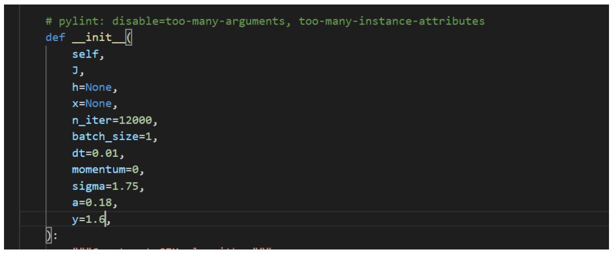
5001HW3

ZHOU Yang

Dec. 1st, 2024

1 Best result of G1 Max-cut

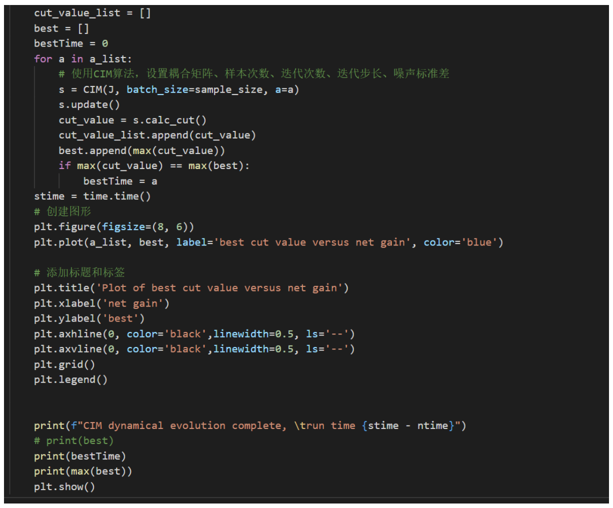
The best result I obtained is 11608. I primarily tuned the parameter a (which corresponds to $p-1$ in the PPT, representing net gain), while keeping the weight of noise fixed at 1.6 (corresponding to a temperature of 0.0128K). The iteration step size was set to 0.01, with a total of 12000 iterations. Besides this, a difference from the Euler method shown in the PPT is that I adjusted the variance of the white noise, setting it (sigma) to 1.75. The default batch size is 1, but when calling the function, the actual passed batch size parameter was 50, meaning that for each set of identical parameters, 50 tests were run simultaneously. (Figure 1)



```
# pylint: disable=too-many-arguments, too-many-instance-attributes
def __init__(
    self,
    J,
    h=None,
    x=None,
    n_iter=12000,
    batch_size=1,
    dt=0.01,
    momentum=0,
    sigma=1.75,
    a=0.18,
    y=1.6,
):
```

Figure 1: parameters

First, all spin values are initialized with random numbers uniformly distributed in (-0.5, 0.5). Then, for each value of a in the a_list and other fixed parameter values, 50 tests are run, and the maximum results are recorded in the cut_value_list. Finally, I plot cut_value_list against a_list and print the value of a that yields the maximum in cut_value_list along with the maximum value itself. (Figure 2)



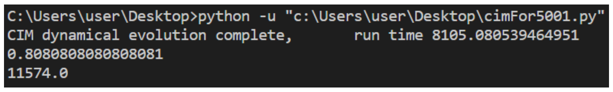
```python
cut_value_list = []
best = []
bestTime = 0
for a in a_list:
    # 使用CIM算法，设置耦合矩阵、样本次数、迭代次数、迭代步长、噪声标准差
    s = CIM(J, batch_size=sample_size, a=a)
    s.update()
    cut_value = s.calc_cut()
    cut_value_list.append(cut_value)
    best.append(max(cut_value))
    if max(cut_value) == max(best):
        bestTime = a
stime = time.time()
# 创建图形
plt.figure(figsize=(8, 6))
plt.plot(a_list, best, label='best cut value versus net gain', color='blue')

# 添加标题和标签
plt.title('Plot of best cut value versus net gain')
plt.xlabel('net gain')
plt.ylabel('best')
plt.axhline(0, color='black',linewidth=0.5, ls='--')
plt.axvline(0, color='black',linewidth=0.5, ls='--')
plt.grid()
plt.legend()


print(f"CIM dynamical evolution complete, \trun time {stime - ntime}")
# print(best)
print(bestTime)
print(max(best))
plt.show()
```

Figure 2: main function

I initially set the range of a to np.linspace(0, 5, 100), and the best result obtained is shown in the figure below (Figure 3), where 0.8080808080808081 is the parameter value of a when the best result occurred, and 11574 is the best result.



```
C:\Users\user\Desktop>python -u "c:\Users\user\Desktop\cimFor5001.py"
CIM dynamical evolution complete,       run time 8105.080539464951
0.8080808080808081
11574.0
```

Figure 3: First time result

The plot is fluctuating but shows a descending trend for $a > 2$. (I forgot to save the plot here.) So the second time, I set the range of a to np.linspace(0, 2, 100). The plot is shown as below:

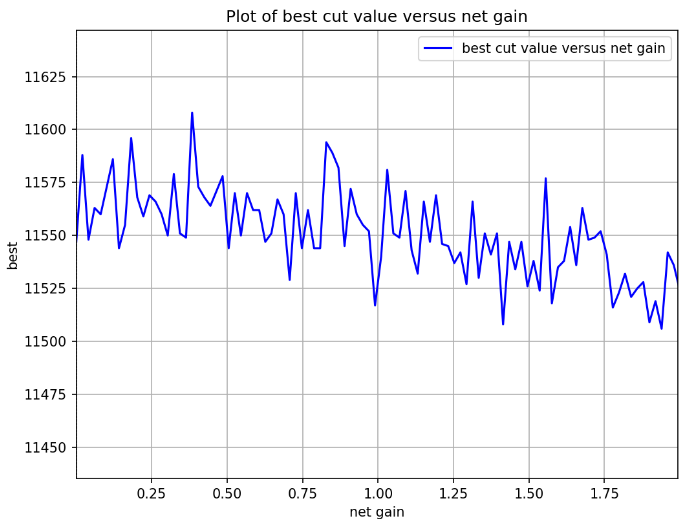
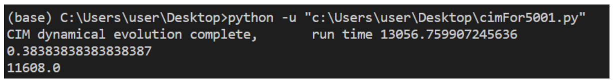


Figure 4: plot for a in (0, 2)

The maximum is 11608.0, when $a = 0.38383838383838387$ (See in Figure 5). From the above plot, we can see that for $a > 1$, although may not be significant, the best cut value also shows a descending trend. Combining this conclusion and the previous best a (0.8080808080808081 and 0.38383838383838387), I think that finding the most suitable a in (0,1) and dividing the interval as finely as possible will help bring the maximum closer to the best known. However, due to the long runtime and my poor computer, I haven't carried out the experiment yet. Details of the code are in cimFor5001.py



```
(base) C:\Users\user\Desktop>python -u "c:\Users\user\Desktop\cimFor5001.py"
CIM dynamical evolution complete,       run time 13056.759907245636
0.38383838383838387
11608.0
```

Figure 5: maximum cut value and corresponding a

2 Amplitude for each node over time

To better show how the amplitude bifurcates over time, we set the initial amplitude of each node to 0, add self.spin_history to the attributes to record the amplitude value of each iteration and plot it at the end.(Figure 6)

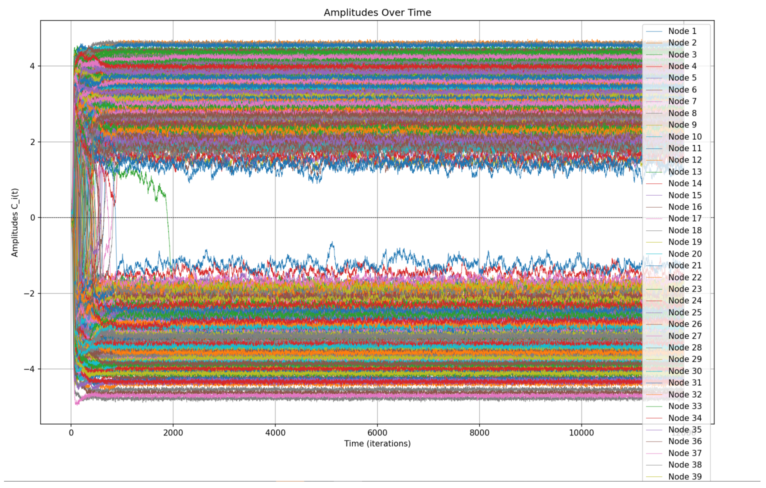


Figure 6: amplitude for each node over time

Details of the code are in cimFor5001Q3.py

3 Cut-size over time

The average cut size over 10 samples versus interaction steps t is given as below. (Figure 7)

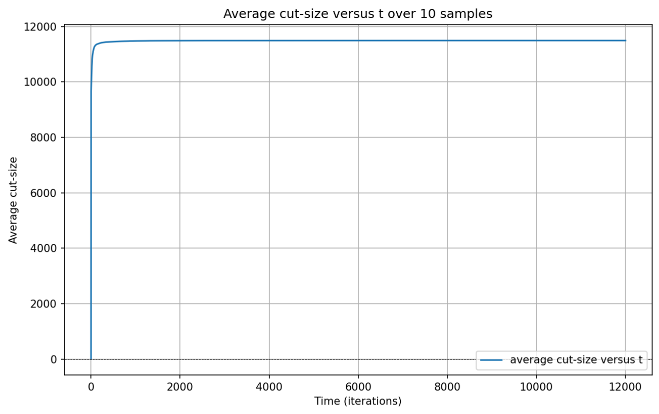


Figure 7: average cut size over 10 samples versus interaction steps t

Details of the code can be found in cimFor5001Q4.py