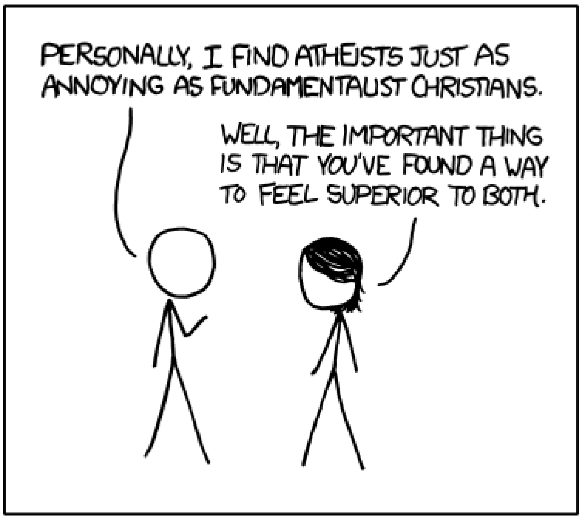


I swear upon the deity of self-referential humour, that all output shown below was produced by running the code listed at the end of this document, without undue assistance from others.

- Benjamin Yi

Question 1

$$f(s) = 5 * \left| \frac{\sum_{i=1}^{n} w(s_i) y_i}{\sum_{i=1}^{n} w_i} \right| + \left| \frac{\sum_{i=1}^{n} w(s_i) x_i}{\sum_{i=1}^{n} w_i} \right|$$

Question 2

$$\begin{aligned} t(s,a,b) &= (s_1, s_2, \ldots, s_{a-1}, s_b, s_{a+1}, \ldots, s_{b-1}, s_a, s_{b+1}, \ldots, s_{120}) && \text{if } b > a \\ &= (s_1, s_2, \ldots, s_{b-1}, s_a, s_{b+1}, \ldots, s_{a-1}, s_b, s_{a+1}, \ldots, s_{120}) && \text{if } a > b \\ &= s && \text{if } a = b \end{aligned}$$

Question 3

We move the container in position a to the empty location b, leaving position a empty.

Question 4

$$\begin{array}{ll} b = a + 60 \longrightarrow t(s, a, a+60) & \text{Swapping containers vertically does not change anything} \\ a = b + 60 \longrightarrow t(s, b+60, b) & \text{Swapping containers vertically does not change anything} \\ w(s_a) = w(s_b) & \text{Swapping a container with an identical container does nothing} \end{array}$$

Note that as we assume uniquely weighted containers, the third scenario reduces down to $s_a = s_b$ i.e. swapping a container with itself, or swapping empty containers.

Question 5

$$N(s) = \Big\{ t(s,a,b) \text{ for } a = 1, 2, \ldots, 119;\ b = a+1, a+2, \ldots, 120;\ b \neq a + 60;\ w(s_a) \neq w(s_b) \Big\}$$

Question 6

We store extra intermediate variables which correspond to the centre of masses:

$$\mathrm{zy}(s) = \sum_{i=1}^{n} w(s_i) y_i$$
$$\mathrm{zx}(s) = \sum_{i=1}^{n} w(s_i) x_i$$

To update these on each evaluation:

$$\mathrm{zy}(s_{\mathrm{new}}) = \mathrm{zy}(s_{\mathrm{old}}) - w(s_{\mathrm{old},a})y_a + w(s_{\mathrm{old},b})y_a - w(s_{\mathrm{old},b})y_b + w(s_{\mathrm{old},a})y_b$$
$$\mathrm{zx}(s_{\mathrm{new}}) = \mathrm{zx}(s_{\mathrm{old}}) - w(s_{\mathrm{old},a})x_a + w(s_{\mathrm{old},b})x_a - w(s_{\mathrm{old},b})x_b + w(s_{\mathrm{old},a})x_b$$

where a, b are equal to the position index swapped for this evaluation. The new objective function is then given by:

$$f(s) = 5 * \left|\frac{\mathrm{zy}(s)}{w_T}\right| + \left|\frac{\mathrm{zx}(s)}{w_T}\right|$$

where w_T is the sum of the weights of all containers. Note that in implementation, division by w_T can be left until the end as it is constant and positive.

Question 7

Create an array of size n, one element for each container. Initialise the array with some arbitrary value $<$ -h. Whenever a container is swapped, update the corresponding array element to be equal to the iteration count. A container would then be considered banned if (current iteration count - array element value) $<$ h. This requires an array element lookup per container considered each iteration, up to a maximum of 119 * 120 lookups each iteration.

Question 8

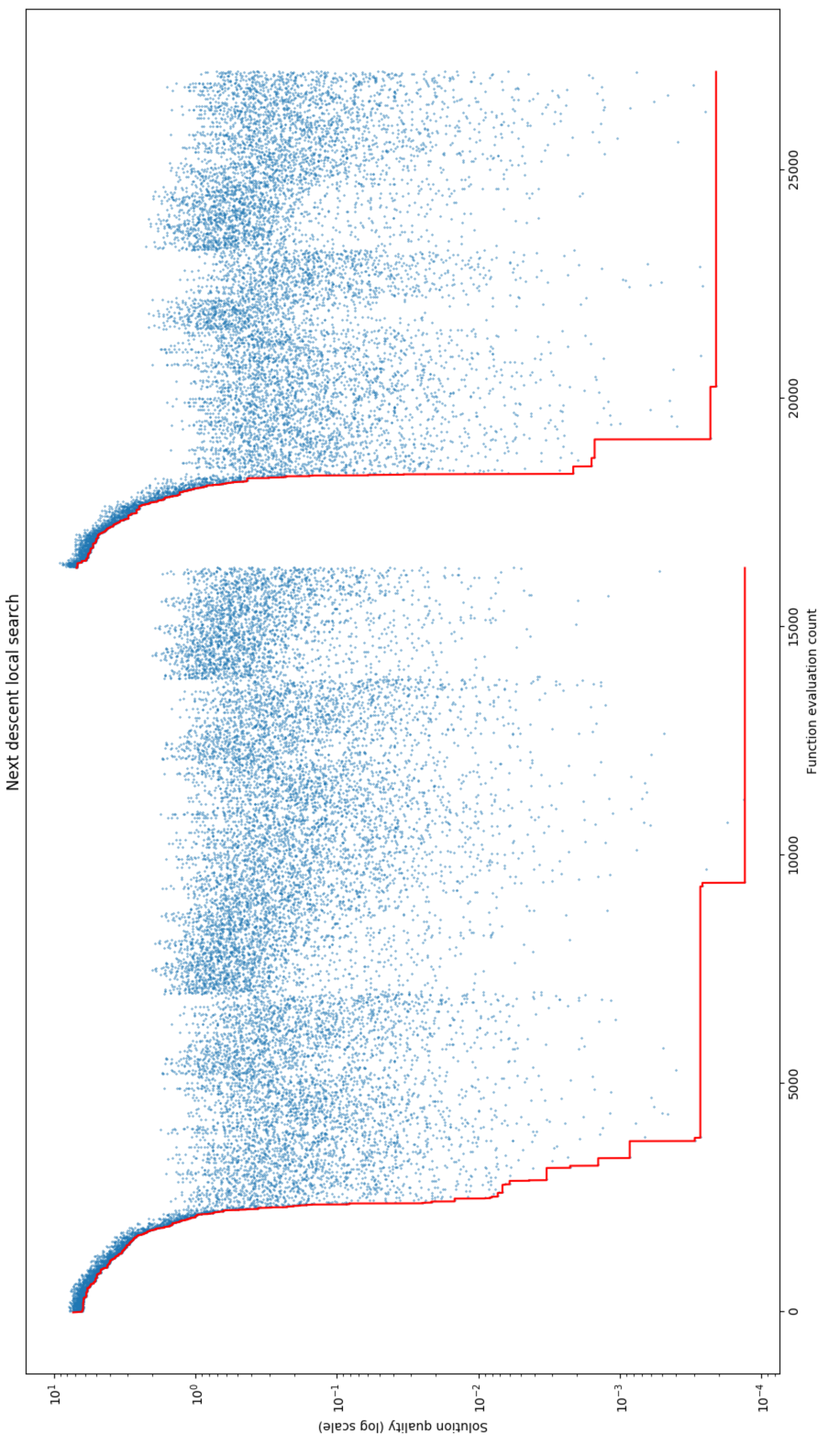
Instead of recording movement of containers, we could record container positions swapped. This would mean recording a, b instead of s[a], s[b].

Two next-descents code

```
# Benjamin Yi
# byi649
# 925302651

import random
import matplotlib.pyplot as plt
import math
import numpy as np

# Input data
with open("ProbA.txt", 'r') as f:
    w = [line.strip() for line in f.readlines()]

n = int(w[0])
w = [0] + [float(x) for x in w[1:]]

with open("Positions.txt", 'r') as f:
    pos = [line.strip() for line in f.readlines()[1:]]

x = [float(tup.split()[1]) for tup in pos]
y = [float(tup.split()[2]) for tup in pos]

# Do two next-descent on problem A
zArray = []
bestzArray = []

for k in range(2):

    # Generate random starting solutions
    s = list(range(1, n+1)) + [0]*(120-n)
    for i in range(119):
        j = random.randint(i+1, 119)
        s[i], s[j] = s[j], s[i]

    # Iterate next-descent
    zx = sum([w[s[i]]*x[i] for i in range(120)])
    zy = sum([w[s[i]]*y[i] for i in range(120)])
    z = 5 * abs(zy) + abs(zx)

    zArray.append(z)
    bestzArray.append(z)

    converged = False
    last_swap = (0, 0)

    while(not converged):
        for a in range(119):
            for b in range(a + 1, 120):
                # Stop if the neighbourhood surrounding the last swap
                    ↪ has been searched
                if (a, b) == last_swap:
                    converged = True
                    break
                else:
                    if b != a + 60 and w[s[a]] != w[s[b]]:
                        zy_new = zy - w[s[a]]*y[a] + w[s[b]]*y[a] - w[s[
                            ↪ b]]*y[b] + w[s[a]]*y[b]
                        zx_new = zx - w[s[a]]*x[a] + w[s[b]]*x[a] - w[s[
                            ↪ b]]*x[b] + w[s[a]]*x[b]
                        z_new = 5 * abs(zy_new) + abs(zx_new)

                        zArray.append(z_new) # For plot

                        if z_new < z:
```

```
                        # Update and swap
                        z, zy, zx = z_new, zy_new, zx_new
                        s[a], s[b] = s[b], s[a]
                        last_swap = (a, b)

                    bestzArray.append(z)

                # In the extremely rare case the shuffle gave us a local
                    ↪  minima
                if last_swap == (0, 0) and (a, b) == (119, 120):
                    converged = True

            if converged:
                break

    bestzArray.append(np.nan) # To create a vertical break between
        ↪ descents
    totalweight = sum(w)
    print("dY =", zy/totalweight)
    print("dX =", zx/totalweight)
    print("z =", z/totalweight)

plt.scatter(x=range(len(zArray)), y=[x/totalweight for x in zArray],
    ↪ marker='x', alpha=0.5, s=1)
plt.plot([x/totalweight for x in bestzArray], 'r')
plt.xlabel("Function evaluation count")
plt.ylabel("Solution quality (log scale)")
plt.yscale('log')
plt.title("Next descent local search")
plt.show()

with open("Results.txt", 'w') as f:
    f.write(str(z/totalweight)+"\n")
    f.write("\n".join([str(x) for x in s]))
```

Two next-descents plot



200 iterations code

```
# Benjamin Yi
# byi649
# 925302651

import random
import matplotlib.pyplot as plt
import math
import numpy as np

# Input data
with open("ProbA.txt", 'r') as f:
    w = [line.strip() for line in f.readlines()]

n = int(w[0])
w = [0] + [float(x) for x in w[1:]]

with open("Positions.txt", 'r') as f:
    pos = [line.strip() for line in f.readlines()[1:]]

x = [float(tup.split()[1]) for tup in pos]
y = [float(tup.split()[2]) for tup in pos]

bestZ = float("inf")

for k in range(200):

    # Generate random starting solutions
    s = list(range(1, n+1)) + [0]*(120-n)
    for i in range(119):
        j = random.randint(i+1, 119)
        s[i], s[j] = s[j], s[i]

    # Iterate next-descent
    zx = sum([w[s[i]]*x[i] for i in range(120)])
    zy = sum([w[s[i]]*y[i] for i in range(120)])
    z = 5 * abs(zy) + abs(zx)

    converged = False
    last_swap = (0, 0)

    while(not converged):
        for a in range(119):
            for b in range(a + 1, 120):
                # Stop if the neighbourhood surrounding the last swap
                    ↪ has been searched
                if (a, b) == last_swap:
                    converged = True
                    break
                else:
                    if b != a + 60 and w[s[a]] != w[s[b]]:
                        zy_new = zy - w[s[a]]*y[a] + w[s[b]]*y[a] - w[s[
                            ↪ b]]*y[b] + w[s[a]]*y[b]
                        zx_new = zx - w[s[a]]*x[a] + w[s[b]]*x[a] - w[s[
                            ↪ b]]*x[b] + w[s[a]]*x[b]
                        z_new = 5 * abs(zy_new) + abs(zx_new)

                        if z_new < z:
                            # Update and swap
                            z, zy, zx = z_new, zy_new, zx_new
                            s[a], s[b] = s[b], s[a]
                            last_swap = (a, b)

                # In the extremely rare case the shuffle gave us a local
                    ↪  minima
```

```
                if last_swap == (0, 0) and (a, b) == (119, 120):
                    converged = True

            if converged:
                break

    totalweight = sum(w)
    z = z/totalweight

    if z < bestZ:
        bestZ, bestS = z, s

print(bestZ)

with open("Results.txt", 'w') as f:
    f.write(str(bestZ)+"\n")
    f.write("\n".join([str(x) for x in bestS]))
```

Problem A

Index of Container in each Loading Position (0=empty)

0	0	63	93	64	79	17	92	94	97
0	0	91	18	37	46	42	88	27	69
0	0	3	62	14	33	19	67	73	44
0	83	34	26	43	31	82	2	7	56
49	0	47	65	28	10	61	57	70	55
0	0	21	13	75	24	25	12	50	53

0	84	39	74	76	68	66	77	51	38
0	0	32	80	100	87	54	20	85	78
0	0	86	11	45	22	89	99	36	15
0	0	0	95	48	5	98	23	8	71
0	0	72	35	4	29	90	9	6	59
52	41	1	40	16	60	96	30	58	81

Total Weight in each Loading Position

0	32.9731088	937.895025	1659.95645	1495.08227	1294.18491	1053.74018	1113.65921	484.216683	1352.18398
0	0	503.495428	1806.89916	978.199674	1450.8922	998.39202	732.124516	1539.09736	944.887796
0	0	422.88651	1640.66185	1005.48725	1110.75594	1505.94383	1191.3012	600.749822	750.933473
0	40.2280783	101.349268	1164.04337	1339.60903	1565.20022	819.071844	1285.14473	1457.06086	876.704225
37.50877195	0	794.505922	1691.84939	972.6507	1290.17909	1492.59113	866.539274	878.933774	1665.31054
59.16333354	394.774074	1082.82972	1255.267	866.679052	718.860321	750.656355	796.392365	1824.777	1148.91427
96.67210549	467.975261	3842.96188	9218.67722	6657.70796	7430.07269	6620.39536	5985.16129	6784.83551	6738.93428

ID Number 925302651

-------- Solution --------

Ship Loading Positions	Index of Container at Posn
1	0
2	0
3	0
4	0
5	49
6	0
7	0
8	0
9	0
10	83
11	0
12	0
13	63
14	91
15	3
16	34
17	47
18	21
19	93
20	18
21	62
22	26

---- Solution Quality ----

ProbA

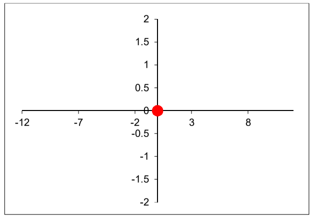
dX=	6.15E-06
dY=	-1.2E-06
Obj Fn=	1.19E-05

- Number of Containers -

Containers	100

- Number of Positions -

Positions	120



--- Container Data ----

Container Index	Container Weight
0	0
1	280.7546
2	669.389
3	207.7648
4	47.5508
5	830.8559
6	275.2896
7	942.0694
8	514.9914
9	346.4315
10	387.0416
11	888.9294
12	589.9275
13	967.7608
14	585.6589
15	405.8013
16	369.025
17	389.7931
18	956.3363
19	745.112
20	391.4605
21	802.0752
22	548.2732

--------------- Workings ----------------

Objective Calculation Workings		
Weights	Data X	Data Y
0	-33	7.5
0	-33	4.5
0	-33	1.5
0	-33	-1.5
37.50877	-33	-4.5
0	-33	-7.5
0	-27	7.5
0	-27	4.5
0	-27	1.5
40.22808	-27	-1.5
0	-27	-4.5
0	-27	-7.5
127.9442	-21	7.5
148.794	-21	4.5
207.7648	-21	1.5
101.3493	-21	-1.5
263.758	-21	-4.5
802.0752	-21	-7.5
988.1317	-15	7.5
956.3363	-15	4.5
751.7324	-15	1.5
864.8866	-15	-1.5

-------- Workings ---------

Duplicate Check	All Containers
OK	OK
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE

Problem B

Index of Container in each Loading Position (0=empty)

22	40	34	75	81	80	85	63	3	87
38	92	44	118	19	43	90	17	89	50
32	42	108	51	74	70	67	49	21	9
14	16	71	11	102	109	5	83	37	73
2	26	27	114	120	115	105	23	66	116
4	18	52	93	65	47	111	12	98	77

64	31	28	91	79	41	25	55	29	94
78	10	95	13	39	76	33	101	53	82
20	56	113	110	57	84	46	15	62	69
6	8	119	86	106	100	7	72	60	117
59	99	103	112	88	68	24	45	54	107
30	58	96	104	61	1	36	48	35	97

Total Weight in each Loading Position

15.80798313	19.6620722	14.4341165	690.672541	347.07809	179.955102	356.968359	95.4666947	161.577258	389.72757
11.51868956	14.5786363	56.9292305	318.397034	169.481822	101.510343	66.3951876	191.585677	212.587807	98.6557077
8.269346576	12.8145075	81.1770566	373.786954	162.901066	184.083909	64.2069438	35.5263137	64.5710179	83.9715463
3.601720718	7.90853459	20.2458726	150.329165	122.340661	177.691271	162.386714	179.818818	190.67455	336.217052
1.222637606	18.6193626	99.3624943	368.088813	184.190724	107.804395	244.381229	82.0531062	50.5236607	241.125135
2.103200872	76.7844862	98.7870648	249.869908	162.386501	153.977248	262.510647	606.627937	108.161285	400.721921
42.52357846	150.367599	370.935835	2151.14442	1148.37886	905.022267	1156.84908	1191.07855	788.095579	1550.41893

ID Number 925302651

-------- Solution --------

Ship Loading Positions	Index of Container at Posn
1	22
2	38
3	32
4	14
5	2
6	4
7	40
8	92
9	42
10	16
11	26
12	18
13	34
14	44
15	108
16	71
17	27
18	52
19	75
20	118
21	51
22	11

---- Solution Quality ----

ProbA

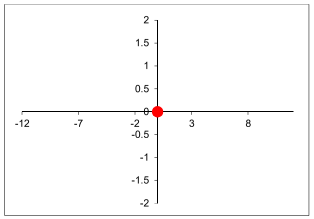
dX=	-1.4E-06
dY=	-3.2E-07
Obj Fn=	3E-06

- Number of Containers -

Containers	120

- Number of Positions -

Positions	120



--- Container Data ----

Container Index	Container Weight
0	0
1	87.51651
2	0.87015
3	61.58868
4	0.745666
5	80.59673
6	2.884038
7	81.78999
8	5.483662
9	37.46672
10	12.91584
11	67.36751
12	248.008
13	79.73182
14	0.717683
15	10.34096
16	2.424872
17	27.63356
18	5.244997
19	99.16572
20	7.49868
21	32.51991
22	0.105823

--------------- Workings ----------------

Objective Calculation Workings		
Weights	Data X	Data Y
0.105823	-33	7.5
0.508907	-33	4.5
0.770666	-33	1.5
0.717683	-33	-1.5
0.87015	-33	-4.5
0.745666	-33	-7.5
0.959409	-27	7.5
1.662801	-27	4.5
2.358016	-27	1.5
2.424872	-27	-1.5
2.625794	-27	-4.5
5.244997	-27	-7.5
3.919351	-21	7.5
5.398177	-21	4.5
7.007721	-21	1.5
7.130167	-21	-1.5
8.972342	-21	-4.5
19.43479	-21	-7.5
391.0193	-15	7.5
238.6652	-15	4.5
80.63375	-15	1.5
67.36751	-15	-1.5

-------- Workings ---------

Duplicate Check	All Containers
OK	OK
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE
TRUE	TRUE

Problem C

Index of Container in each Loading Position (0=empty)

0	0	0	0	0	0	14	0	0	0
0	0	0	0	0	0	9	0	0	0
0	0	0	0	0	15	5	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	18	13	1	0	0
0	0	0	0	16	0	0	0	0	0

0	0	0	0	8	0	0	0	0	11
0	0	0	0	19	0	0	0	0	0
2	0	0	0	0	10	0	6	0	20
0	0	0	0	0	12	7	0	0	0
0	0	0	17	3	0	0	0	0	0
0	0	0	0	4	0	0	0	0	0

Total Weight in each Loading Position

0	0	0	0	55.598	0	185.369	0	0	107.2458
0	0	0	0	397.123	0	70.973	0	0	0
4.259	0	0	0	0	305.907	23.299	32.159	0	455.768
0	0	0	0	0	130.781	43.258	0	0	0
0	0	0	298.215	8.258	345.945	156.589	1.234	0	0
0	0	0	0	272.41	0	0	0	0	0
4.259	0	0	298.215	733.389	782.633	479.488	33.393	0	563.0138

ID Number 925302651

---- Solution Quality ----

ProbA

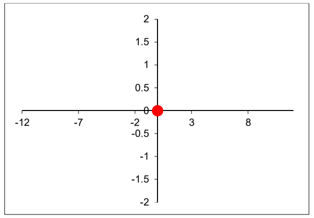
dX=	4.1E-05
dY=	-3.5E-05
Obj Fn=	0.000217

- Number of Containers -

Containers	20

- Number of Positions -

Positions	120



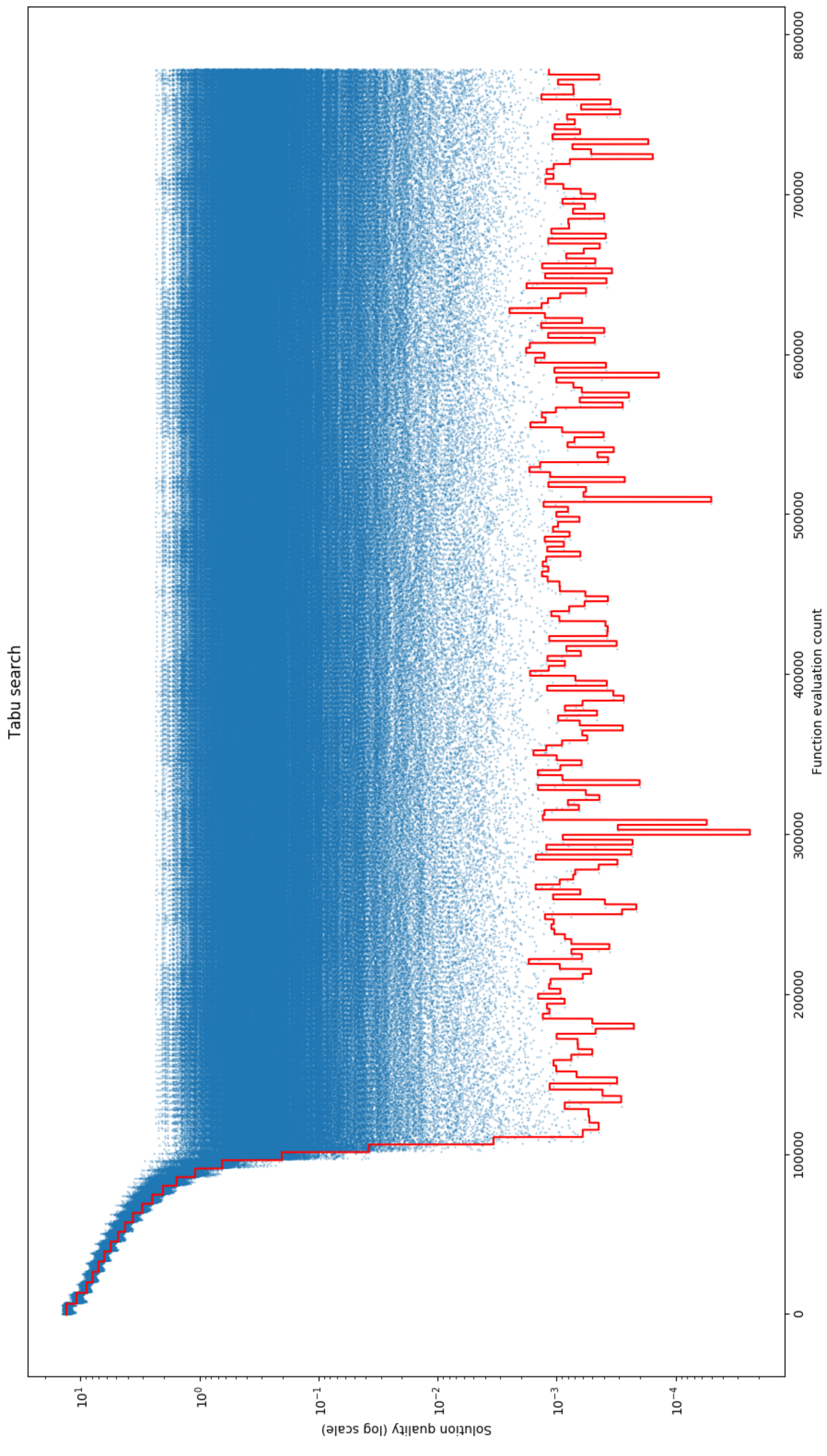
-------- Solution --------		--- Container Data ----		--------------- Workings ----------------			-------- Workings ---------	
Ship Loading Positions	Index of Container at Posn	Container Index	Container Weight	Objective Calculation Workings			Duplicate Check	All Containers
		0	0	Weights	Data X	Data Y	OK	Incomplete
1	0	1	1.234	0	-33	7.5	TRUE	TRUE
2	0	2	4.259	0	-33	4.5	TRUE	TRUE
3	0	3	8.258	0	-33	1.5	TRUE	TRUE
4	0	4	15.958	0	-33	-1.5	TRUE	TRUE
5	0	5	23.299	0	-33	-4.5	TRUE	TRUE
6	0	6	32.159	0	-33	-7.5	TRUE	TRUE
7	0	7	43.258	0	-27	7.5	TRUE	TRUE
8	0	8	55.598	0	-27	4.5	TRUE	TRUE
9	0	9	70.973	0	-27	1.5	TRUE	TRUE
10	0	10	87.258	0	-27	-1.5	TRUE	TRUE
11	0	11	107.2458	0	-27	-4.5	TRUE	TRUE
12	0	12	130.781	0	-27	-7.5	TRUE	TRUE
13	0	13	156.589	0	-21	7.5	TRUE	TRUE
14	0	14	185.369	0	-21	4.5	TRUE	TRUE
15	0	15	218.649	0	-21	1.5	TRUE	TRUE
16	0	16	256.452	0	-21	-1.5	TRUE	TRUE
17	0	17	298.215	0	-21	-4.5	TRUE	TRUE
18	0	18	345.945	0	-21	-7.5	TRUE	TRUE
19	0	19	397.123	0	-15	7.5	TRUE	TRUE
20	0	20	455.768	0	-15	4.5	TRUE	TRUE
21	0	21	32.51991	0	-15	1.5	TRUE	FALSE
22	0	22	0.105823	0	-15	-1.5	TRUE	FALSE

Tabu search code

```
# Benjamin Yi
# byi649
# 925302651

import matplotlib.pyplot as plt
import math

# Input data
with open("ProbA.txt", 'r') as f:
    w = [line.strip() for line in f.readlines()]

n = int(w[0])
w = [0] + [float(x) for x in w[1:]]

with open("Positions.txt", 'r') as f:
    pos = [line.strip() for line in f.readlines()[1:]]

x = [float(tup.split()[1]) for tup in pos]
y = [float(tup.split()[2]) for tup in pos]

# Do steepest-descent
zArray = []
bestzArray = []
h = min(20, int(float(n)/3.0))
history = [-float('inf')]*120 # Arbitrary value < -h

# Generate starting solution
s = list(range(1, n+1)) + [0]*(120-n)

# Iterate steepest-descent
zx = sum([w[s[i]]*x[i] for i in range(120)])
zy = sum([w[s[i]]*y[i] for i in range(120)])
z = 5 * abs(zy) + abs(zx)

zArray.append(z)
bestzArray.append(z)

converged = False
iter_count = 0
best_index = float("inf") # Arbitrary value > 0
worsen_count = 0

while(not converged):
    neighbourhood = []
    iter_count += 1
    for a in range(119):
        # Skip tabu swaps
        if (not (iter_count - history[s[a]] < h)):
            for b in range(a + 1, 120):
                # Skip tabu swaps
                if (not (iter_count - history[s[b]] < h)):
                    if b != a + 60 and w[s[a]] != w[s[b]]:
                        zy_new = zy - w[s[a]]*y[a] + w[s[b]]*y[a] - w[s[
                            ↪ b]]*y[b] + w[s[a]]*y[b]
                        zx_new = zx - w[s[a]]*x[a] + w[s[b]]*x[a] - w[s[
                            ↪ b]]*x[b] + w[s[a]]*x[b]
                        z_new = 5 * abs(zy_new) + abs(zx_new)

                        zArray.append(z_new) # For plot
                        neighbourhood.append((a, b, z_new, zy_new,
                            ↪ zx_new))
                        bestzArray.append(z) # For plot

    # Terminate at max iterations or stuck in local minima
```

```
    if iter_count > 1e5 or worsen_count > 2*best_index:
        converged = True

    # Best swap minimises z_new
    newSwap = min(neighbourhood, key=lambda t: t[2])

    # If our new solution is worse than the best we know
    if newSwap[2] > min(bestzArray):
        worsen_count += 1
    else:
        # We've found the best solution so far
        worsen_count = 0
        best_index = iter_count

    (a, b, z, zy, zx) = newSwap

    # Keep a record of recent swaps (not including empty containers)
    if s[a] != 0:
        history[s[a]] = iter_count
    if s[b] != 0:
        history[s[b]] = iter_count

    # Swap positions
    s[a], s[b] = s[b], s[a]


totalweight = sum(w)
z = min(bestzArray)/totalweight
print("z =", z)

plt.scatter(x=range(len(zArray)), y=[x/totalweight for x in zArray],
    ↪ alpha=0.5, s=0.2)
plt.plot([x/totalweight for x in bestzArray], 'r')
plt.xlabel("Function evaluation count")
plt.ylabel("Solution quality (log scale)")
plt.yscale('log')
plt.title("Tabu search")
plt.show()
```

Tabu search plot



Tabu search zoomed in plot