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

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

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

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

Question 6

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

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

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

To update these on each evaluation:

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

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.

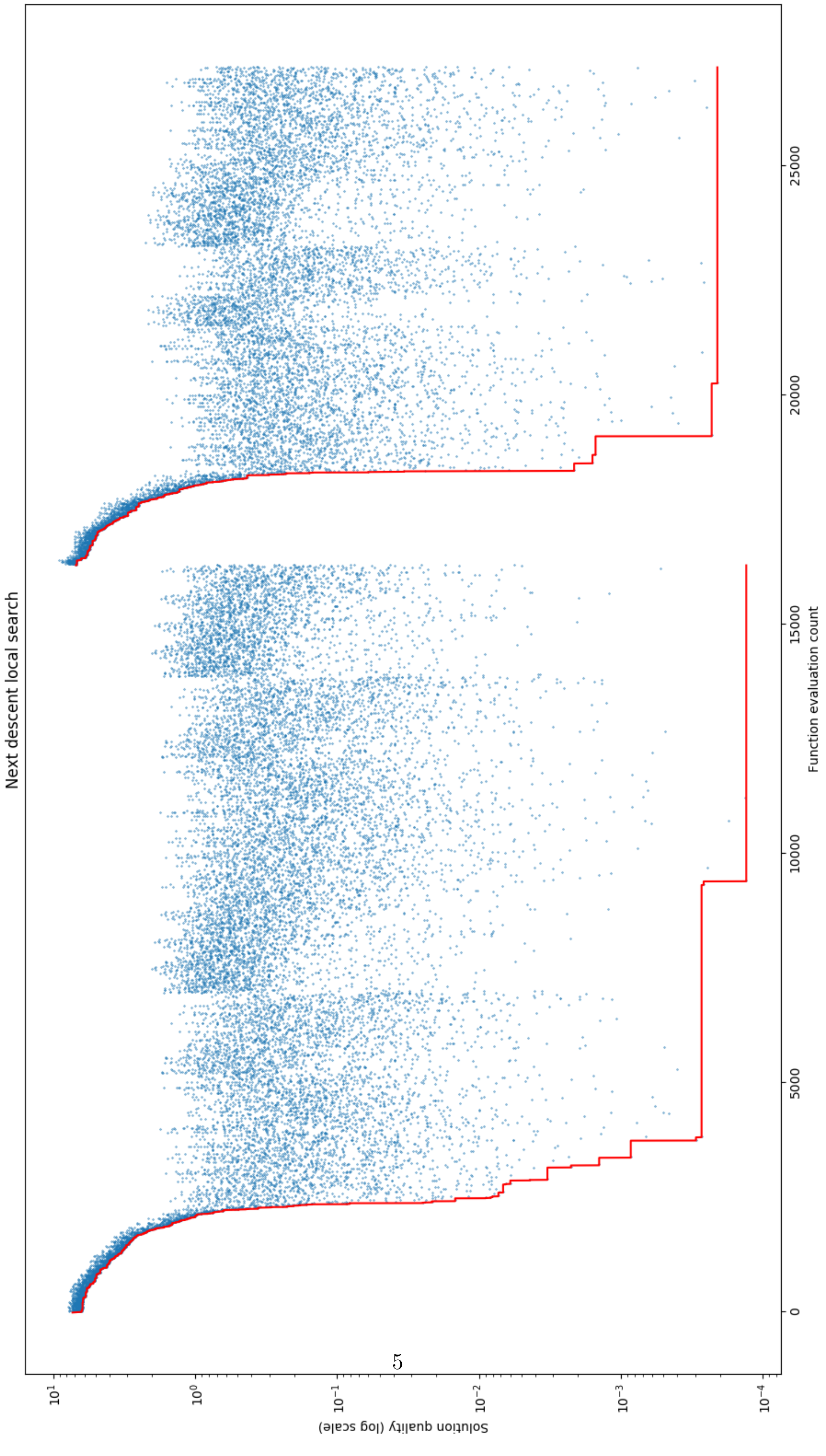
Question 8

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]))
```


Next descent local search
Function evaluation count
0
5000
10000
15000
20000
25000
Solution quality (log scale)
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}

200 iterations code

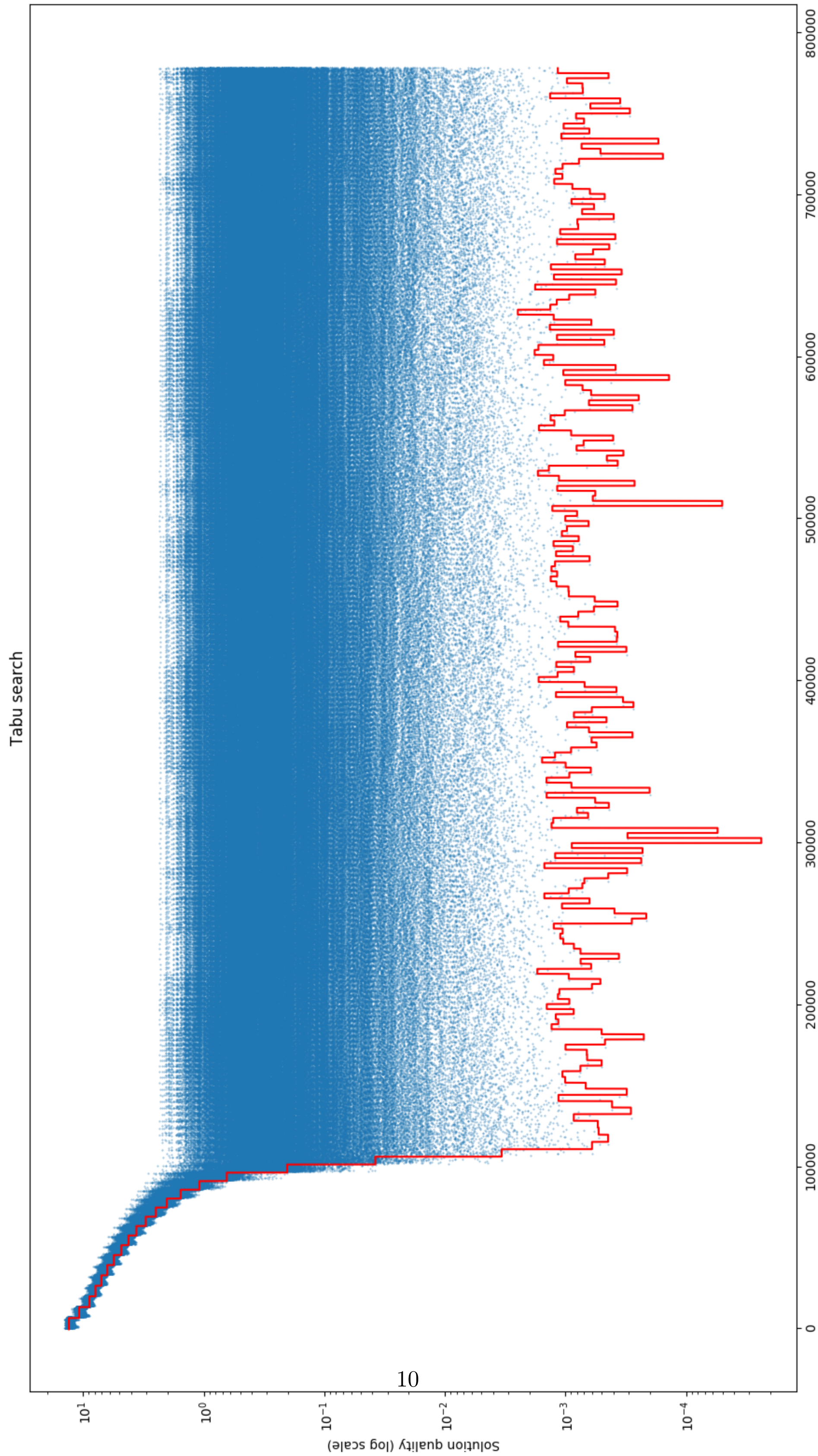
```
# Benjamin Yi
# byi649
# 925302651

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]))
```

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
Function evaluation count
Solution quality (log scale)
0
100000
200000
300000
400000
500000
600000
700000
800000
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}