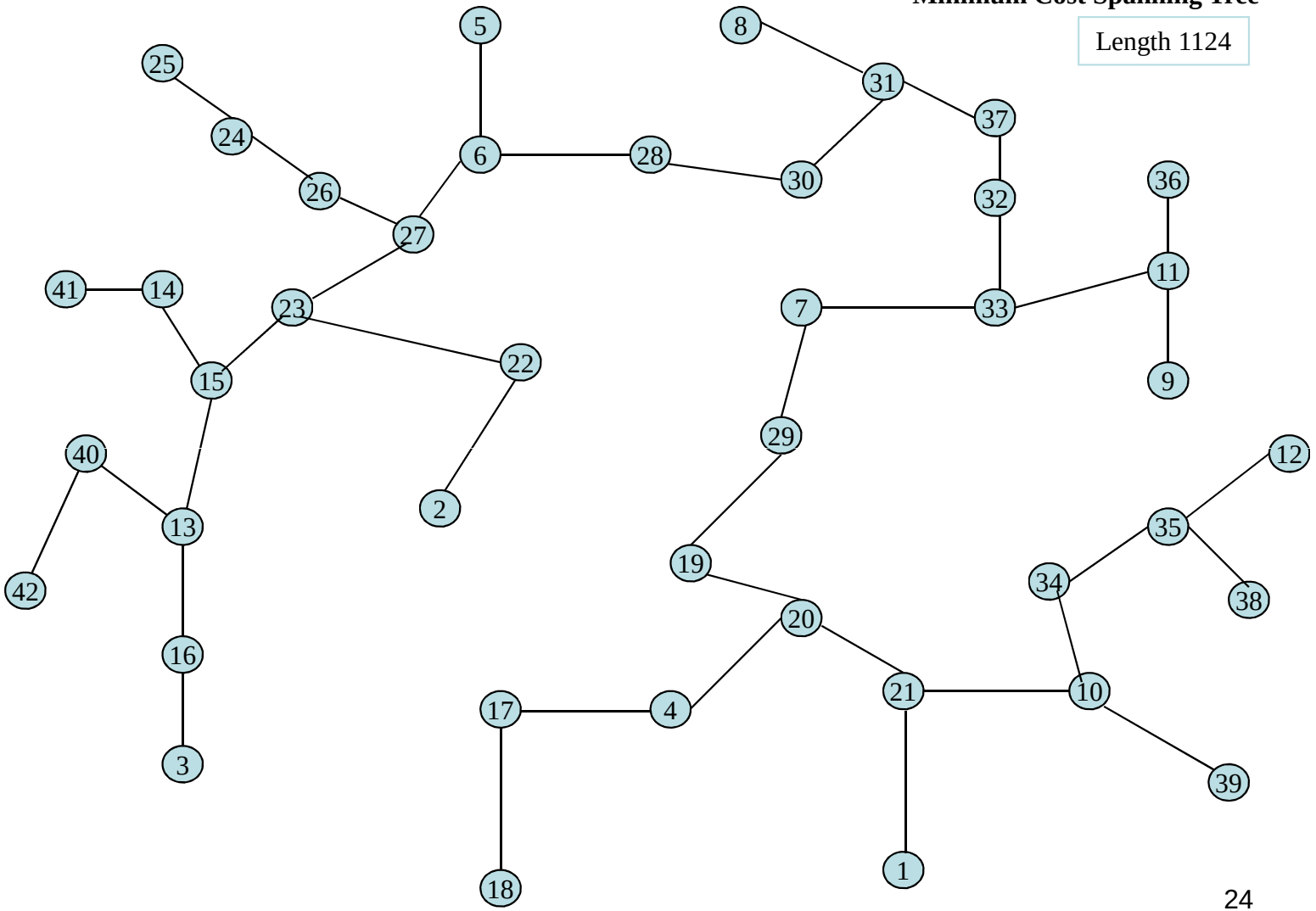
2. ‘Match’ Odd Degree Nodes

3. Create an Eulerian Tour - Short circuit cities revisited

Christofides Method 42 – City Problem

Minimum Cost Spanning Tree

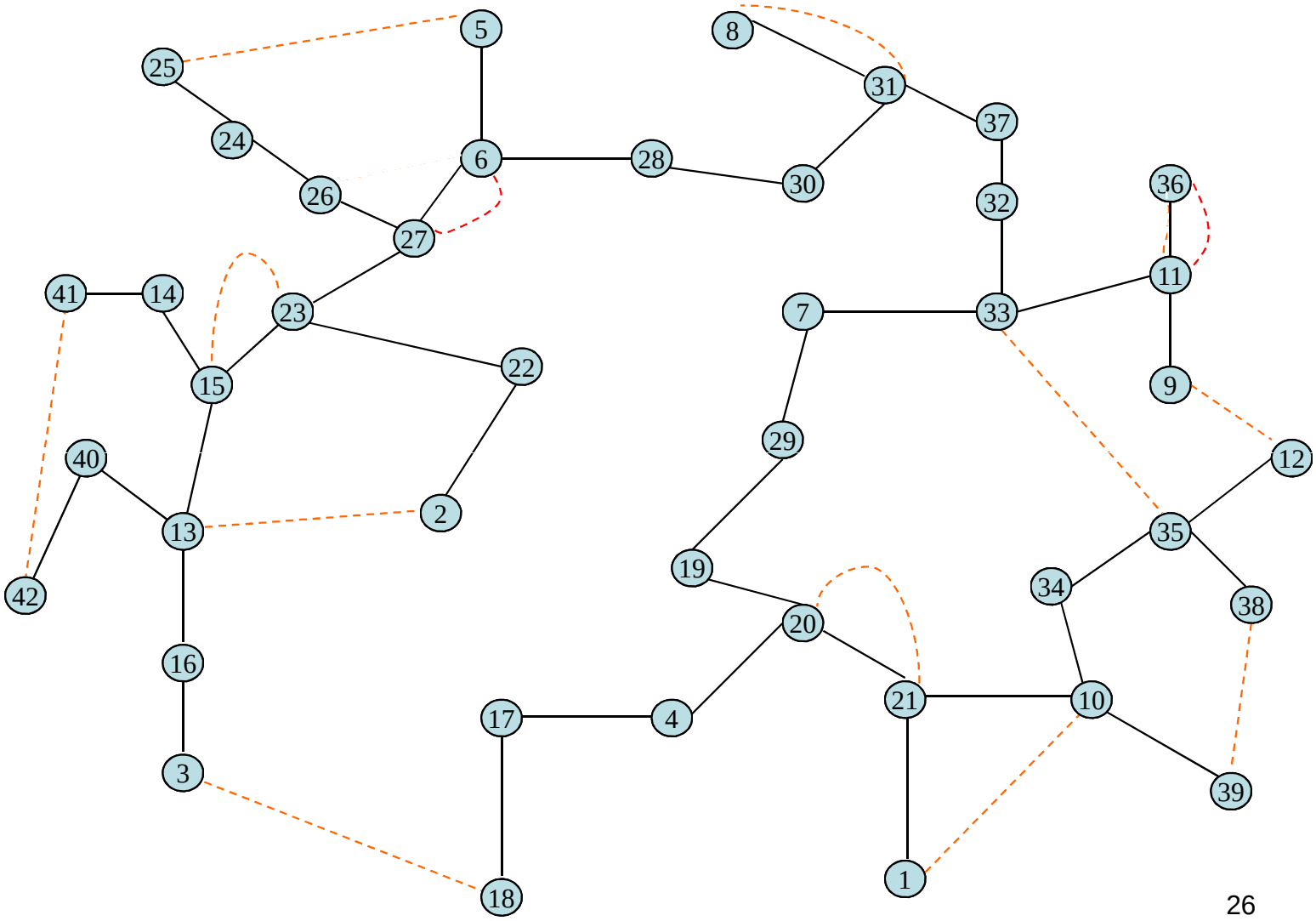
Length 1124



Minimum Cost Spanning Tree by Greedy Algorithm

Match Odd Degree Nodes

Match Odd Degree Nodes in Cheapest Way – Matching Problem



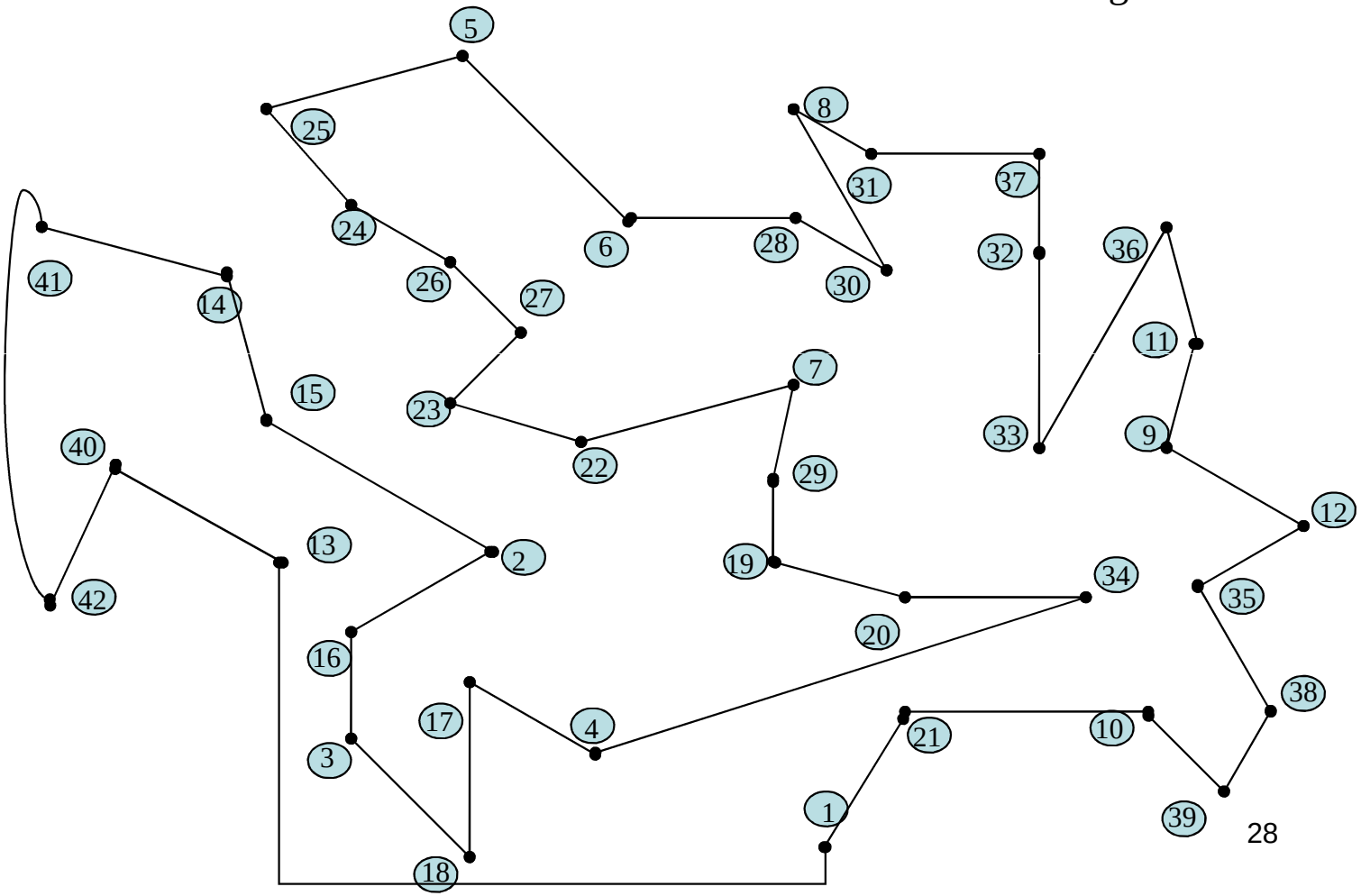
1. Create Minimum Cost Spanning Tree (Greedy Algorithm)

2. ‘Match’ Odd Degree Nodes

3. Create an Eulerian Tour - Short circuit cities revisited

Create a Eulerian Tour – Short Circuiting Cities revisited

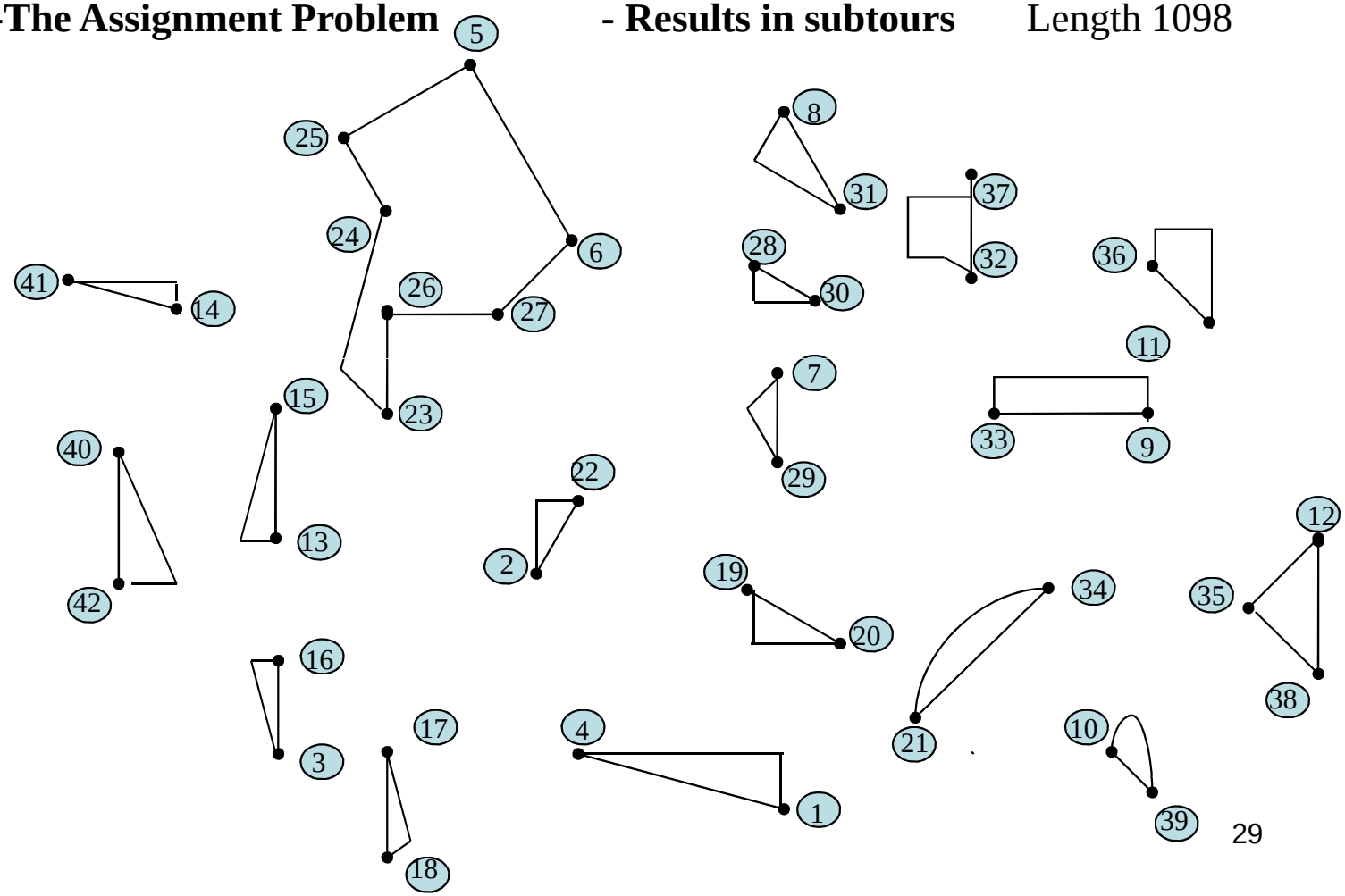
Length 1436



Optimising Method

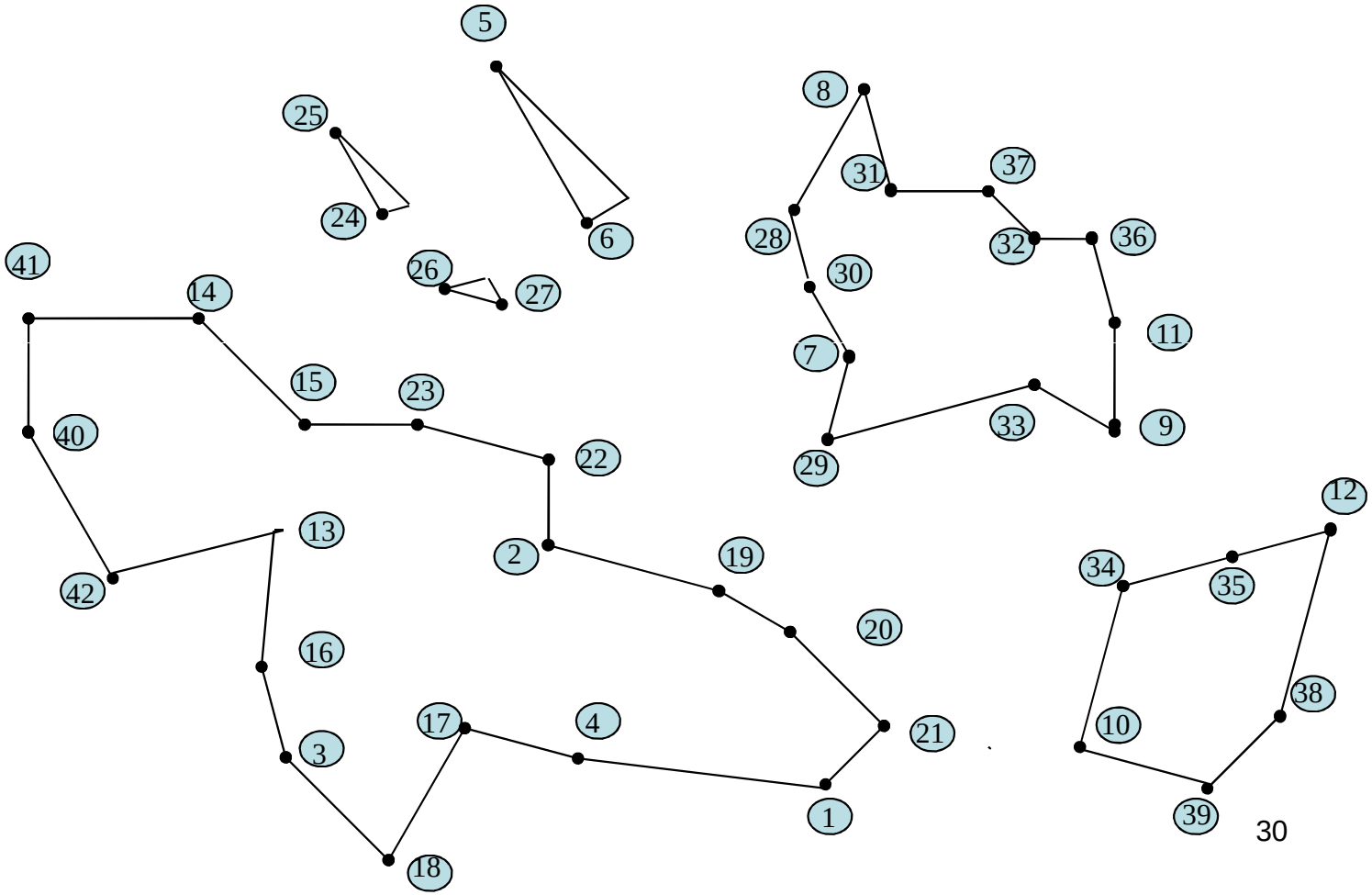
1.Make sure every city visited once and left once – in cheapest way (Easy)

-The Assignment Problem **- Results in subtours** Length 1098



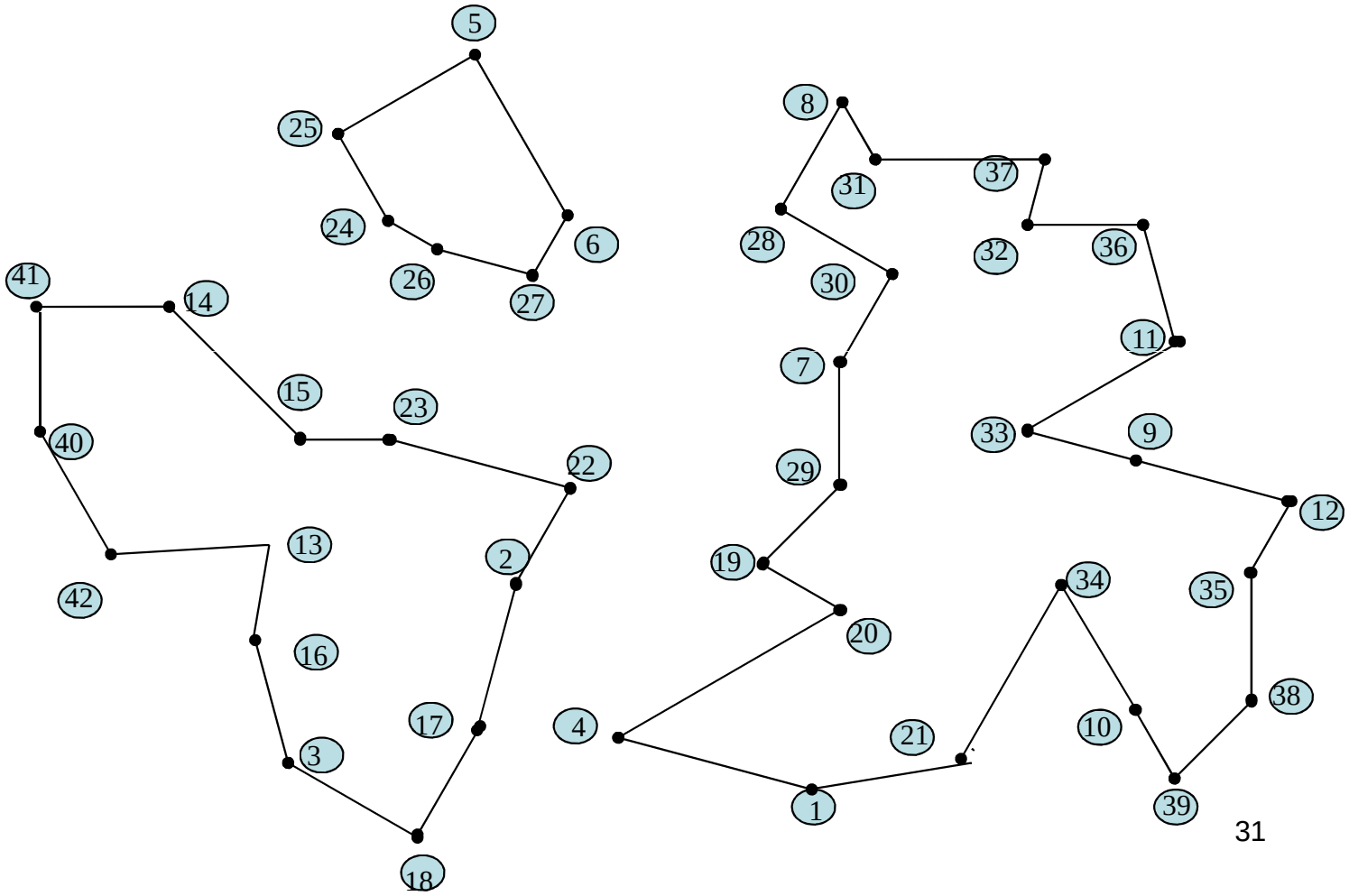
Put in extra constraints to remove subtours (More Difficult)

Results in new subtours Length 1154

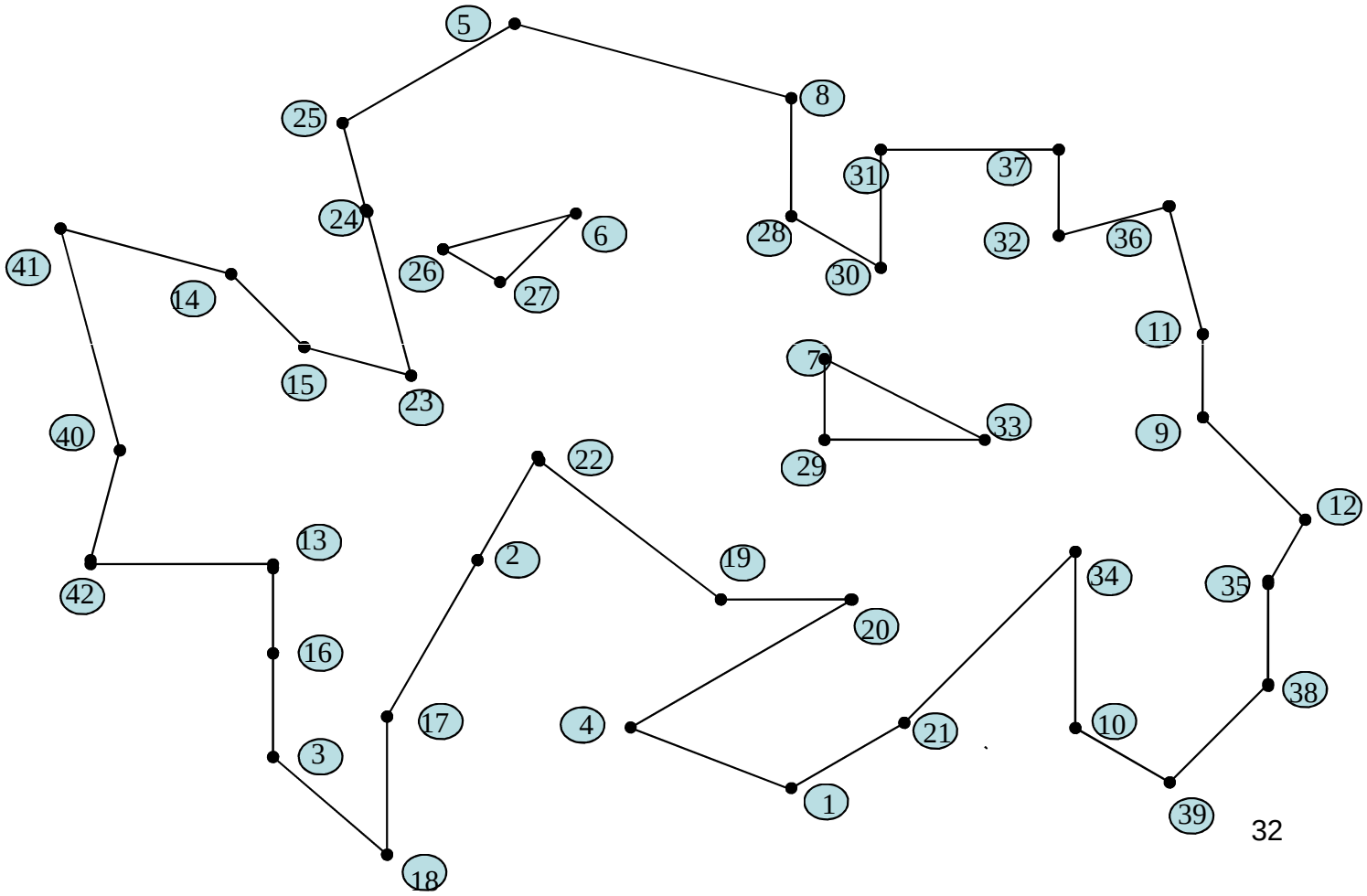


Remove new subtours

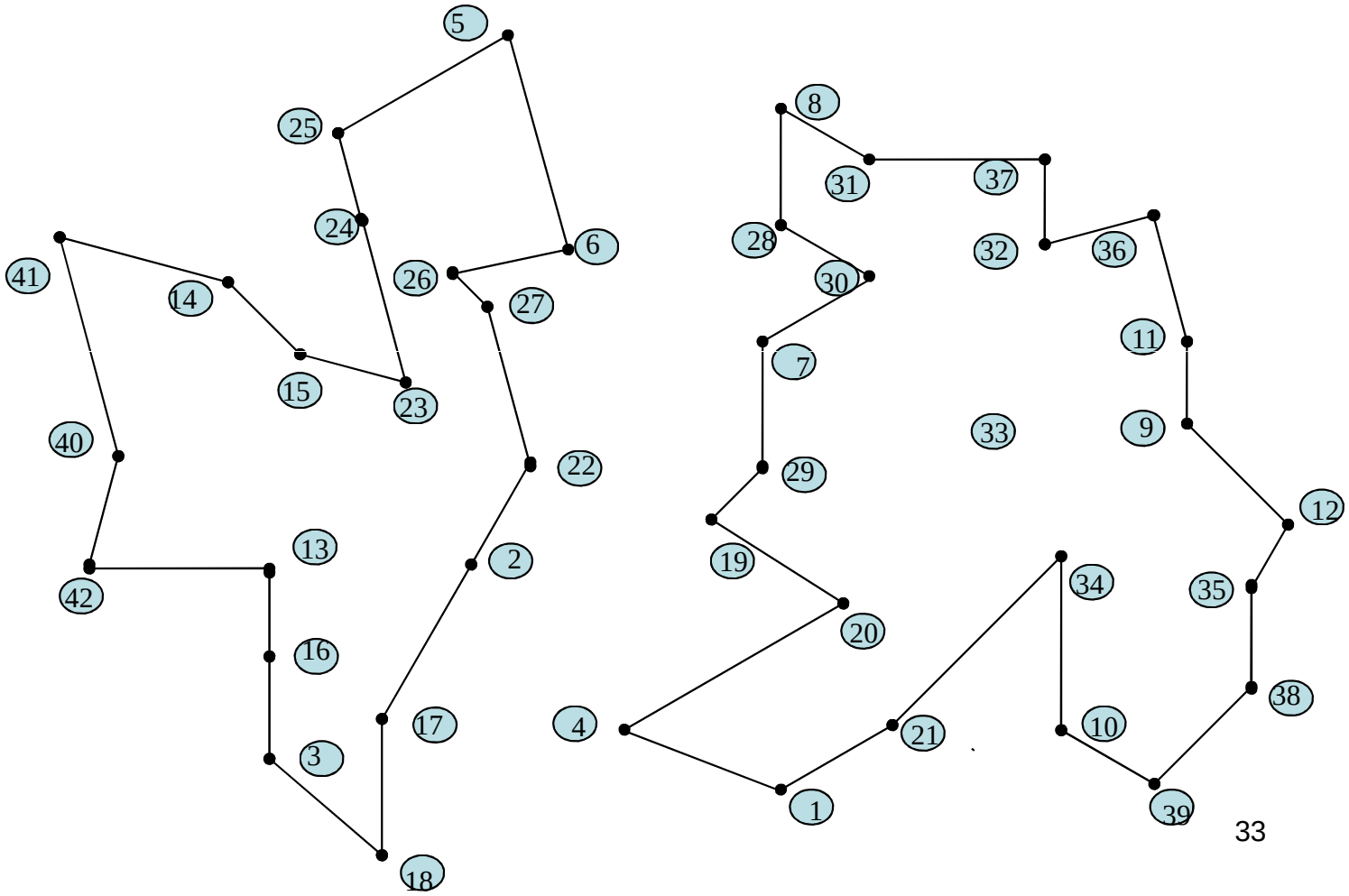
Results in further subtours Length 1179



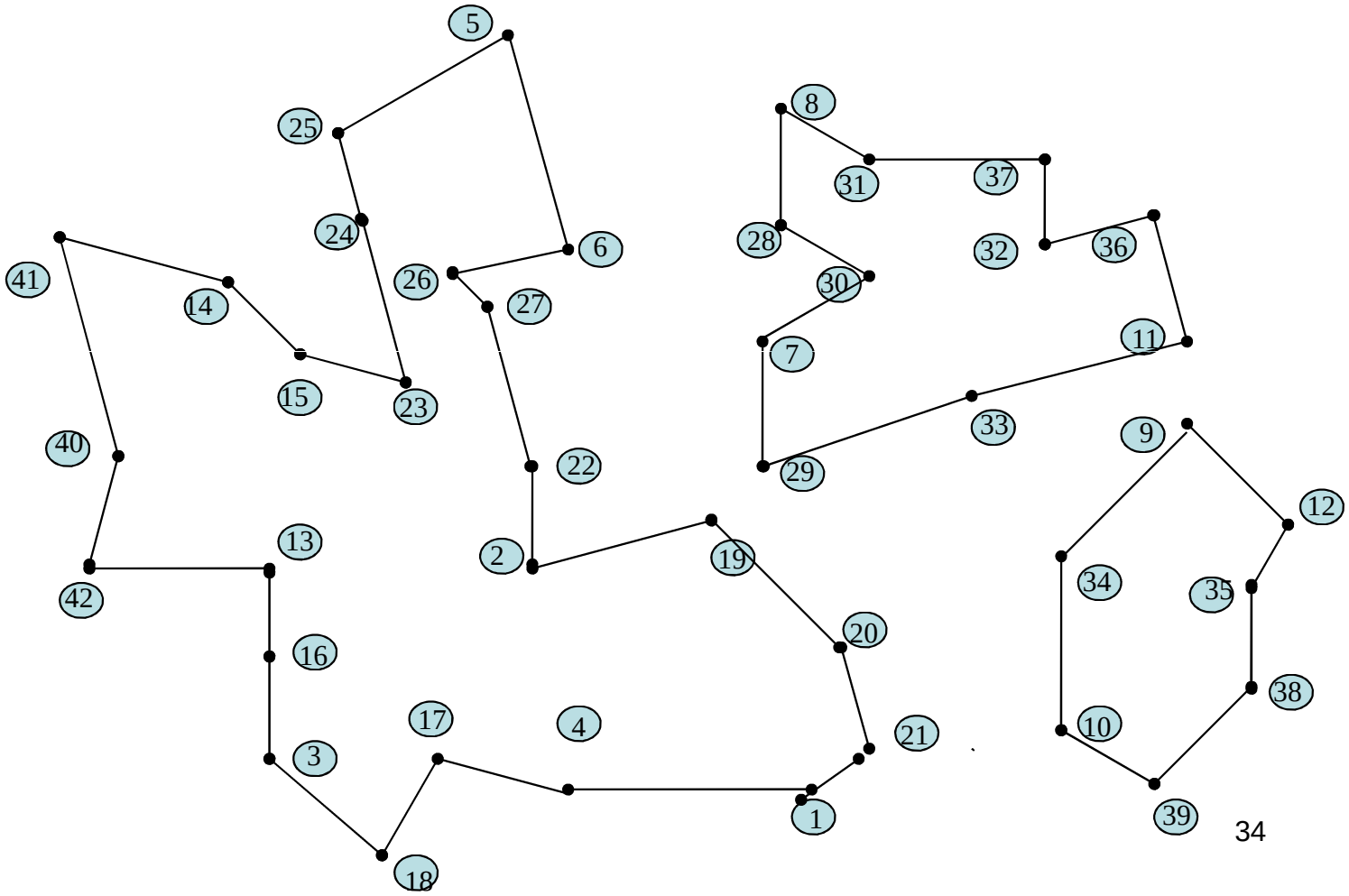
Further subtours Length 1189



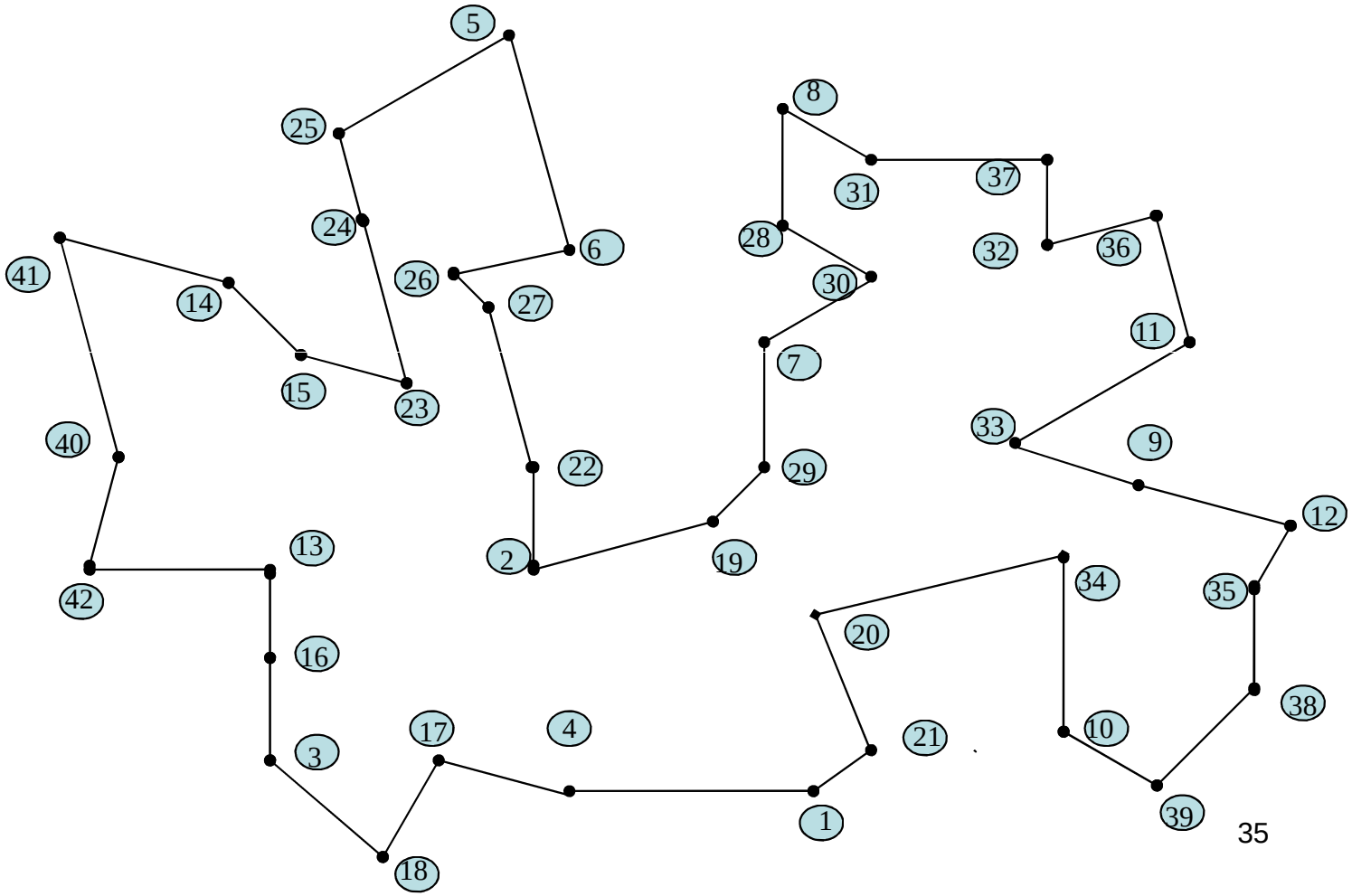
Further subtours Length 1192



Further subtours Length 1193



Optimal Solution Length 1194



END