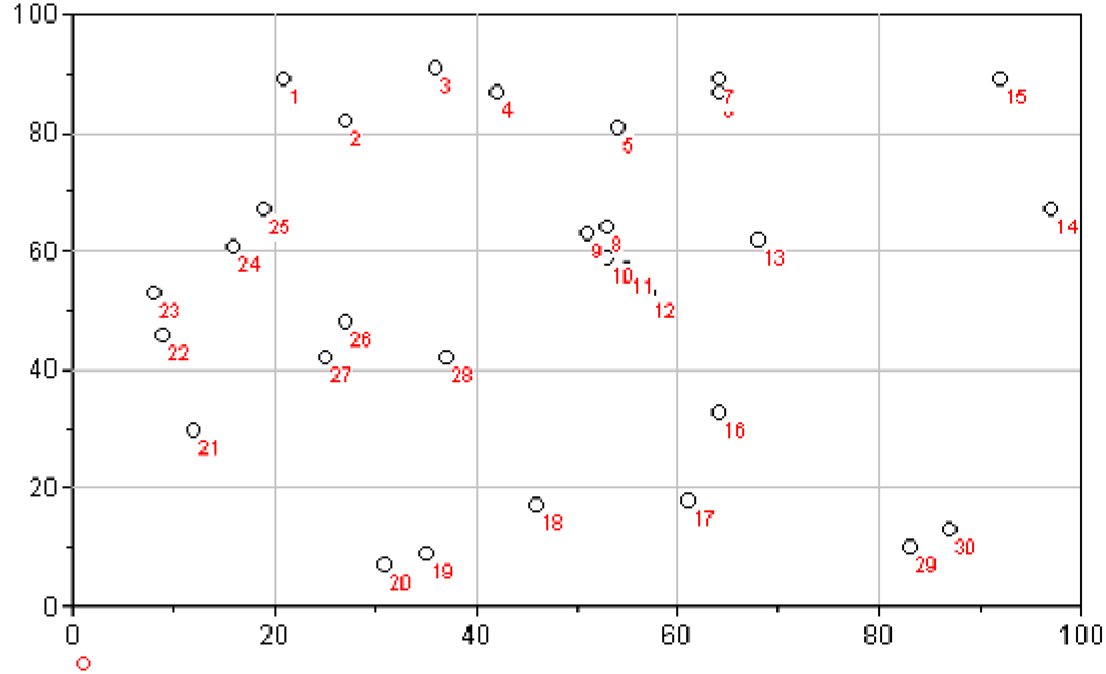
Symmetric Traveling Salesman Problem



Random distribution of 30 nodes (seed= 94054)

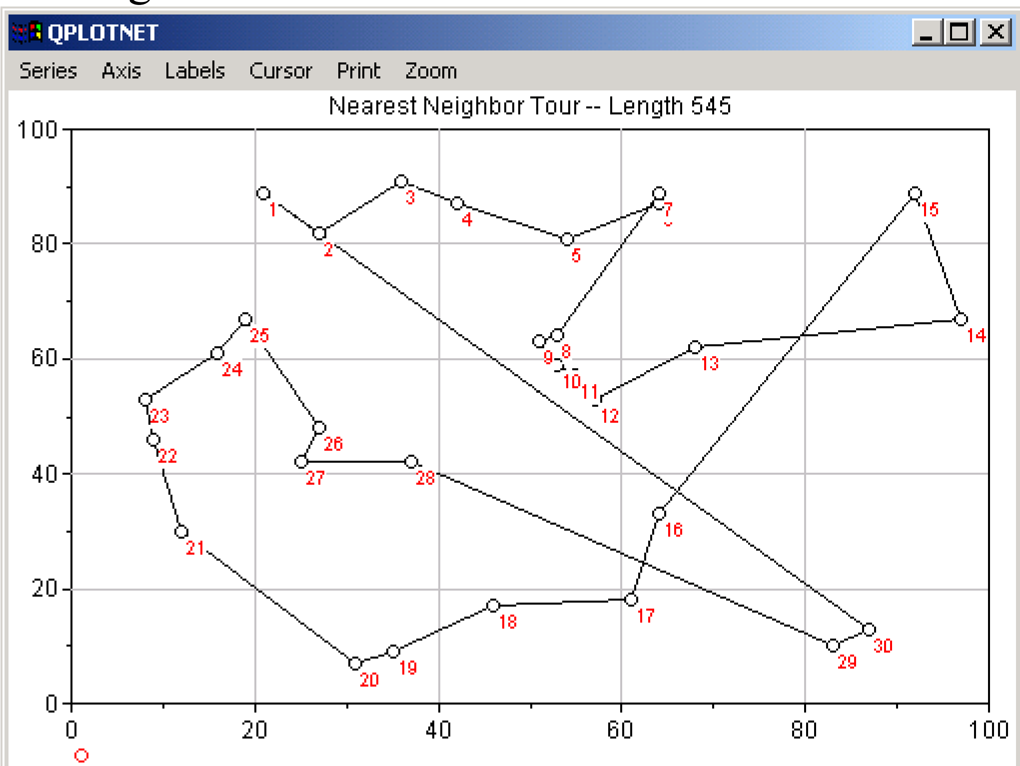
To illustrate several of the algorithms for solving symmetric TSPs, thirty nodes were uniformly distributed in a 100×100 square, and the Euclidean distances (rounded to integers) were computed.

Distance Matrix

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	0	9	15	21	34	43	43	41	40	44	47	51	54	79	71	71	81	76	81	83	60	45	38	28	22	41	47	50	100	101
2	9	0	13	16	27	37	38	32	31	35	38	42	46	72	65	61	72	68	73	75	54	40	35	24	17	34	40	41	91	91
3	15	13	0	7	21	28	28	32	32	36	39	43	43	66	56	64	77	75	82	84	66	52	47	36	29	44	50	49	94	93
4	21	16	7	0	13	22	22	25	26	30	33	37	36	59	50	58	72	70	78	81	64	53	48	37	30	42	48	45	87	87
5	34	27	21	13	0	12	13	17	18	22	24	28	24	45	39	49	63	64	74	77	66	57	54	43	38	43	49	43	77	76
6	43	37	28	22	12	0	2	25	27	30	31	35	25	39	28	54	69	72	83	87	77	69	66	55	49	54	60	52	79	77
7	43	38	28	22	13	2	0	27	29	32	33	37	27	40	28	56	71	74	85	88	79	70	67	56	50	55	61	54	81	79
8	41	32	32	25	17	25	27	0	2	5	7	12	15	44	46	33	47	48	58	61	53	48	46	37	34	31	36	27	62	61
9	40	31	32	26	18	27	29	2	0	4	7	12	17	46	49	33	46	46	56	59	51	45	44	35	32	28	33	25	62	62
10	44	35	36	30	22	30	32	5	4	0	3	7	15	45	49	28	42	43	53	56	50	46	45	37	35	28	33	23	57	57
11	47	38	39	33	24	31	33	7	7	3	0	4	14	43	49	26	39	41	52	55	51	47	47	39	37	29	34	23	55	54
12	51	42	43	37	28	35	37	12	12	7	4	0	14	42	50	21	35	38	49	53	51	49	49	42	40	30	34	23	50	50
13	54	46	43	36	24	25	27	15	17	15	14	14	0	29	36	29	45	50	62	66	64	61	61	52	49	43	47	37	54	53
14	79	72	66	59	45	39	40	44	46	45	43	42	29	0	23	47	61	71	85	89	93	90	90	81	78	73	76	65	59	55
15	71	65	56	50	39	28	28	46	49	49	49	50	36	23	0	63	77	85	98	102	99	93	91	81	76	77	82	72	80	76
16	71	61	64	58	49	54	56	33	33	28	26	21	29	47	63	0	15	24	38	42	52	57	59	56	56	40	40	28	30	30
17	81	72	77	72	63	69	71	47	46	42	39	35	45	61	77	15	0	15	28	32	50	59	64	62	65	45	43	34	23	26
18	76	68	75	70	64	72	74	48	46	43	41	38	50	71	85	24	15	0	14	18	36	47	52	53	57	36	33	27	38	41
19	81	73	82	78	74	83	85	58	56	53	52	49	62	85	98	38	28	14	0	4	31	45	52	55	60	40	34	33	48	52
20	83	75	84	81	77	87	88	61	59	56	55	53	66	89	102	42	32	18	4	0	30	45	51	56	61	41	36	36	52	56
21	60	54	66	64	66	77	79	53	51	50	51	51	64	93	99	52	50	36	31	30	0	16	23	31	38	23	18	28	74	77
22	45	40	52	53	57	69	70	48	45	46	47	49	61	90	93	57	59	47	45	45	16	0	7	17	23	18	16	28	82	85
23	38	35	47	48	54	66	67	46	44	45	47	49	61	90	91	59	64	52	52	51	23	7	0	11	18	20	20	31	86	89
24	28	24	36	37	43	55	56	37	35	37	39	42	52	81	81	56	62	53	55	56	31	17	11	0	7	17	21	28	84	86
25	22	17	29	30	38	49	50	34	32	35	37	40	49	78	76	56	65	57	60	61	38	23	18	7	0	21	26	31	86	87
26	41	34	44	42	43	54	55	31	28	28	29	30	43	73	77	40	45	36	40	41	23	18	20	17	21	0	6	12	68	69
27	47	40	50	48	49	60	61	36	33	33	34	34	47	76	82	40	43	33	34	36	18	16	20	21	26	6	0	12	66	68
28	50	41	49	45	43	52	54	27	25	23	23	23	37	65	72	28	34	27	33	36	28	28	31	28	31	12	12	0	56	58
29	100	91	94	87	77	79	81	62	62	57	55	50	54	59	80	30	23	38	48	52	74	82	86	84	86	68	66	56	0	5
30	101	91	93	87	76	77	79	61	62	57	54	50	53	55	76	30	26	41	52	56	77	85	89	86	87	69	68	58	5	0

Nearest Neighbor tours

Starting node: #1

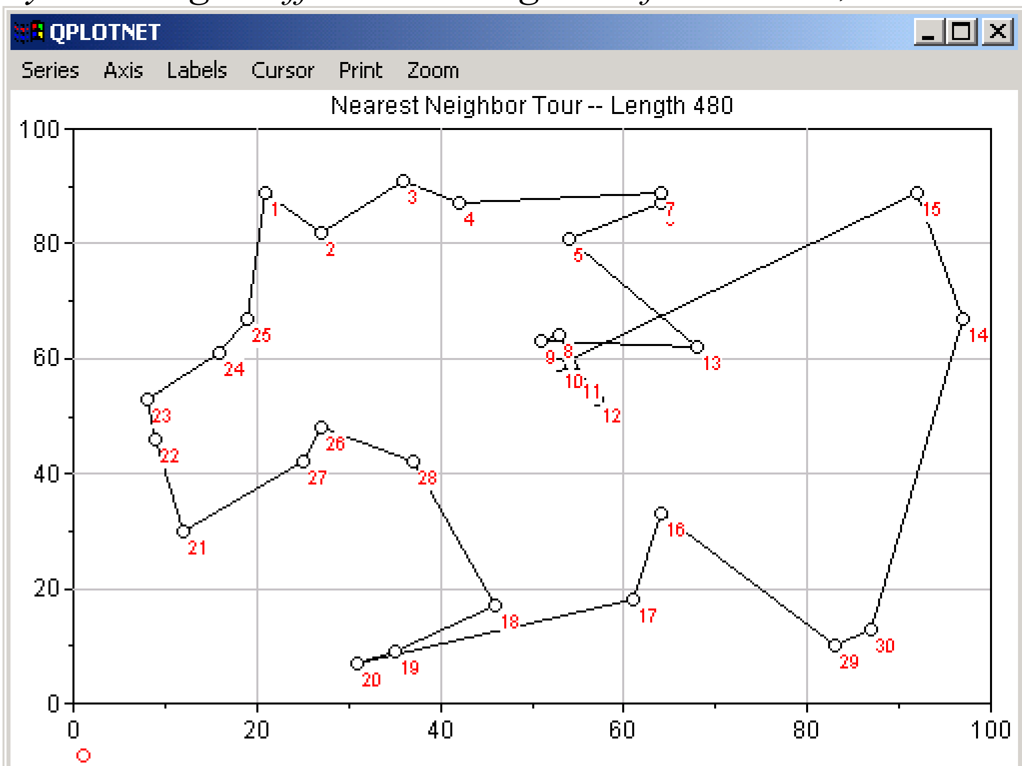



```
 1 ->  2 ->  3 ->  4 ->  5 ->  6 ->  7 ->  8 ->  9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15
16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23 -> 24 -> 25 -> 26 -> 27 -> 28 -> 29 -> 30
    ->1
```

The tour exhibits "crossing", which cannot be optimal when the distances are Euclidean!

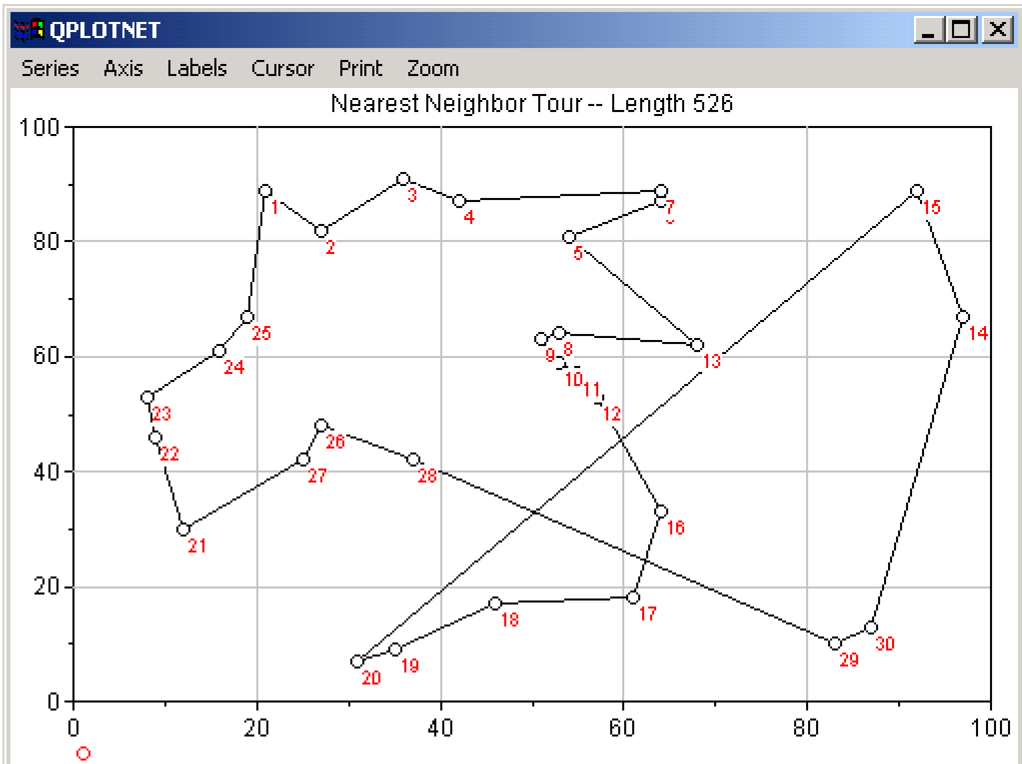
Nearest Neighbor Tour, Starting Node #10:

By choosing a different starting node for the tour, a better tour results!



10 -> 11 -> 12 -> 8 -> 9 -> 13 -> 5 -> 6 -> 7 -> 4 -> 3 -> 2 -> 1 -> 25 -> 24 ->
23 -> 22 -> 21 -> 27 -> 26 -> 28 -> 18 -> 19 -> 20 -> 17 -> 16 -> 29 -> 30 -> 14 -> 15
-> 10

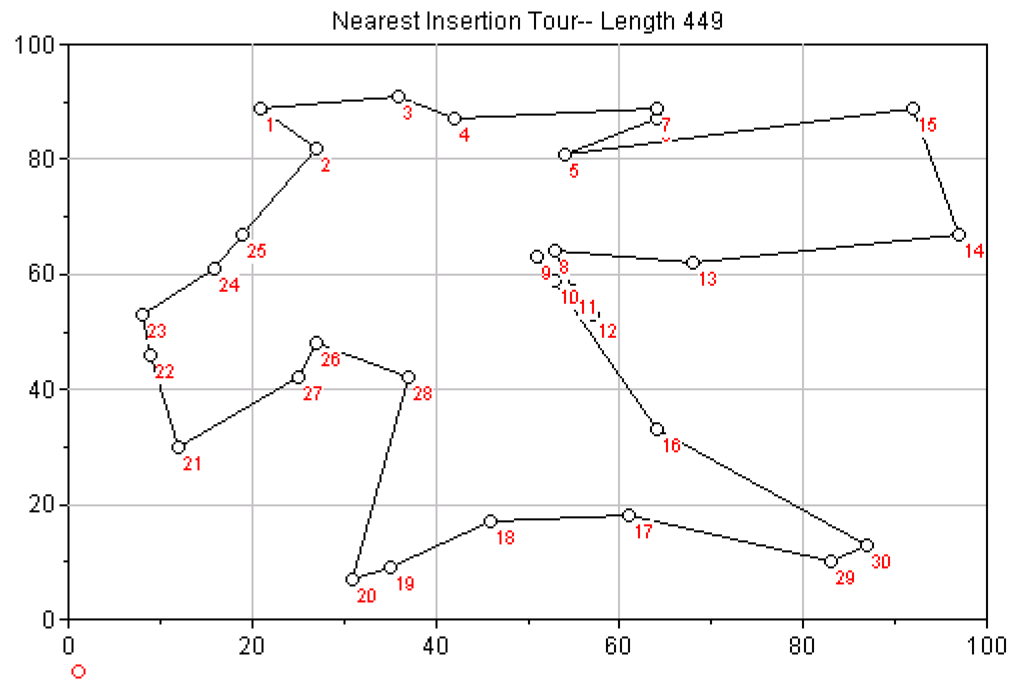
Nearest Neighbor Tour, Starting Node #20:




```
20 -> 19 -> 18 -> 17 -> 16 -> 12 -> 11 -> 10 ->  9 ->  8 -> 13 ->  5 ->  6 ->  7 ->
 4 ->  3 ->  2 ->  1 -> 25 -> 24 -> 23 -> 22 -> 21 -> 27 -> 26 -> 28 -> 29 -> 30 ->
14 -> 15 -> 20
```

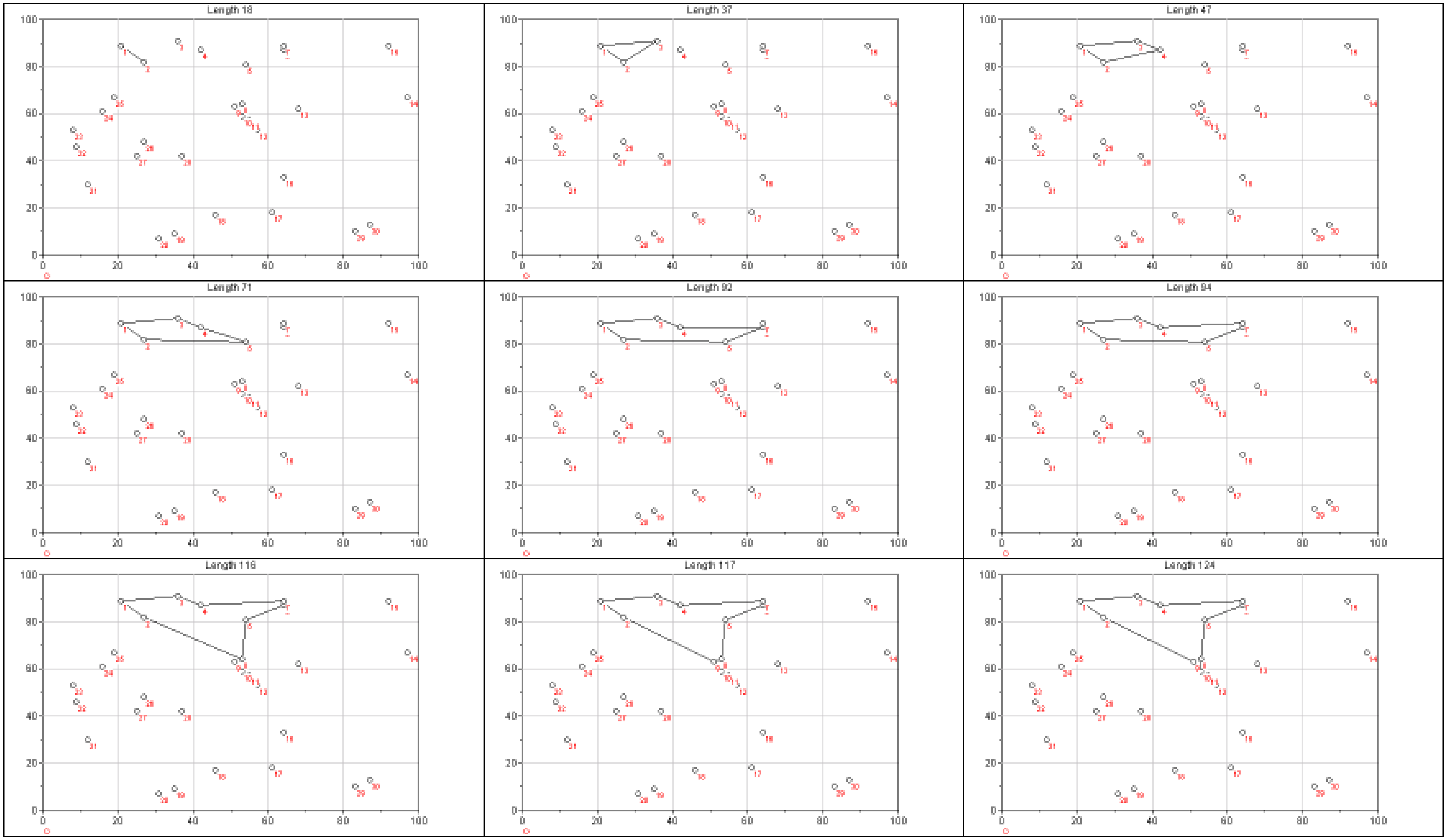
Nearest Insertion Tour

Starting Node: #1
Final tour:




```
 1 ->  3 ->  4 ->  7 ->  6 ->  5 -> 15 -> 14 -> 13 ->  8 -> 11 -> 12 -> 10 ->  9 -> 16 ->
30 -> 29 -> 17 -> 18 -> 19 -> 20 -> 28 -> 26 -> 27 -> 21 -> 22 -> 23 -> 24 -> 25 ->  2 ->
 1
```

The Nearest Insertion Algorithm is a heuristic method which grows a tour by inserting the node *nearest* to the set of nodes already on the tour.



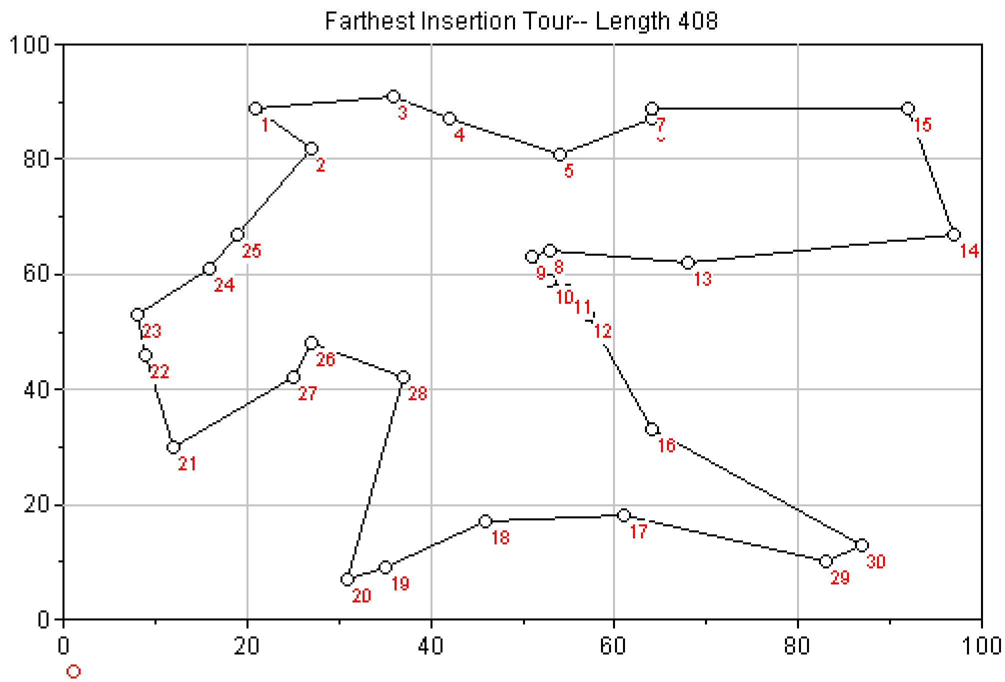
The first 9 iterations of the Nearest Insertion heuristic method

The tour found by this method is dependent upon the starting node which is used!

Farthest Insertion Tour

Starting node: #1

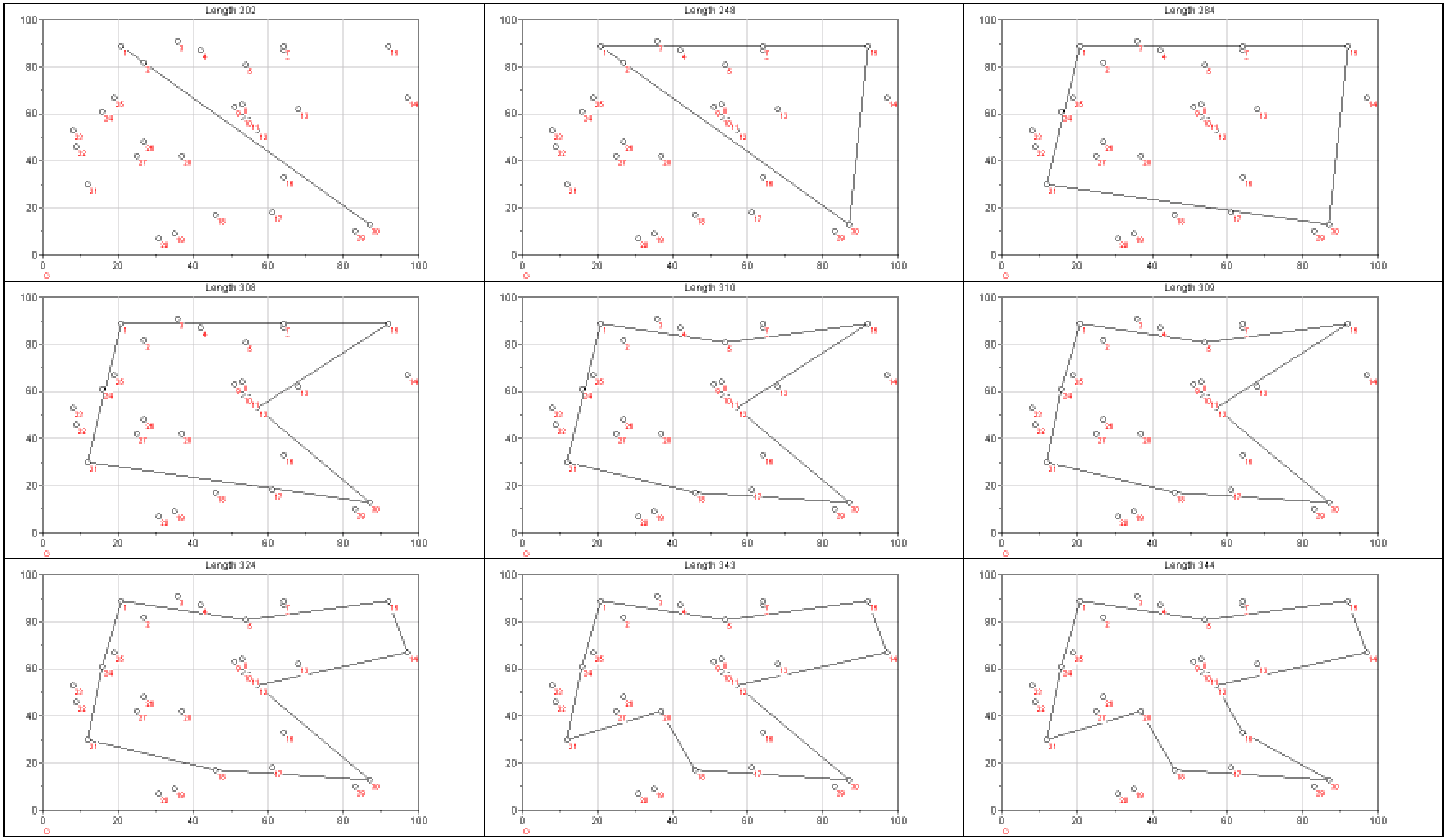
Final tour:




```
 1 ->  3 ->  4 ->  5 ->  6 ->  7 -> 15 -> 14 -> 13 ->  8 ->  9 -> 10 -> 11 -> 12 -> 16 ->
30 -> 29 -> 17 -> 18 -> 19 -> 20 -> 28 -> 26 -> 27 -> 21 -> 22 -> 23 -> 24 -> 25 -> 2 ->
 1
```

Like the Nearest Insertion Algorithm, this is a heuristic method which grows a tour by inserting the nodes one at a time, but always selecting next the node which is *farthest* from the nodes already on the tour!

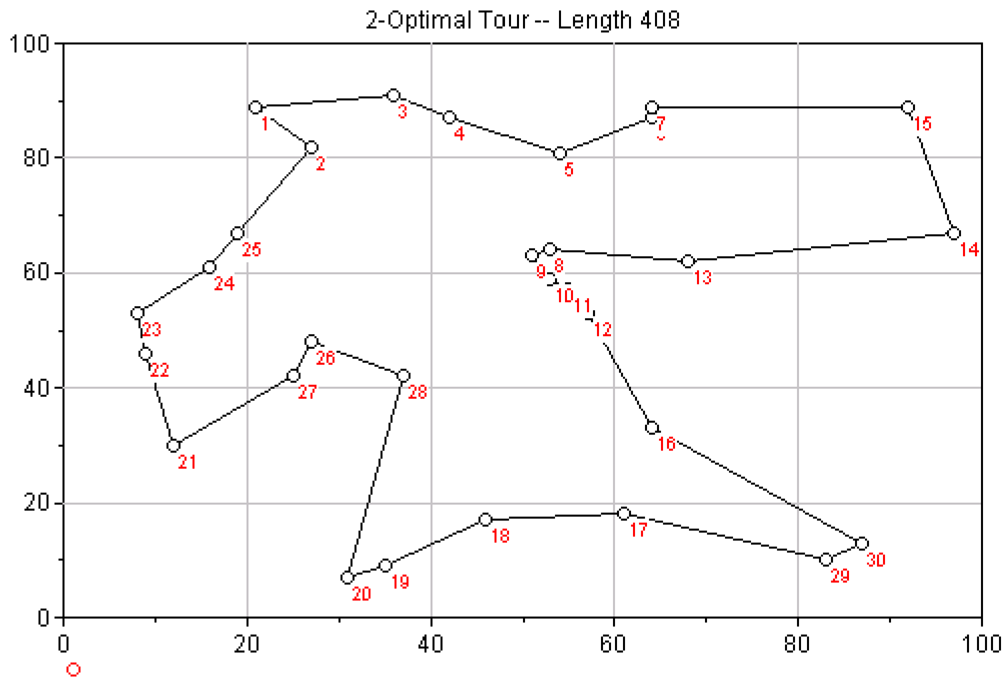
Intermediate tours:



The first several iterations of the Farthest Insertion heuristic method

2-Exchange Heuristic Method

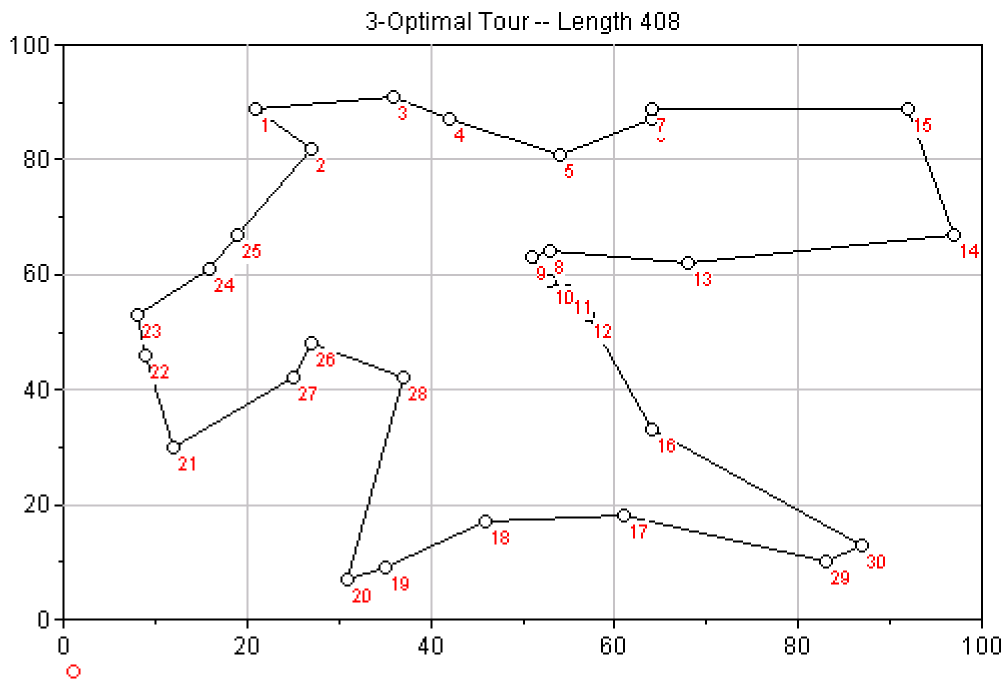
starting at tour found by Farthest Insertion method (above):



No improvement was achieved!

3-Exchange Heuristic method

starting at tour found by Farthest Insertion method (above):



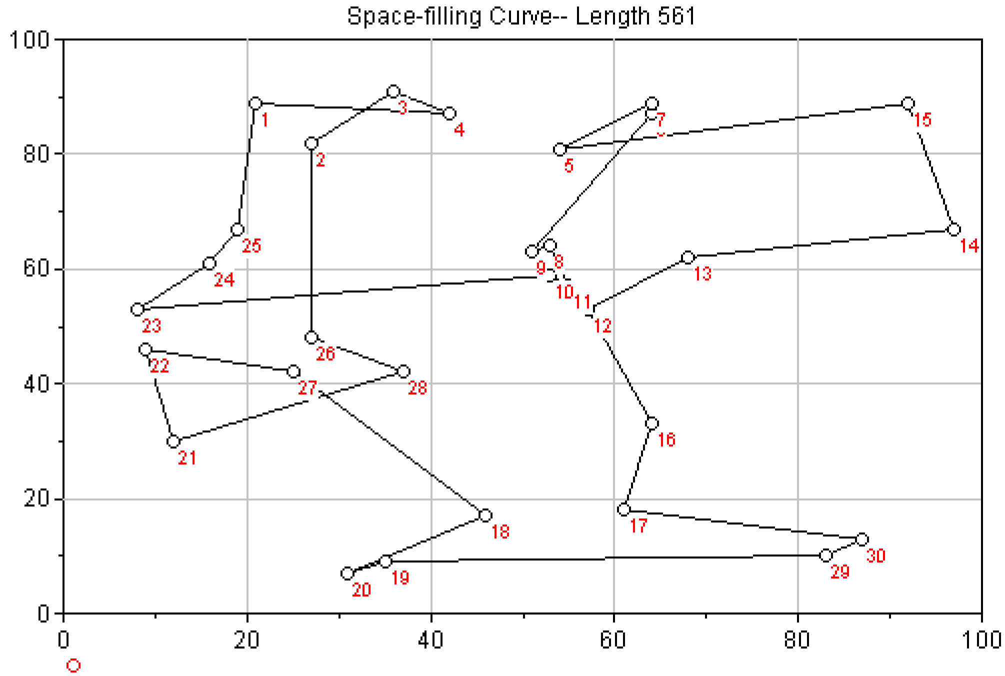
No improvements were achieved!

Tour found by Spacefilling Curve:

i	X	Y	position	r
1	21	89	0.23716696	9
2	27	82	0.16085424	6
3	36	91	0.16522571	7
4	42	87	0.20967317	8
5	54	81	0.45476204	19
6	64	87	0.41402190	17
7	64	89	0.41418419	18
8	53	64	0.39434681	15
9	51	63	0.39444565	16
10	53	59	0.37939707	13
11	55	57	0.38080958	14
12	57	53	0.62034717	23
13	68	62	0.61618536	22
14	97	67	0.58258217	21
15	92	89	0.50671323	20

i	X	Y	position	r
16	64	33	0.64953677	24
17	61	18	0.69624295	25
18	46	17	0.95860363	30
19	35	9	0.91551490	28
20	31	7	0.92383827	29
21	12	30	0.07524730	3
22	9	46	0.06013129	2
23	8	53	0.30804283	12
24	16	61	0.30352429	11
25	19	67	0.28350610	10
26	27	48	0.10898578	5
27	25	42	0.05206395	1
28	37	42	0.10282683	4
29	83	10	0.74165160	27
30	87	13	0.72653817	26

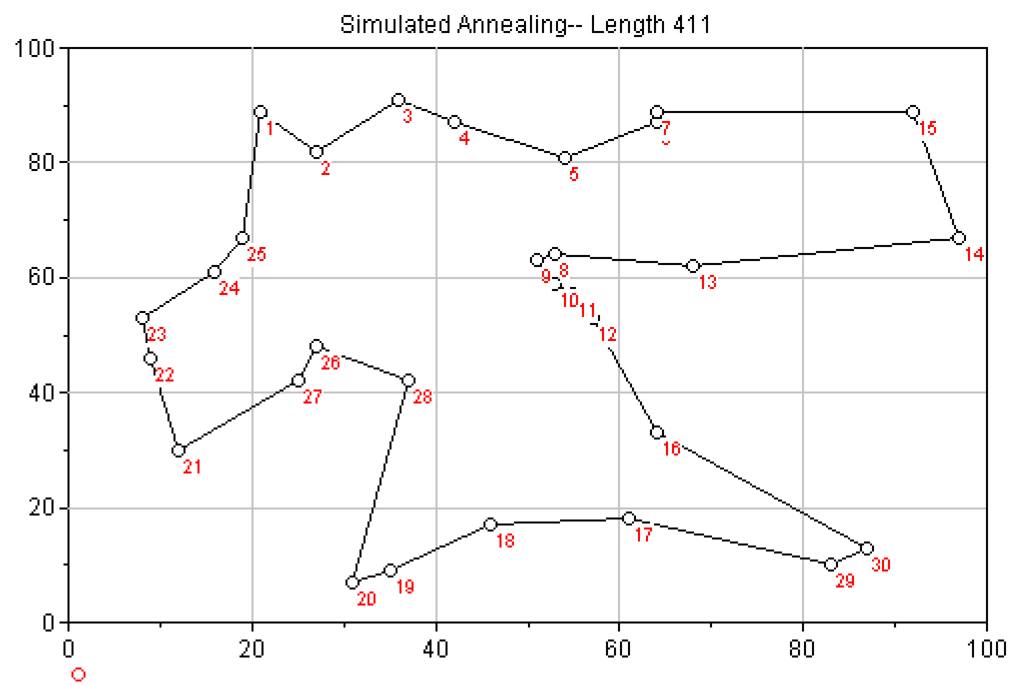
r=rank on space-filling curve



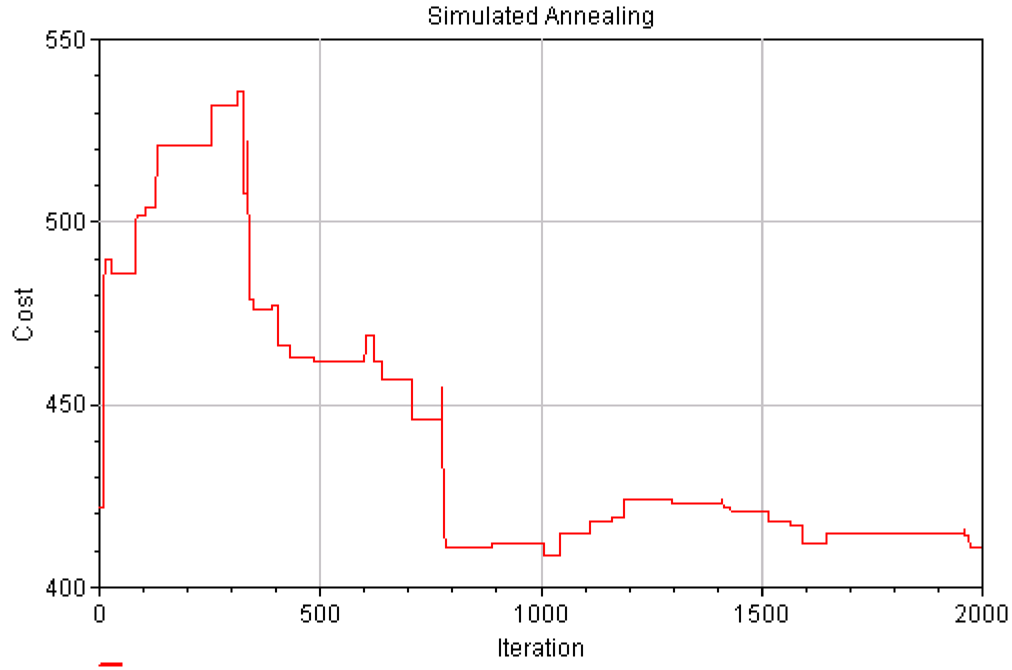

```
 1 -> 25 -> 24 -> 23 -> 10 -> 11 -> 8 ->   9 ->  6 ->  7 ->  5 -> 15 -> 14 -> 13 ->
12 -> 16 -> 17 -> 30 -> 29 -> 19 -> 20 -> 18 -> 27 -> 22 -> 21 -> 28 -> 26 ->  2 ->
 3 ->  4 -> 1
```

Simulated Annealing

Starting with tour found by Farthest Insertion Method:

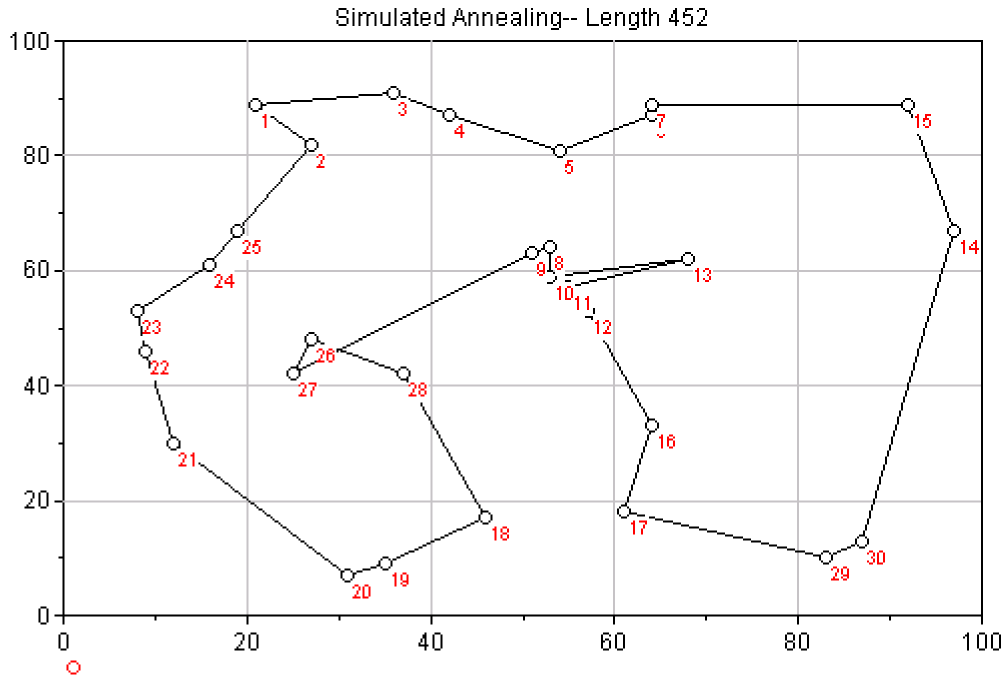


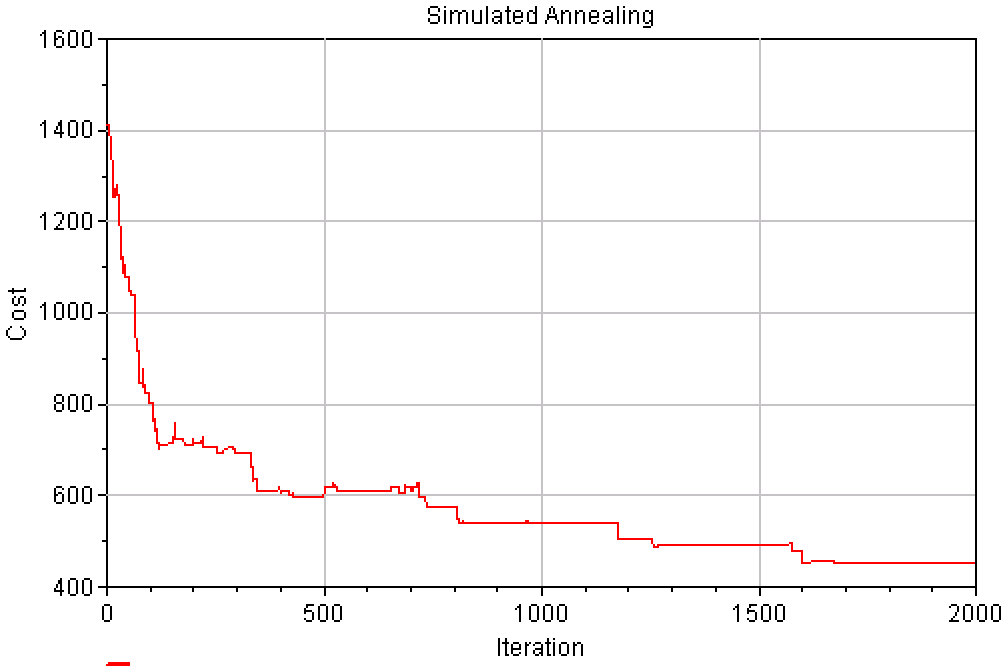
This is an exchange algorithm, but one in which it is possible that exchanges will be performed which lengthen the tour! In this case, the "cooling schedule" was set so that 2000 iterations were performed, an exchange increasing the length by 20 is performed with probability 25%



Note that initial exchanges often lengthened the tour, but later exchanges tend to shorten the tour.

Simulated Annealing, starting with a random tour:




Simulated Annealing
Cost
1600
1400
1200
1000
800
600
400
0
500
1000
1500
2000
Iteration

Vertex Penalty Method (using target value 408)

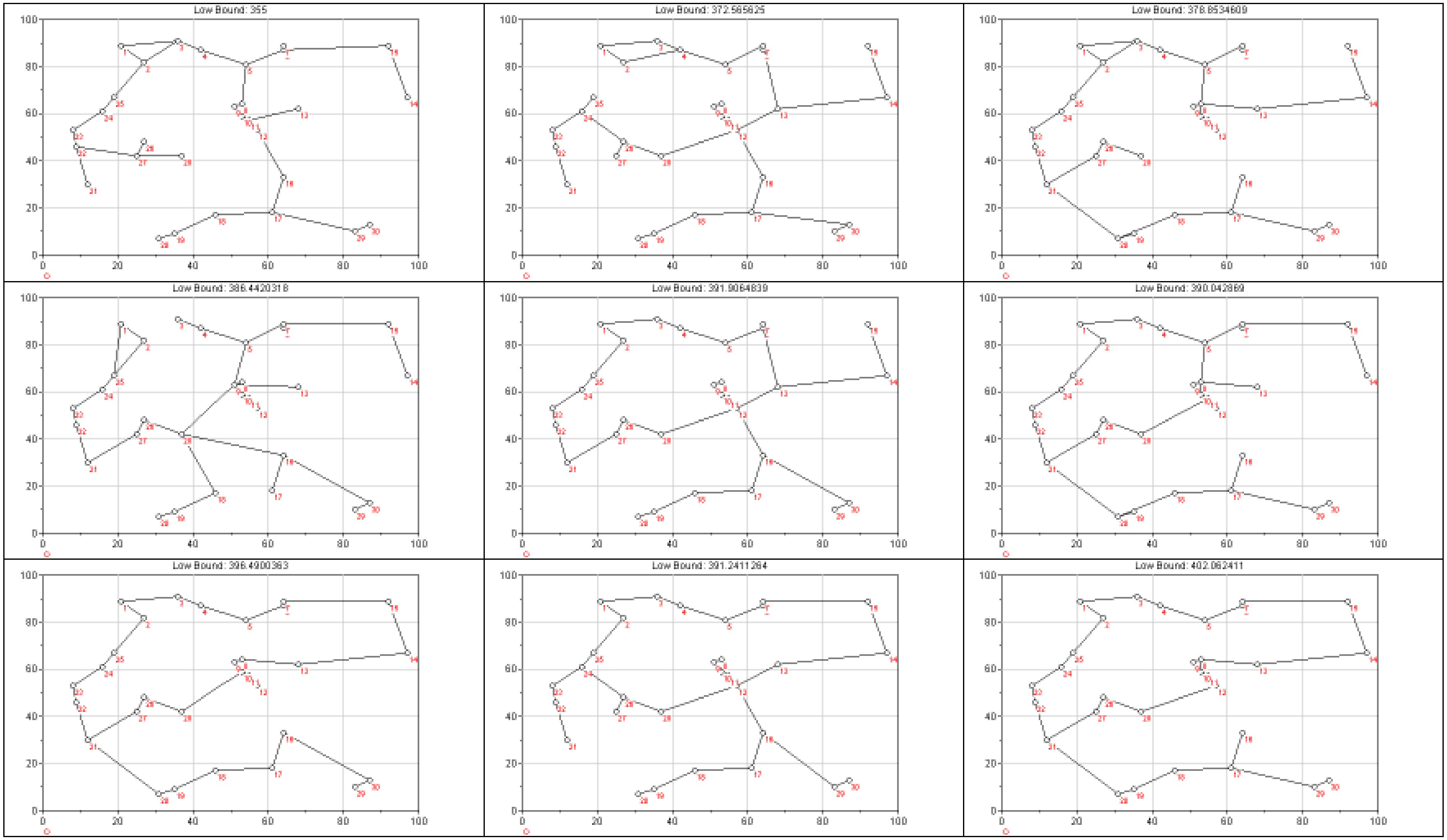
- This method is essentially a Lagrangian Relaxation of the TSP, relaxing the degree constraints but imposing the "spanning one-tree" constraints.

- The vertex "penalties" are, in fact, Lagrangian multipliers corresponding to the degree constraints—they are increased for nodes with degree greater than 2, and reduced for nodes with degree equal to 1.

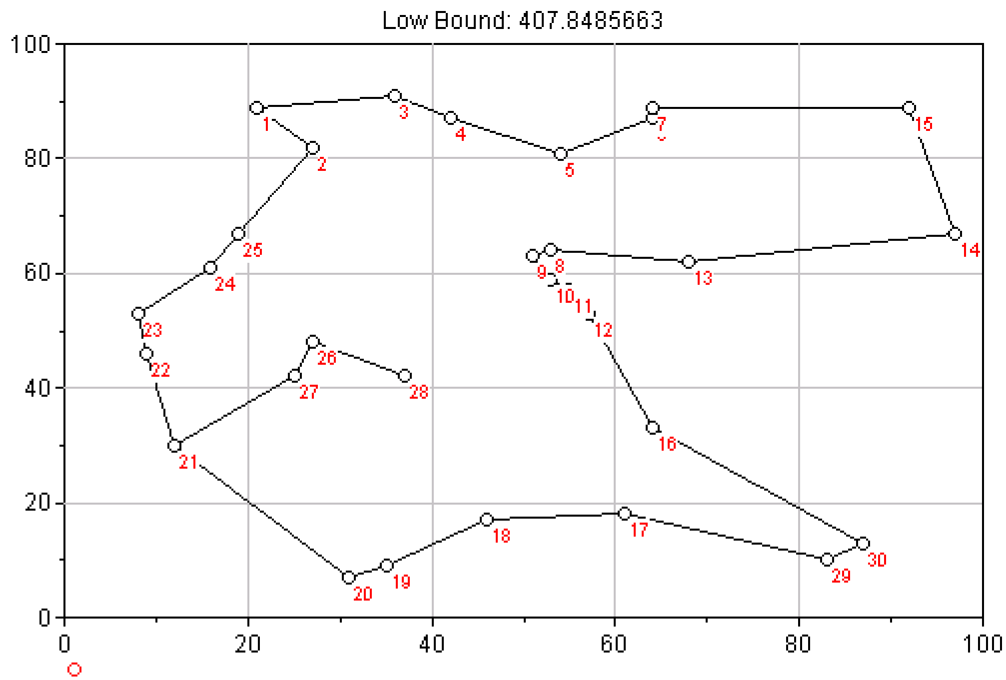
- If the spanning one-tree converges to a tour, it is guaranteed to be optimal!

- A "target" value is used in computing the "stepsize", i.e., the size of the adjustment to the vertex penalties.

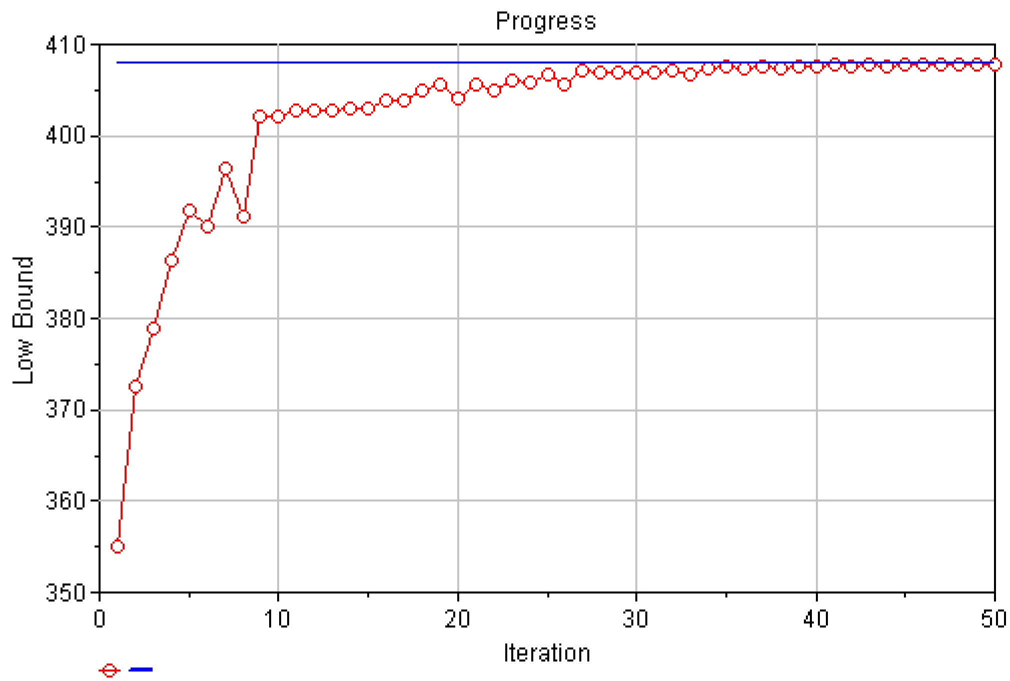
Initial spanning one-trees:



Final spanning one-tree:



Even though the one-tree did not converge to a tour, knowing that the optimal length is integer-valued, we are guaranteed that the previous tour (with length 408) is optimal!


Progress
Low Bound
410
400
390
380
370
360
350
0
10
20
30
40
50
Iteration

Final penalties:

Node	Penalty
1	0.00000000000
2	5.91793078496
3	5.04732052955
4	1.86030468750
5	7.09644934167
6	¯0.59232136912
7	¯1.10789340952
8	1.13849312980
9	1.75178582941
10	3.62475462772
11	3.56549133480
12	3.45921490975
13	¯2.12774721548
14	¯5.39623456302
15	¯4.96870212619
16	¯1.75453496915
17	6.93841252526
18	0.00000000000
19	¯5.71955757821
20	¯9.71465160304
21	2.11237881953
22	4.17009095324
23	0.95575118766
24	3.91945513170
25	¯0.04642899848
26	1.41137698111
27	2.16095419055
28	¯3.75973096707
29	¯9.92955920821
30	¯10.01280295674

Note that, since the relaxed degree constraints are equations, the Lagrangian multipliers (vertex penalties) are unrestricted in sign!