In [36]:
```
!pip install numba
!pip install scipy
```

```
Requirement already satisfied: numba in /opt/conda/lib/python3.9/site-packages (0.5
6.4)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /opt/conda/lib/python3.
9/site-packages (from numba) (0.39.1)
Requirement already satisfied: numpy<1.24,>=1.18 in /opt/conda/lib/python3.9/site-pa
ckages (from numba) (1.22.3)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.9/site-packages
(from numba) (58.2.0)
Requirement already satisfied: scipy in /opt/conda/lib/python3.9/site-packages (1.9.
3)
Requirement already satisfied: numpy<1.26.0,>=1.18.5 in /opt/conda/lib/python3.9/sit
e-packages (from scipy) (1.22.3)
```

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
import warnings
from numba import njit
from scipy.optimize import curve_fit
from scipy.optimize import leastsq

warnings.filterwarnings("ignore")
%matplotlib inline
```

In [2]:
```
@njit
def init_pos(N, L):
    pos = np.random.uniform(0.0,L/2,size=(N,3))
    ph = np.zeros(pos.shape) #placeholder
    return pos - L*np.round(pos/L, 0, ph)
```

In [3]:
```
# Constants
epsilon = 1
sigma = 1
r_c = 2.5*sigma

# Parameters for Line Search and Conjugate Gradient
LStol = 10**(-8)
MaxLSSteps = 100000
ECtol = 10**(-10)
MaxCGSteps = 100000
delta = 0.1
```

In [4]:
```
@njit
def calc_energy(pos):
    u_ = 0
    N = len(pos)
    alpha = 0.0001 * (N ** (-2 / 3))
    #non_conv_term = 4*epsilon*((r_c/sigma)**(-12)-(r_c/sigma)**(-6))
    for i in range(N):
        u_ += alpha * np.dot(pos[i], pos[i])
        for j in range (i+1, N):
            r_ij = pos[j] - pos[i]
            r_ij_norm = np.linalg.norm(r_ij)
            if r_ij_norm <= r_c:
                u_ += 4*epsilon*((r_ij_norm/sigma)**(-12)-(r_ij_norm/sigma)**(-6))
    return u_
```

In [5]:
```
@njit
def calc_force(pos):
    N = len(pos)
    alpha = 0.0001*(N**(-2/3))
    f = np.zeros(pos.shape)
    for i in range(len(pos)):
        f[i] -= 2 * alpha * pos[i]
        for j in range (i+1, N):
            r_ij = pos[j] - pos[i]
            r_ij_norm = np.linalg.norm(r_ij)
            if r_ij_norm <= r_c:
                f[i]+=4*epsilon*(-12*(sigma)*(r_ij_norm)**(-13)+6*(sigma)*(r_ij_no
                f[j]-=4*epsilon*(-12*(sigma)*(r_ij_norm)**(-13)+6*(sigma)*(r_ij_no
    return f
```

In [6]:
```
@njit
def deriv_dot(pos,force): #derivative and dot product
    d = force/np.sqrt((force**2).sum())
    pos_d = -np.dot((calc_force(pos)).reshape(-1), d.reshape(-1))
    return pos_d
@njit
def bisection(pos_0,pos_2,u_0,u_2):
    while True:
        der_pos_0 = deriv_dot(pos_0,calc_force(pos_0))
        der_pos_2 = deriv_dot(pos_2,calc_force(pos_2))
        pos_mid = (pos_0 + pos_2) / 2
        der_pos_mid = deriv_dot(pos_mid,calc_force(pos_mid))

        pos_1 = (pos_0+pos_2)/2 #center position
        u_1 = calc_energy(pos_1) #potential energy at center position

        if abs(u_1 - u_0) < LStol*abs(u_1) :
            Varray = [0, 0.5, 1]
            u_arr = np.array([u_0, u_1, u_2])
            pos_x = parabola_min(Varray, u_arr)
            pos_min = pos_0 + pos_x * (pos_2 - pos_0)

            return pos_min, calc_energy(pos_min)

        if der_pos_mid * der_pos_2 > 0 :
            pos_2 = pos_mid
        else:
            pos_0 = pos_mid
        u_0, u_2 = calc_energy(pos_0), calc_energy(pos_2)
@njit
def parabola_min(x, y):

    V = np.vander(x) # Vandermonde matrix from the positions
    a, b, c = np.linalg.solve(V, y) # get coefficients of the polynomial
    if a == 0:
        xp = - c / b     # x= -c/b  if a=0, (else divide by 0 encounters)
    else:
        xp = - b / (2 * a) # x = -b/2a
    return xp
@njit
def ls(pos, f, delta, LStol, MaxLSSteps):
    #line search finds the interval between which we have our minima
    #and bisection method locates the minima

    #clip and normalize
    f = np.clip(f, -10000, 10000)
    f = f / np.sqrt(np.sum(f * f))

    i=0
    j=0
    pos_0 = pos
    u_0 = calc_energy(pos_0)
    #we could set delta=0.5 until Δu=10^-2 and after that to 0.01. it can reduce ti
    delta=0.01

    pos_1 = pos_0 + delta*f
    u_1 = calc_energy(pos_1)
    pos_2 = pos_1 + delta*f
    u_2 = calc_energy(pos_2)

    while i<MaxLSSteps:
        if u_0>u_1 and u_1<u_2:
            return bisection(pos_0,pos_2,u_0,u_2)
        elif u_0<u_1 and i==0:
            #for case it skips the minima in first step
            return bisection(pos_0,pos_1,u_0,u_1)
        else:
            pos_0, pos_1 = pos_1, pos_2
            u_0, u_1 = u_1, u_2
            pos_2 = pos_1 + delta*f
            u_2=calc_energy(pos_2)
        if abs((u_1 - u_2)/u_2) < 10**-2:
            delta=0.01
        i+=1
    return pos_1,u_2
```

In [7]:

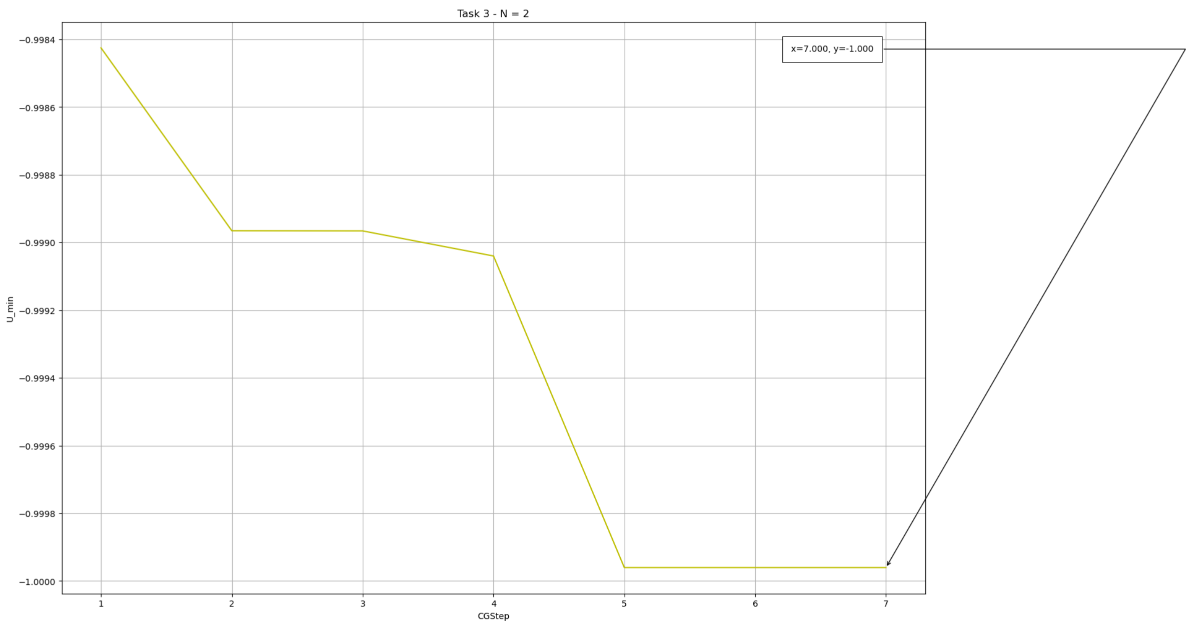
```
@njit
def cg(pos):
    #PES is an array to save all Potential Energy values
    PES = []

    #Initial values
    forces = calc_force(pos)
    PE = calc_energy(pos)
    dir = forces

    #Big enough to enter the loop
    old_PE = 10000000

    #Iterator of CG
    CGStep = 0

    #Conjugate Gradient
    while abs(PE - old_PE) > ECtol*abs(PE) and CGStep < MaxCGSteps:

        old_PE = PE
        CGStep += 1
        PE = ls(pos, dir, delta, LStol, MaxLSSteps)[1]
        pos = ls(pos, dir, delta, LStol, MaxLSSteps)[0]

        PES.append(PE)

        old_forces = forces
        forces = calc_force(pos)

        gamma = np.sum((forces - old_forces) * forces) / np.sum(old_forces * old_fo
        dir = forces + gamma * dir
    return CGStep, PE, PES, pos
```

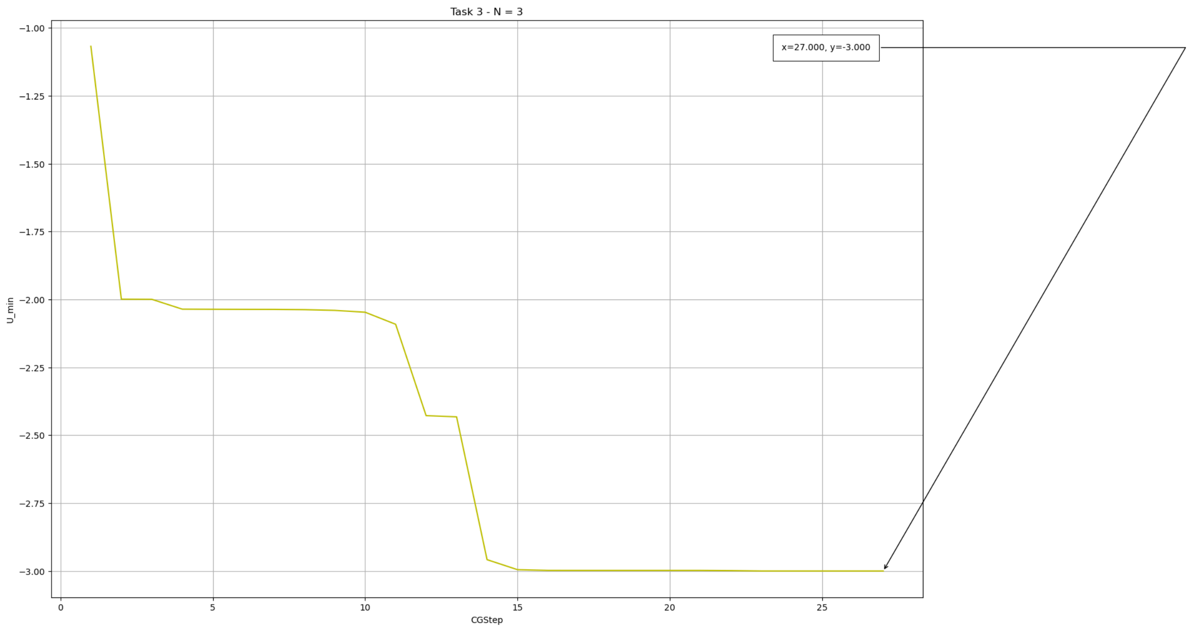
In [8]:

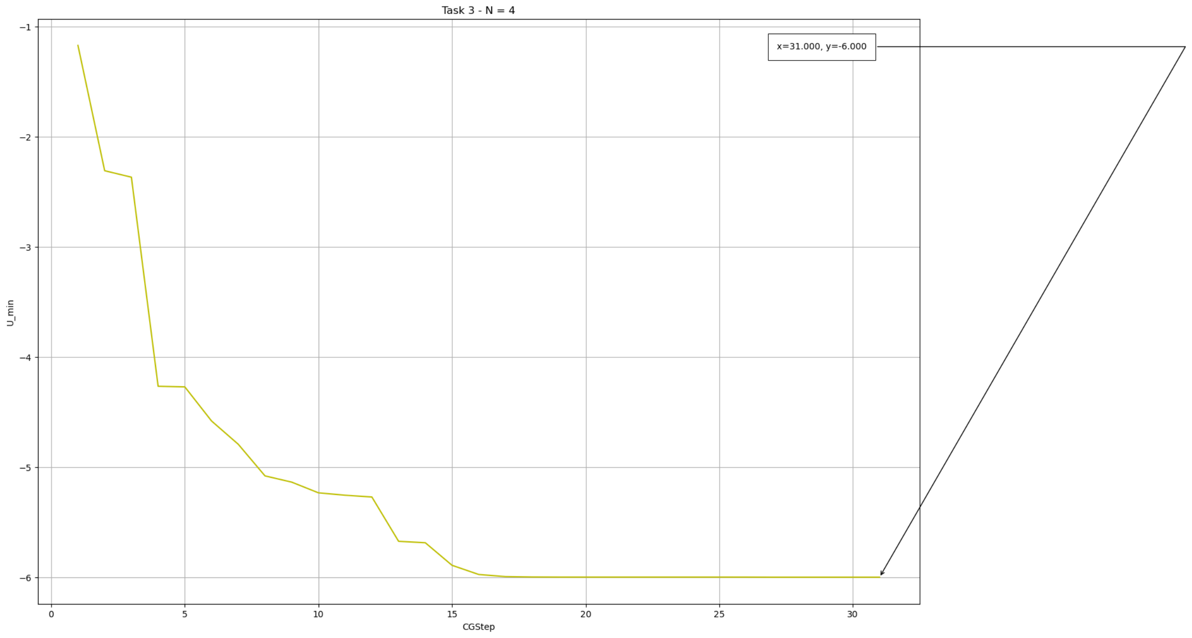
```
#Help function for plotting and Task 4
def annot_max(x,y, ax=None):
    xmin = x[np.argmin(y)]
    ymin = y.min()
    text= "x={:.3f}, y={:.3f}".format(xmin, ymin)
    if not ax:
        ax=plt.gca()
    bbox_props = dict(boxstyle="square,pad=1", fc="w", ec="k", lw=0.72)
    arrowprops=dict(arrowstyle="->",connectionstyle="angle,angleA=0,angleB=60")
    kw = dict(xycoords='data',textcoords="axes fraction",
              arrowprops=arrowprops, bbox=bbox_props, ha="right", va="top")
    ax.annotate(text, xy=(xmin, ymin), xytext=(0.94,0.96), **kw)

def Task4(n_array,u_min):
    def func(params, n_array, u_min):
        a, b, c = params[0], params[1], params[2]
        residual = u_min - (a*n_array**(2/3)+b*n_array+c)
        return residual
    u_macro=np.zeros(len(u_min))
    params = [0, 0, 0]
    result = leastsq(func, params, (n_array, u_min))
    a, b, c = result[0][0], result[0][1], result[0][2]
    for i in range(len(u_min)):
        u_macro[i] = a*n_array[i]**(2/3)+b*n_array[i]+c

    plt.figure(figsize = (18, 12))
    plt.grid()
    plt.plot(n_array,u_min-u_macro)
    plt.xlabel('N')
    plt.ylabel('U_min-U_macro')
    plt.title('task 4')
    plt.show()

    print("The parameters of magic clusters")
    print(f"a = {c}")
    print(f"b = {a}")
    print(f"c = {b}")
```

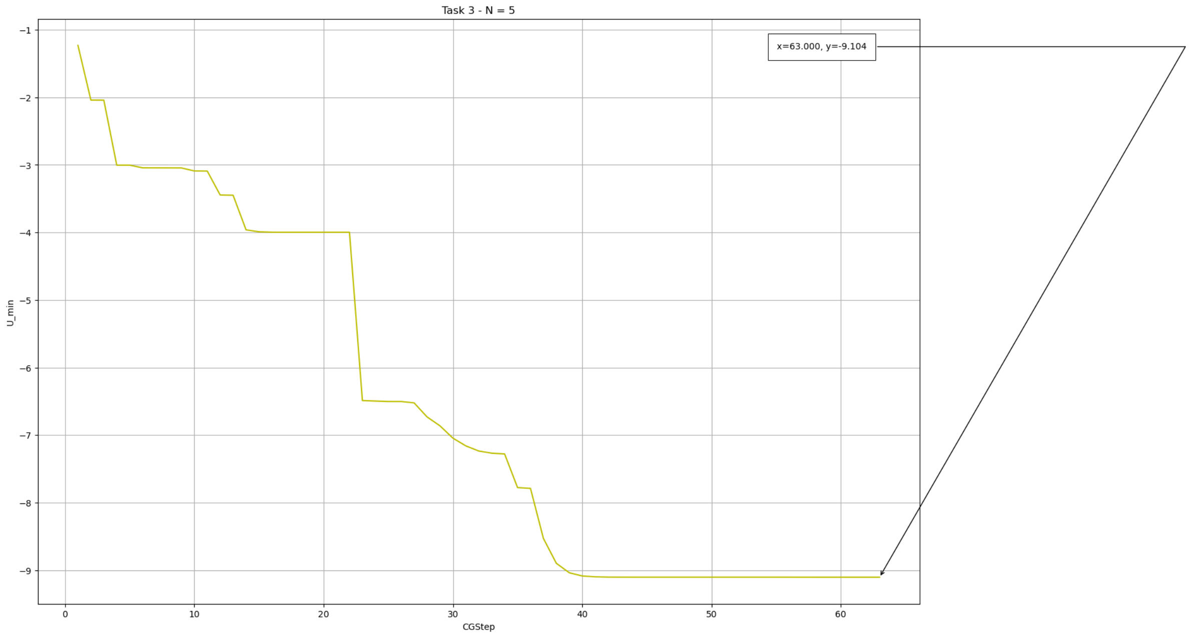
In [14]:

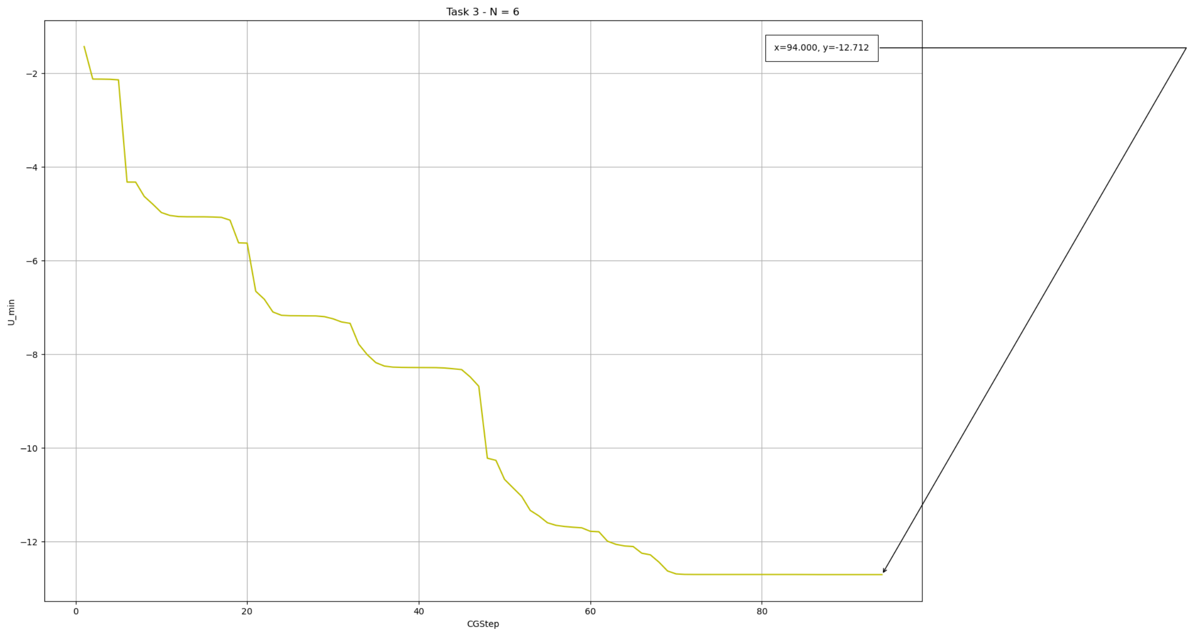
```
%%time
def run(N, M):
    res = []
    n_array = np.arange(2, N+1, 1)
    cgsteps = []
    umin_n = []
    for i in range(2, N+1):
        L = (i / 0.01) ** (1 / 3)
        for k in range(M):
            t = init_pos(i, L)
            res.append(list(cg(t)))

        #Capture needed values and index of minimal energy
        new = np.array(res)
        energy_array = new[:, 1]
        index = np.argmin(energy_array)

        #Task 2
        umin_n.append(new[index, 1])
        cgsteps.append(new[index, 0])

        #Task 3
        U_min = np.array(new[index, 2])
        Steps = np.arange(1, len(U_min)+1)

        #Plot Task 3
        plt.figure(figsize = (18, 12))
        plt.grid()
        plt.plot(Steps, U_min, 'y')
        plt.xlabel('CGStep')
        plt.ylabel('U_min')
        plt.title('Task 3 - N = %i' %i)
        annot_max(Steps,U_min)
        plt.show()

    #Plot Task 2a
    plt.figure(figsize = (18, 12))
    plt.grid()
    plt.plot(n_array, cgsteps, 'r')
    plt.xlabel('N')
    plt.ylabel('CG Steps')
    plt.title('Task 3')
    plt.show()

    #Plot Task 2b
    plt.figure(figsize = (18, 12))
    plt.grid()
    plt.plot(n_array, umin_n, 'b')
    plt.xlabel('N')
    plt.ylabel('U_min')
    plt.title('Task 3')
    plt.show()

    #Plot Task 4
    Task4(n_array, umin_n)

run(25,1000)
```

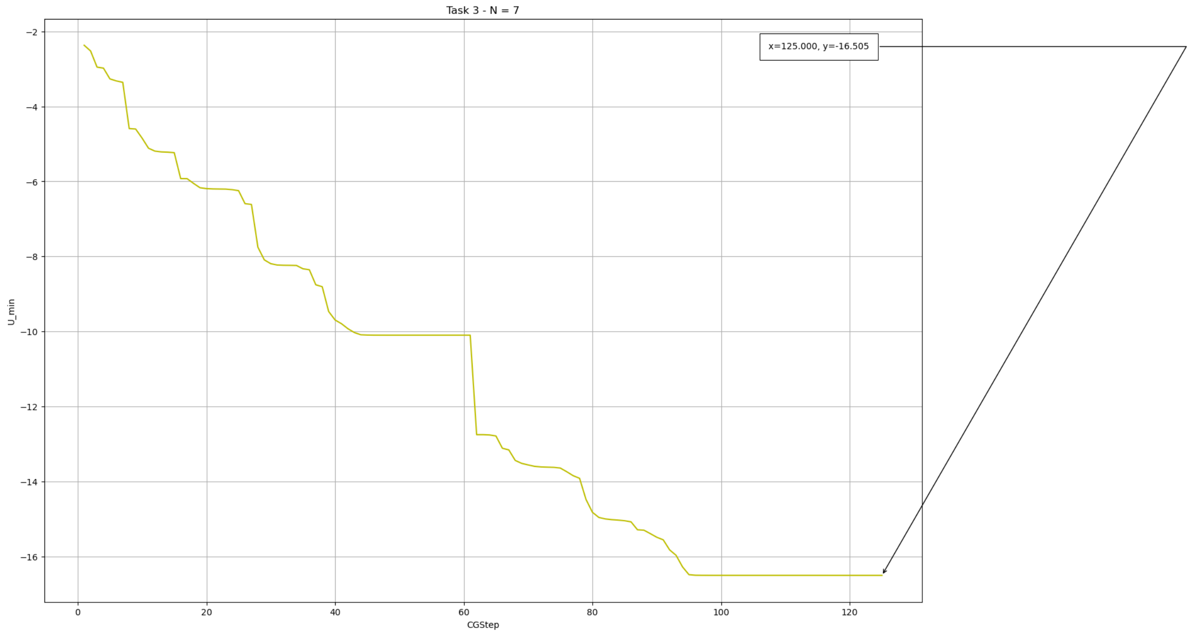


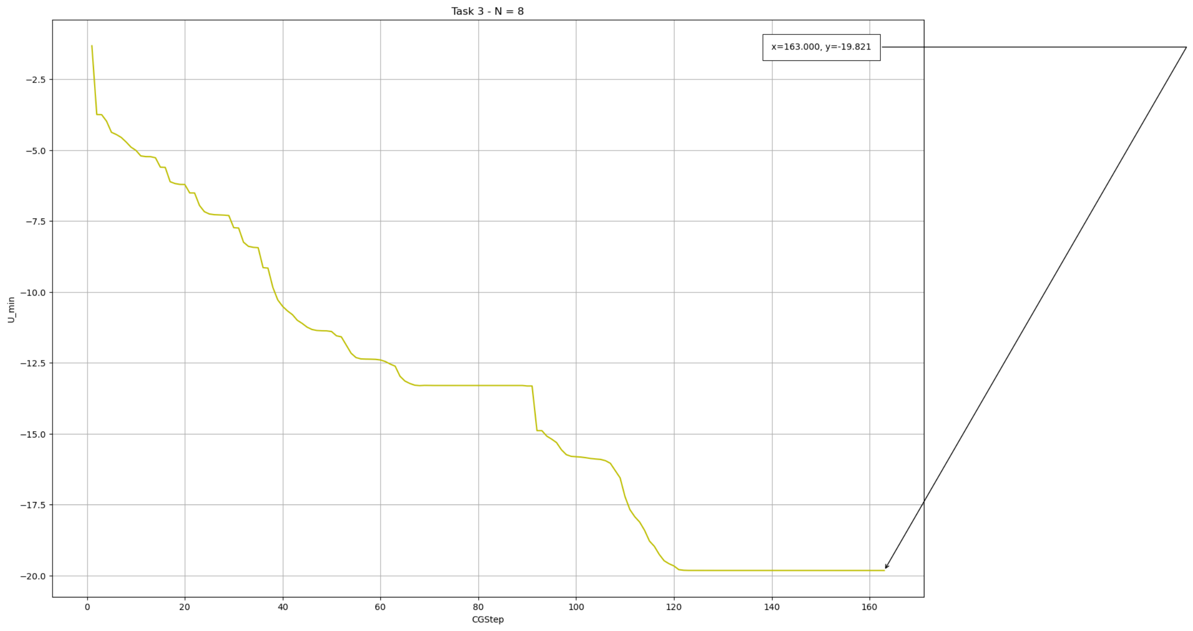

Task 3 - N = 3
x=27.000, y=-3.000
U_min
CGStep
−1.00
−1.25
−1.50
−1.75
−2.00
−2.25
−2.50
−2.75
−3.00
0
5
10
15
20
25

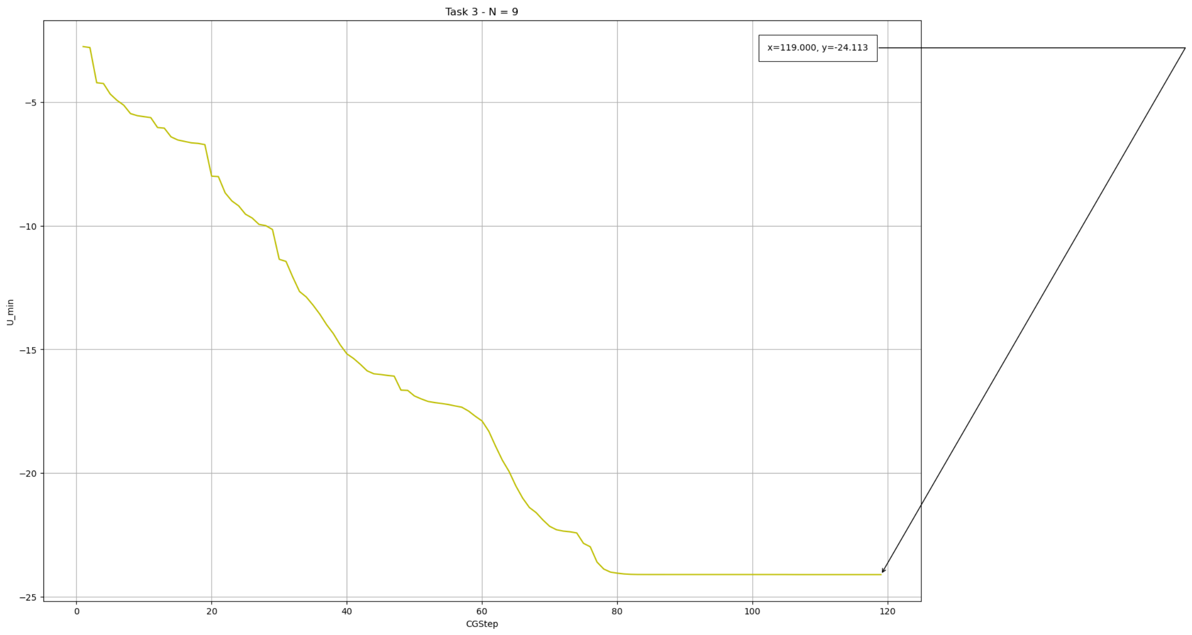

Task 3 - N = 4
x=31.000, y=-6.000
U_min
CGStep
−1
−2
−3
−4
−5
−6
0
5
10
15
20
25
30

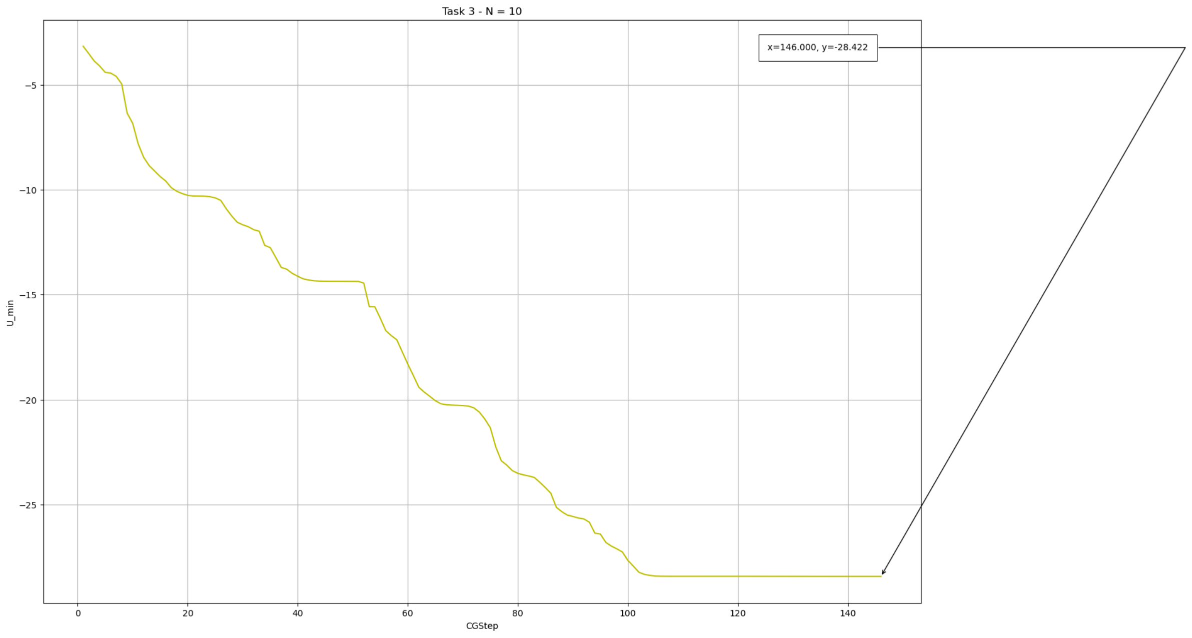

Task 3 - N = 5
x=63.000, y=-9.104
U_min
CGStep
−1
−2
−3
−4
−5
−6
−7
−8
−9
0
10
20
30
40
50
60

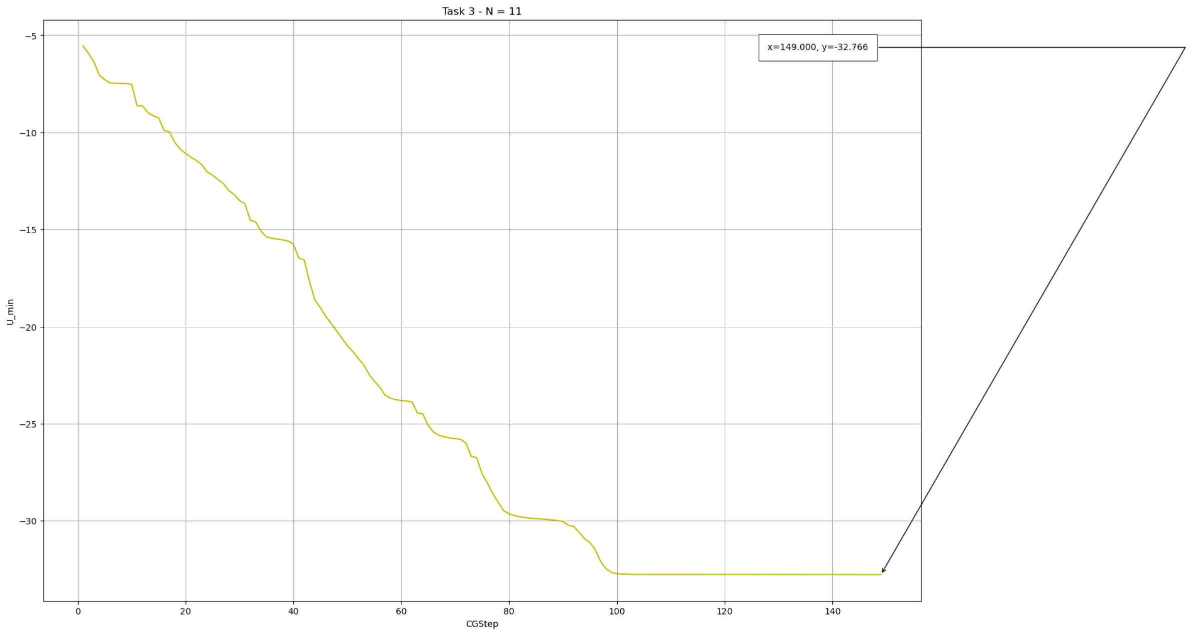

Task 3 - N = 6
x=94.000, y=-12.712
U_min
CGStep

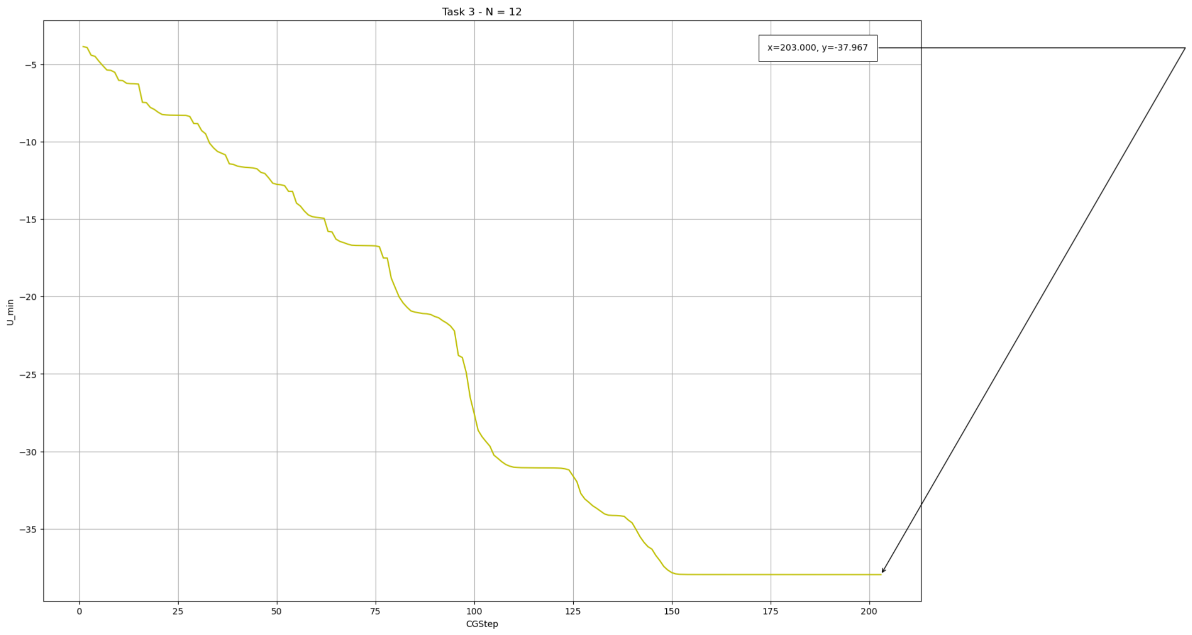

Task 3 - N = 7
x=125.000, y=-16.505
U_min
CGStep

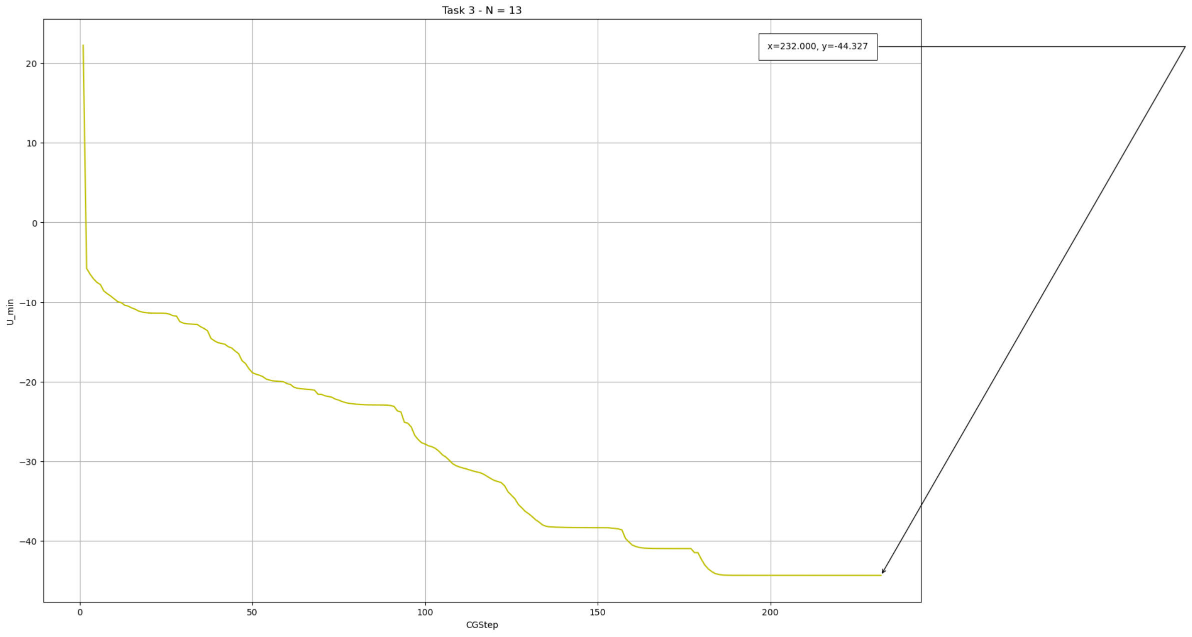

Task 3 - N = 8
x=163.000, y=-19.821
U_min
CGStep

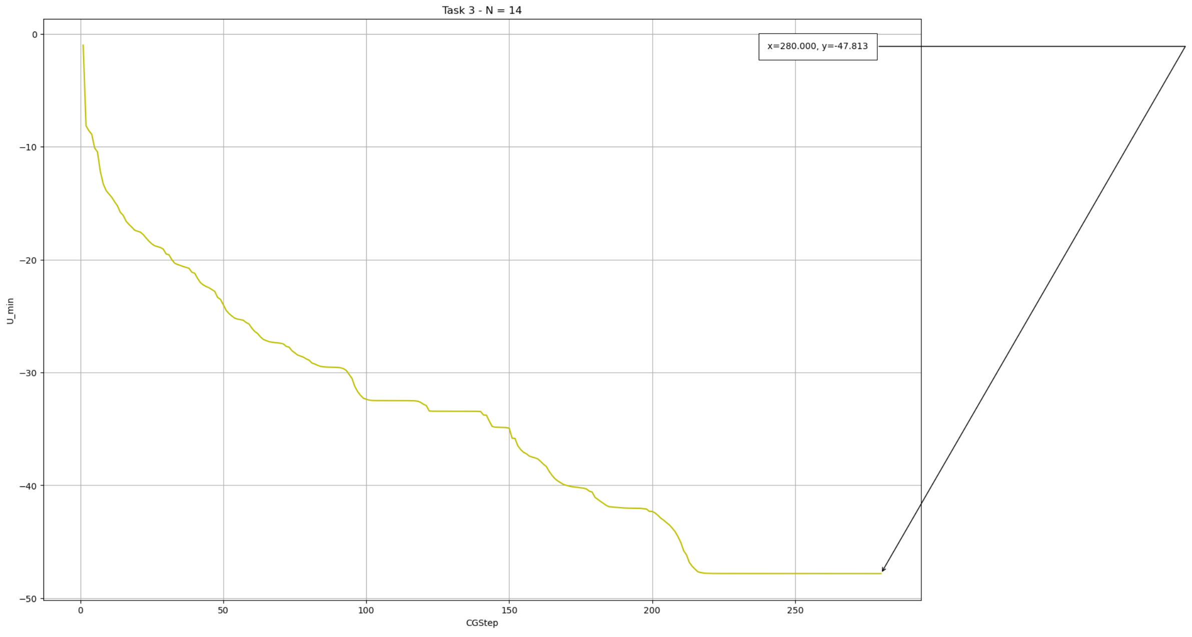

Task 3 - N = 9
x=119.000, y=-24.113
U_min
CGStep

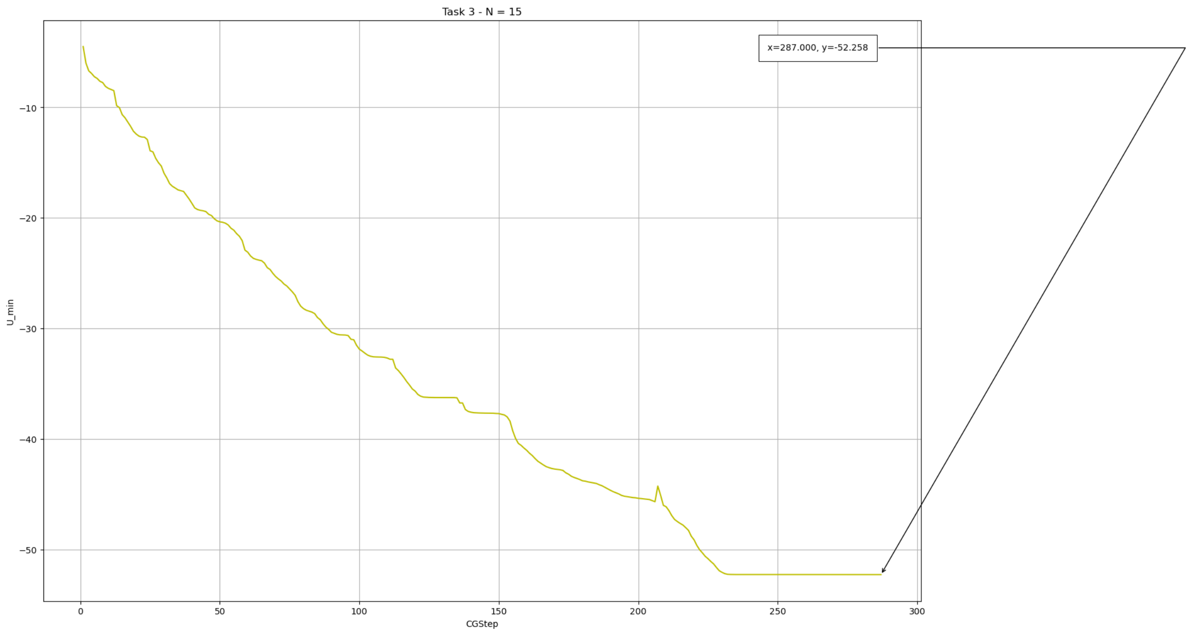

Task 3 - N = 10
x=146.000, y=-28.422
U_min
CGStep

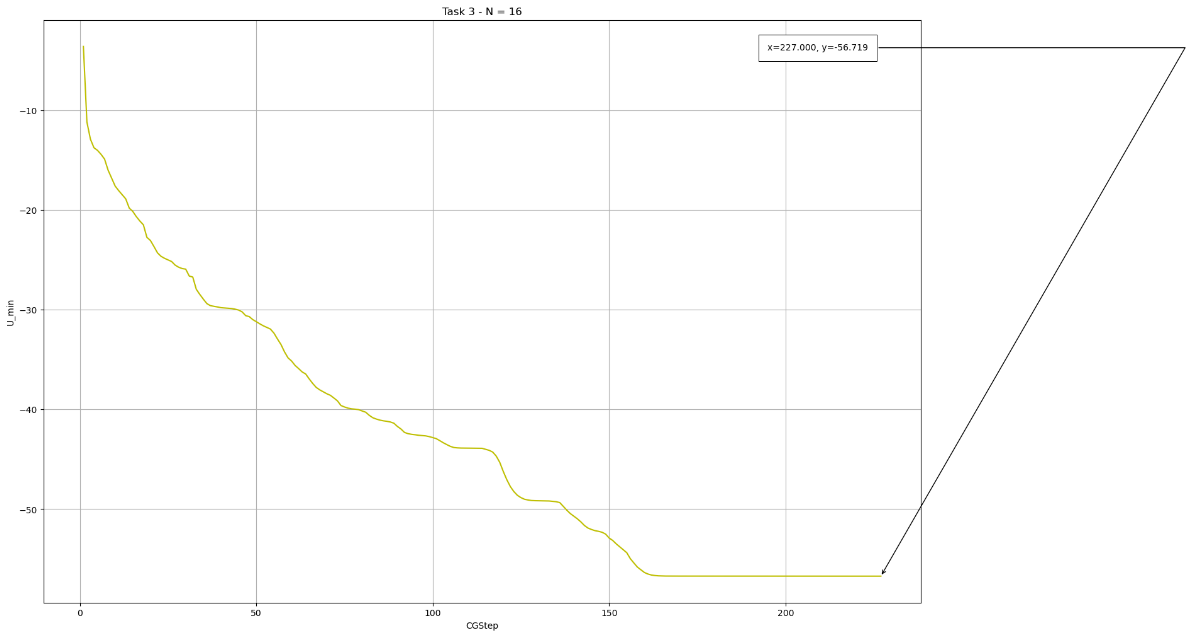

Task 3 - N = 11
x=149.000, y=-32.766
U_min
CGStep

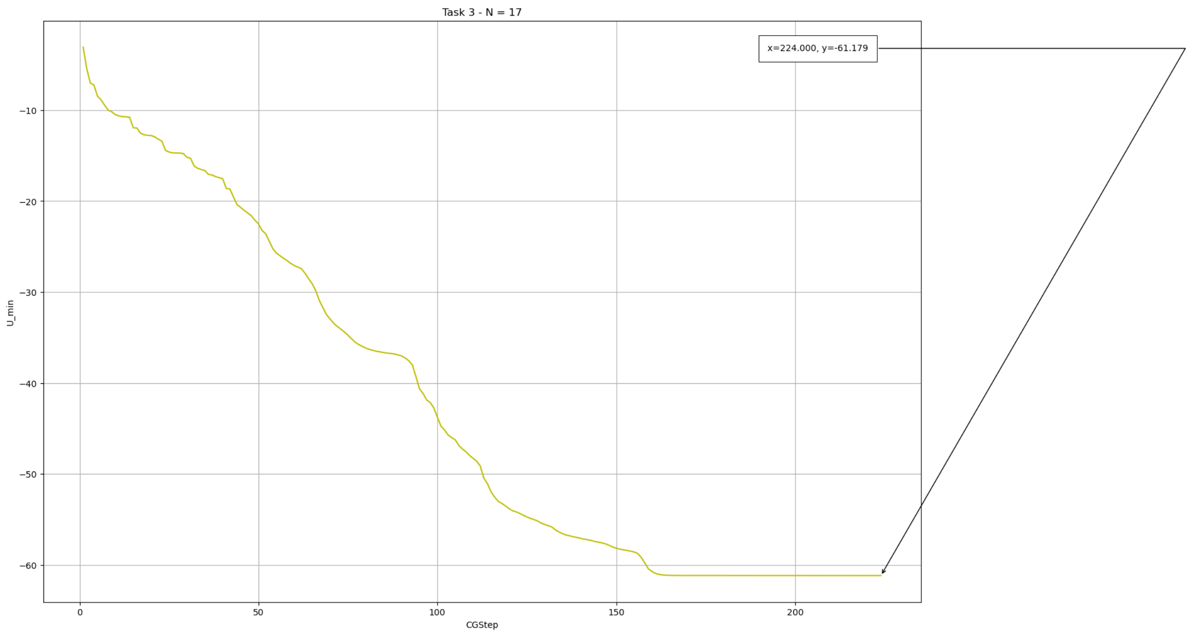

Task 3 - N = 12
x=203.000, y=-37.967
U_min
CGStep

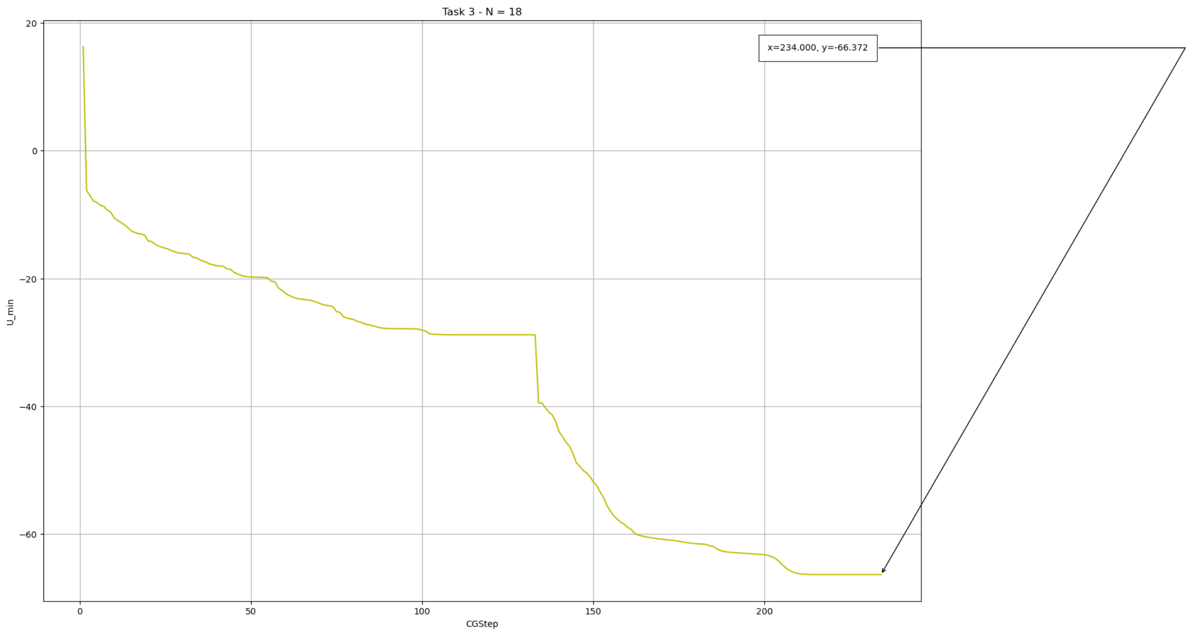

Task 3 - N = 13
x=232.000, y=-44.327
U_min
CGStep

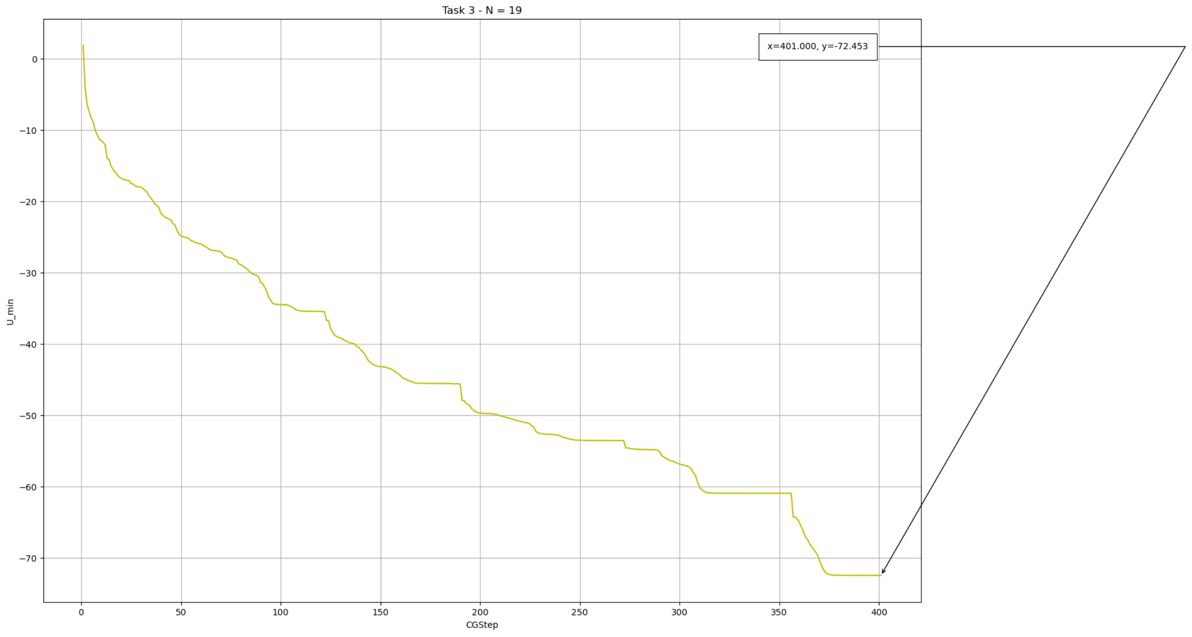

Task 3 - N = 14
x=280.000, y=-47.813
U_min
CGStep

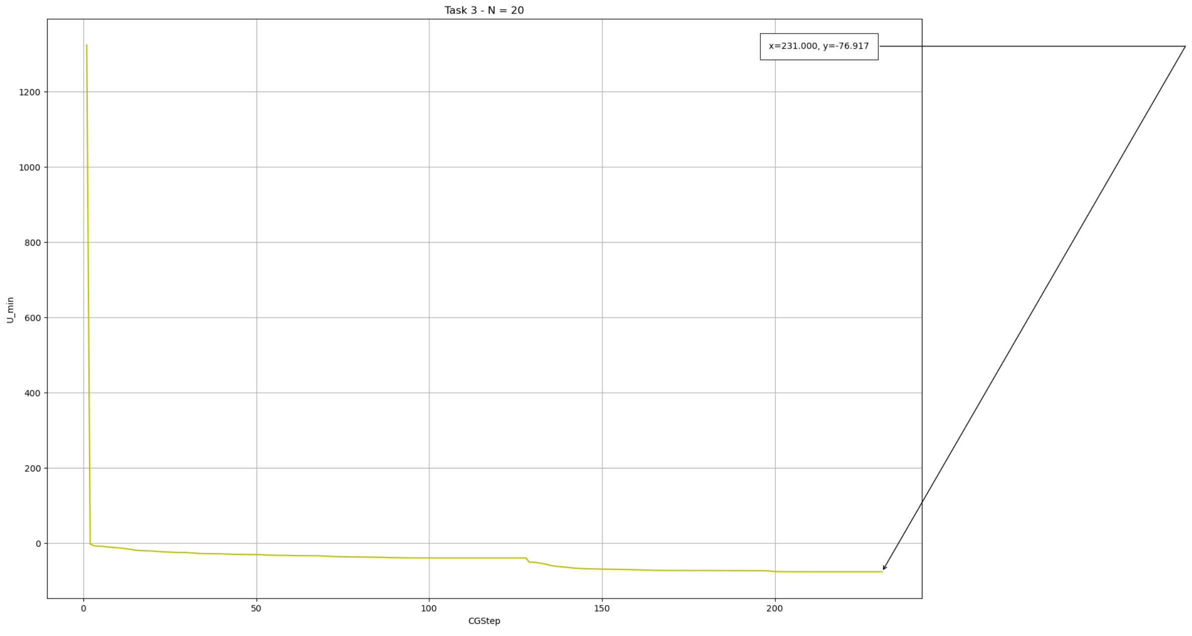

Task 3 - N = 15
x=287.000, y=-52.258
U_min
CGStep
−10
−20
−30
−40
−50
0
50
100
150
200
250
300

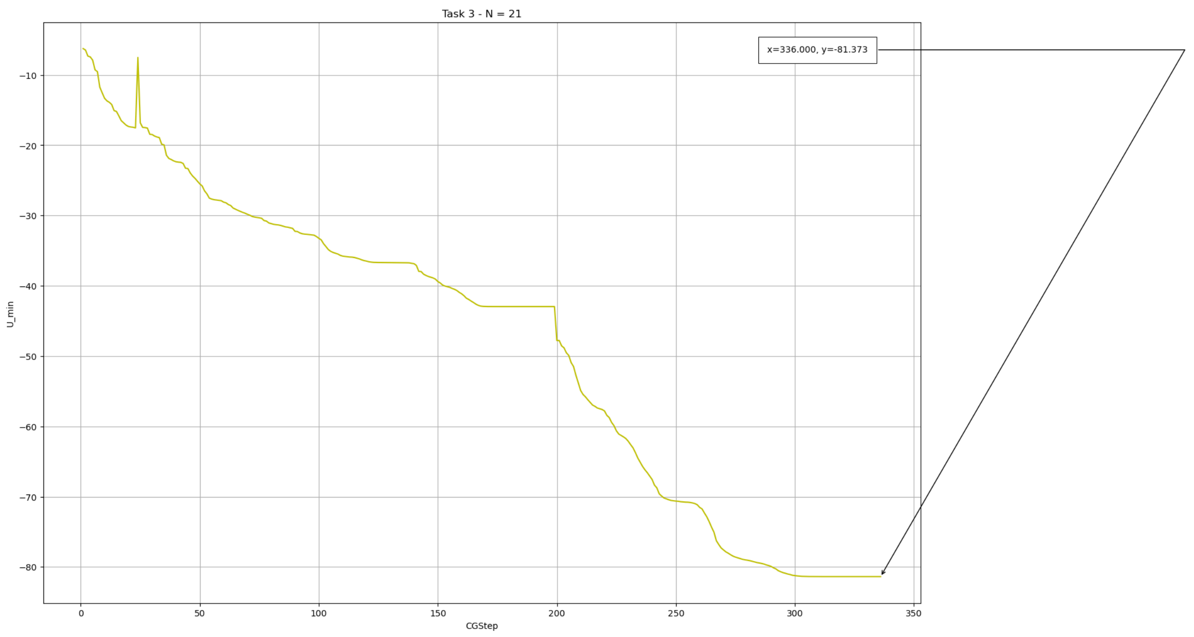

Task 3 - N = 16
x=227.000, y=-56.719
U_min
CGStep
−10
−20
−30
−40
−50
0
50
100
150
200

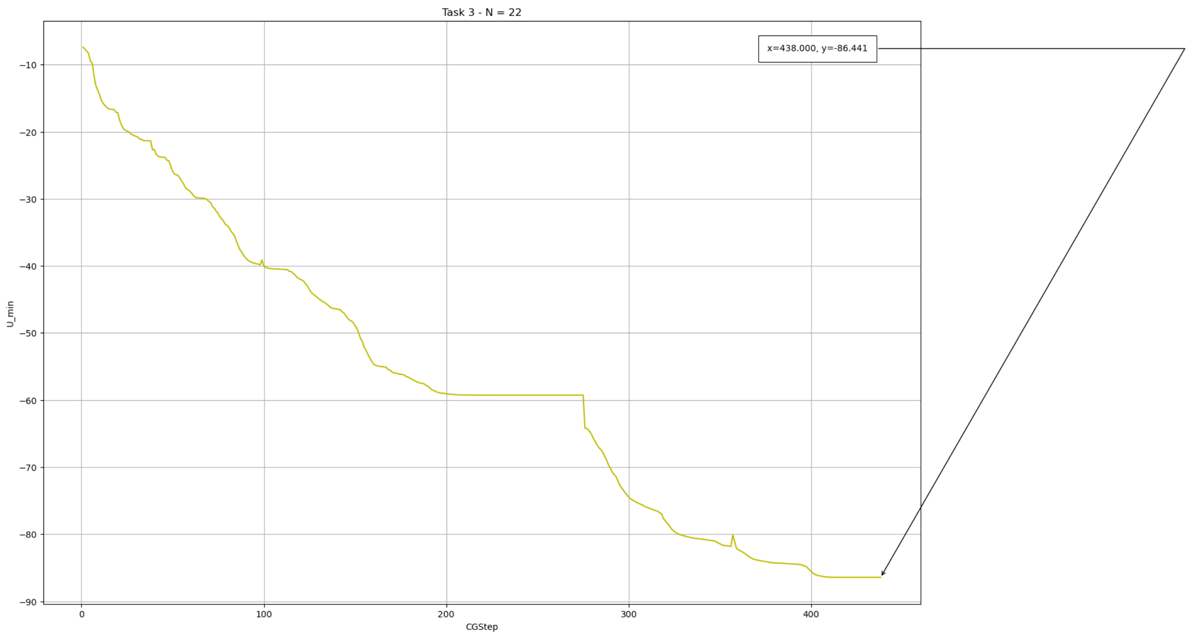

Task 3 - N = 17
x=224.000, y=-61.179
U_min
CGStep
−10
−20
−30
−40
−50
−60
0
50
100
150
200

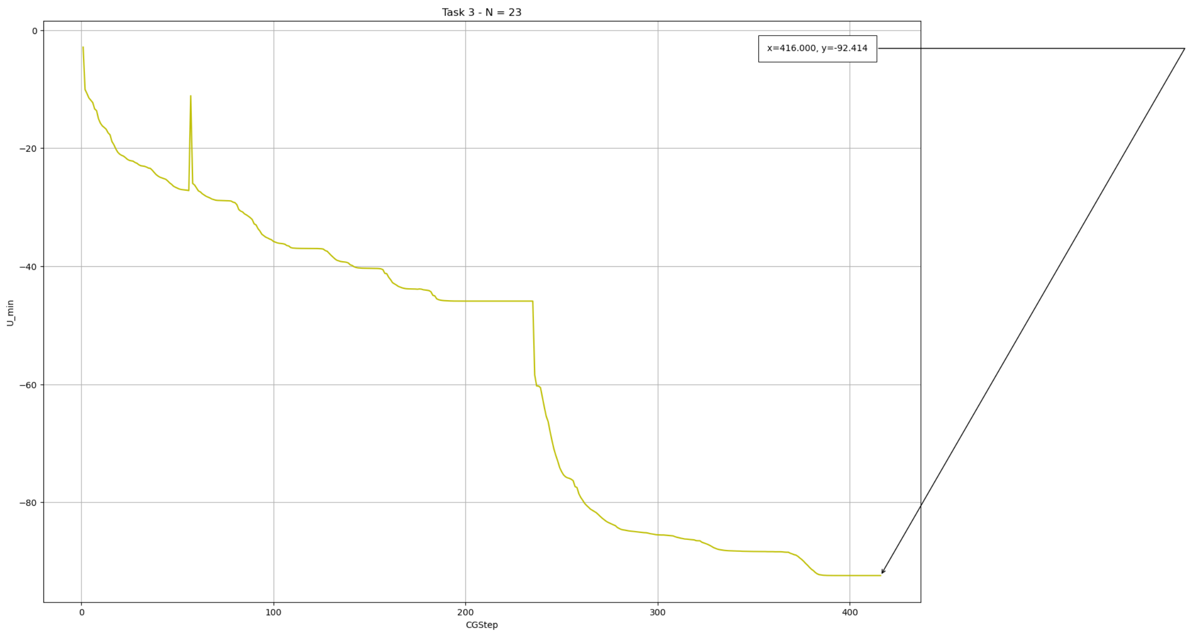

Task 3 - N = 18
x=234.000, y=-66.372
U_min
CGStep
20
0
−20
−40
−60
0
50
100
150
200

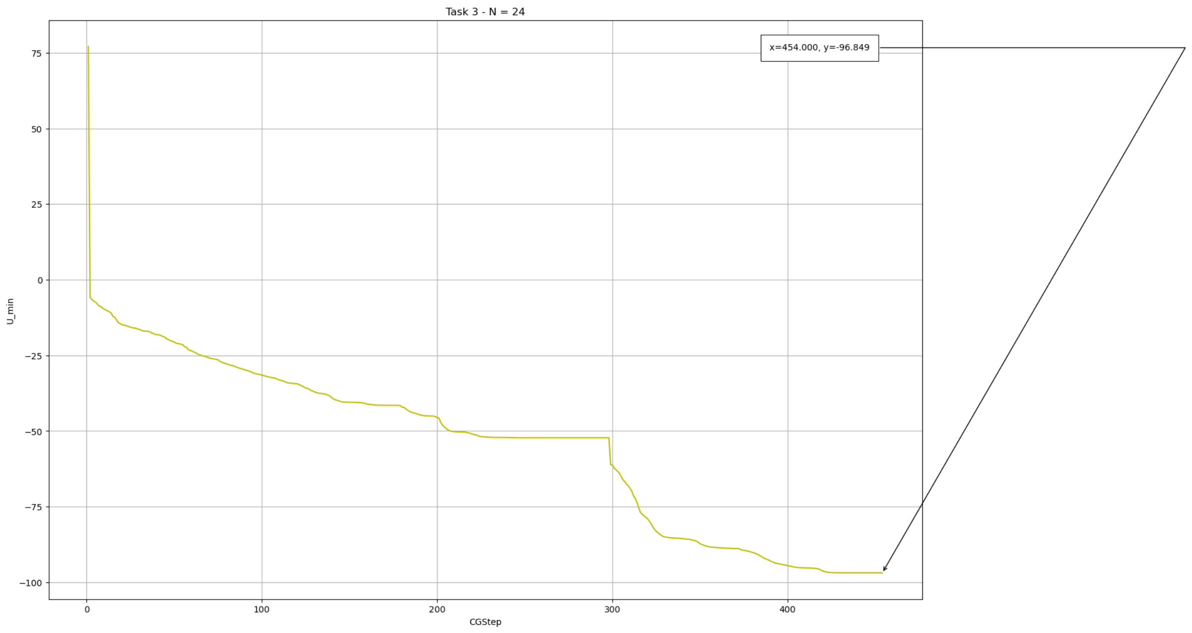

Task 3 - N = 19
x=401.000, y=-72.453
U_min
CGStep
0
−10
−20
−30
−40
−50
−60
−70
0
50
100
150
200
250
300
350
400


Task 3 - N = 20
x=231.000, y=-76.917
U_min
CGStep
1200
1000
800
600
400
200
0
0
50
100
150
200

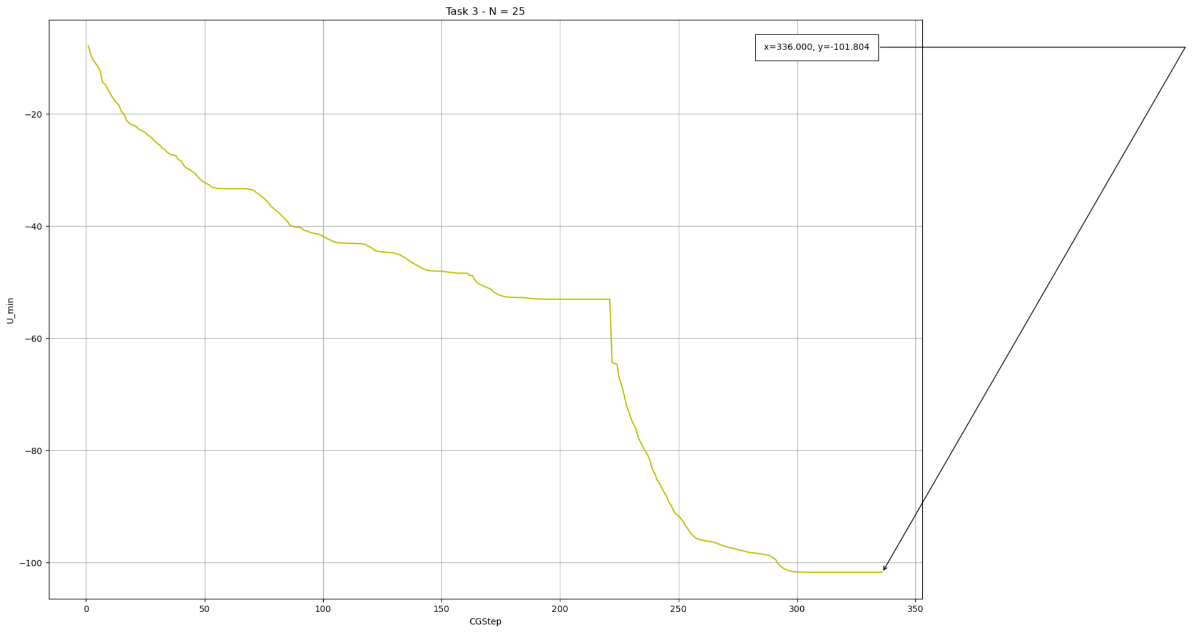

Task 3 - N = 21
x=336.000, y=-81.373
U_min
CGStep
0
50
100
150
200
250
300
350
−10
−20
−30
−40
−50
−60
−70
−80


Task 3 - N = 22
x=438.000, y=-86.441
U_min
CGStep
0
100
200
300
400
−10
−20
−30
−40
−50
−60
−70
−80
−90

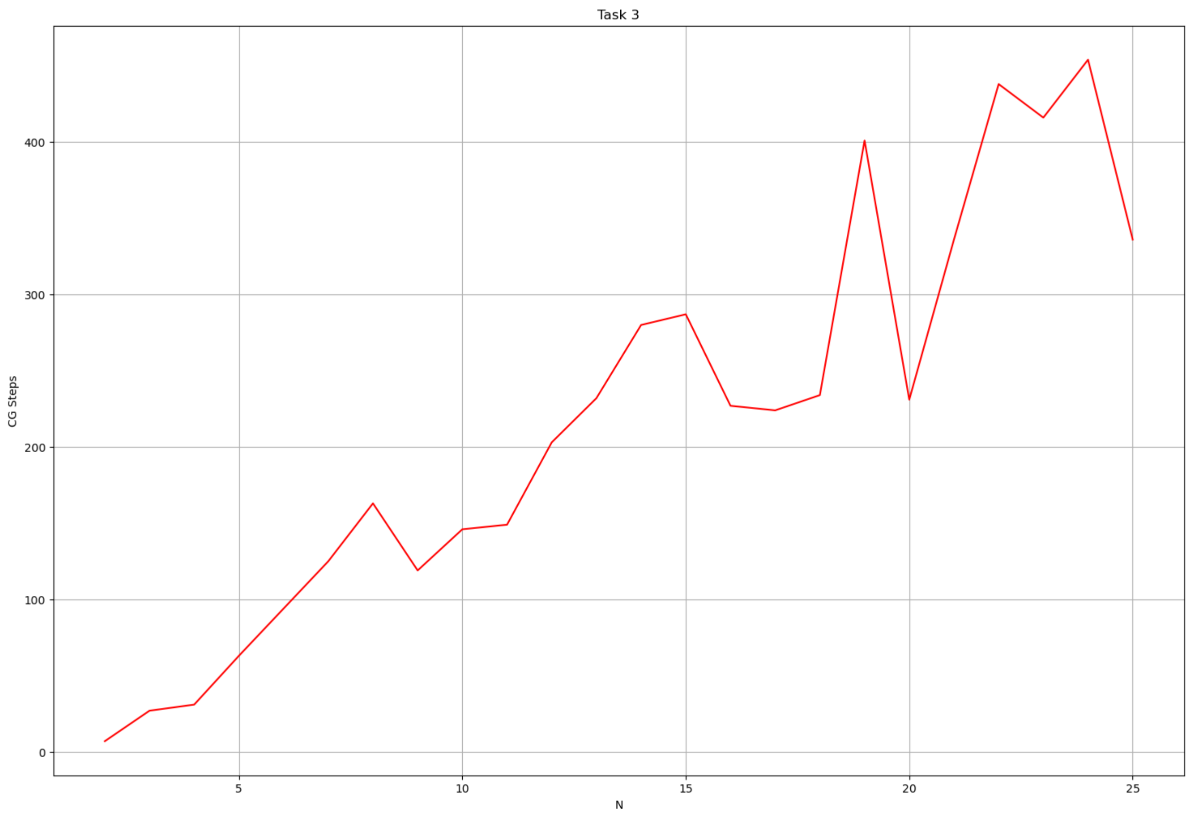

Task 3 - N = 23
x=416.000, y=-92.414
U_min
CGStep
0
100
200
300
400
0
−20
−40
−60
−80

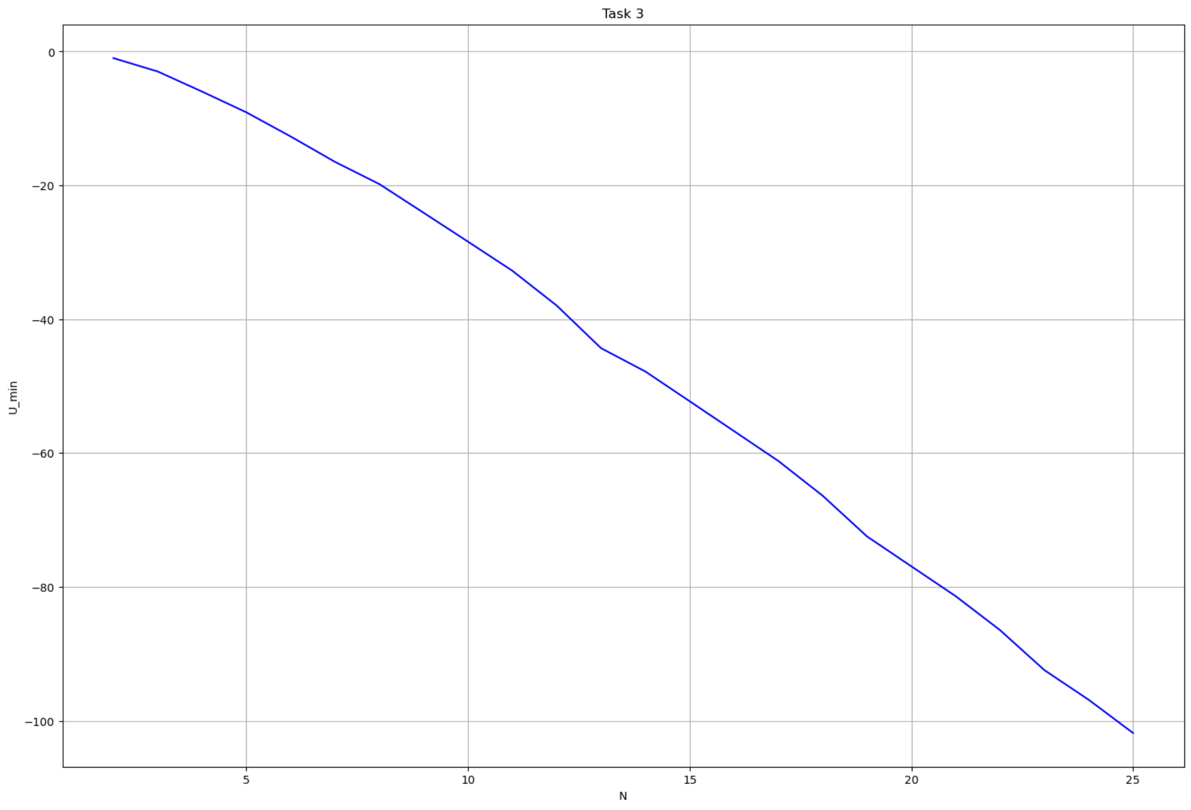


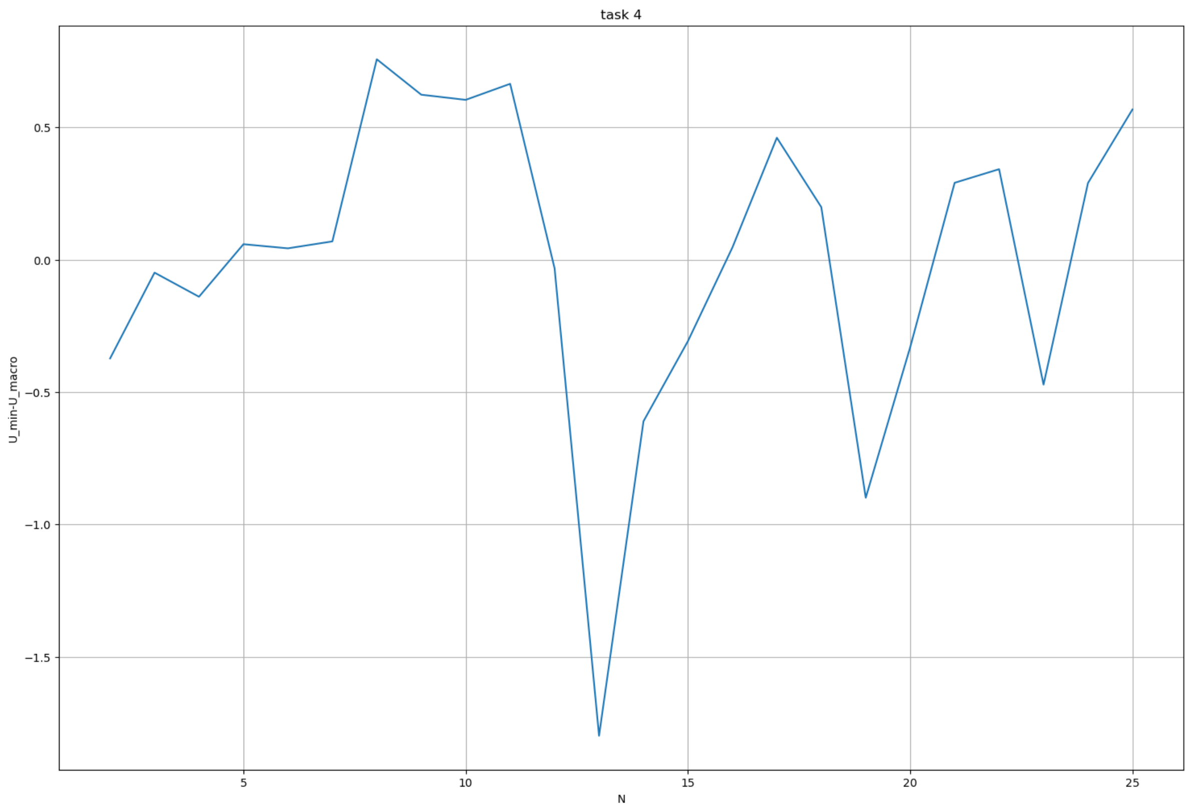
Task 3 - N = 24
x=454.000, y=-96.849
U_min
CGStep
75
50
25
0
−25
−50
−75
−100
0
100
200
300
400



Task 3 - N = 25
x=336.000, y=-101.804
U_min
CGStep
−20
−40
−60
−80
−100
0
50
100
150
200
250
300
350


Task 3
CG Steps
N
0
100
200
300
400
5
10
15
20
25


Task 3
U_min
N
0
−20
−40
−60
−80
−100
5
10
15
20
25



```
The parameters of magic clusters
a = -2.6332634989900776
b = 11.052894870939776
c = -7.769561605792467
Wall time: 1d 1h 20min 50s
```

```
In [ ]:
```