PCIC 2021 Competition
Causal Discovery Track

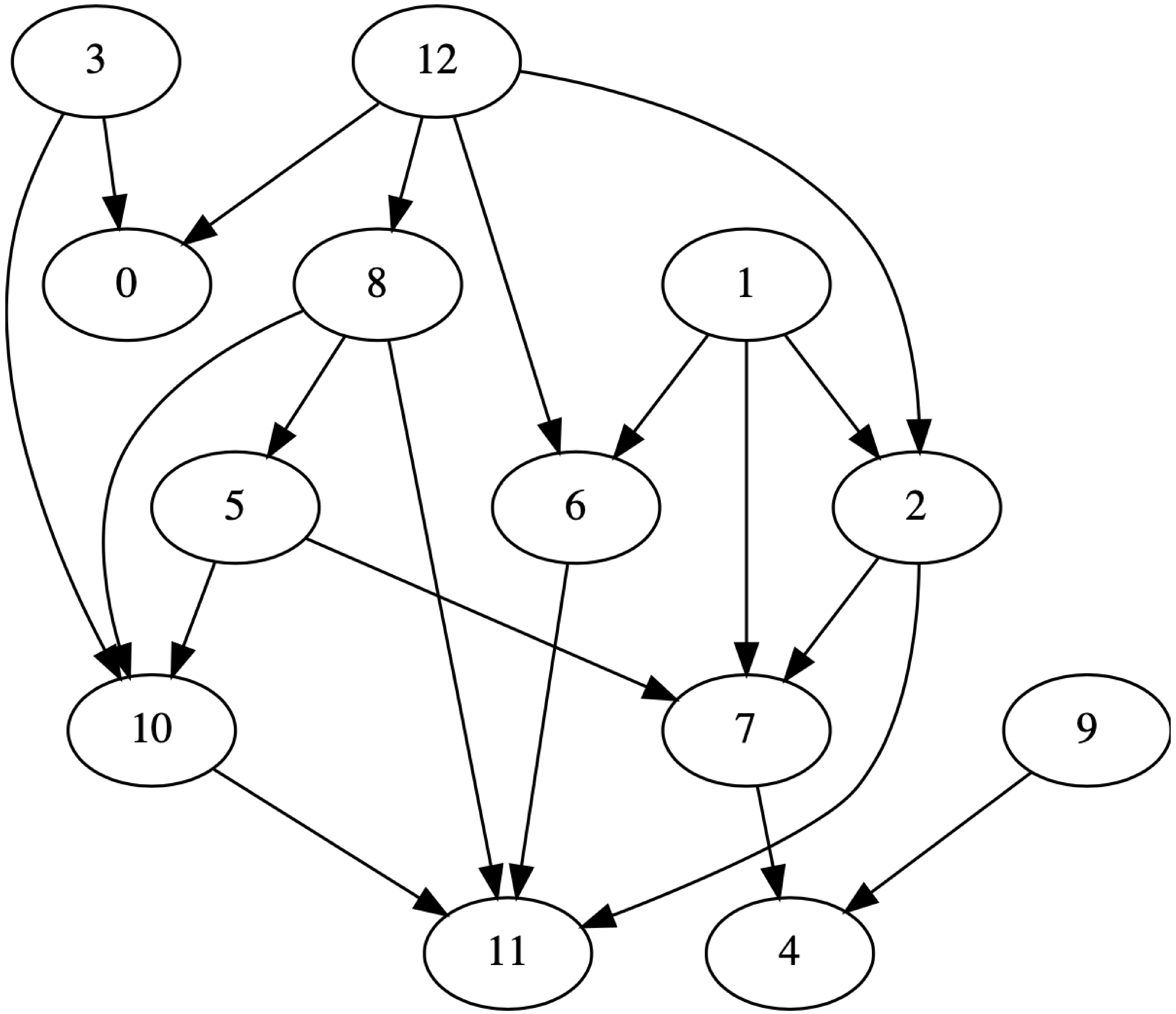
Solution Analysis

Team: JayceHaha
Member: Fuqiang Jiang

1. Get intuition: check the data
2. PC algorithm do not work
3. Estimate DAG by causal effect
4. Estimate DAG by TTPM method

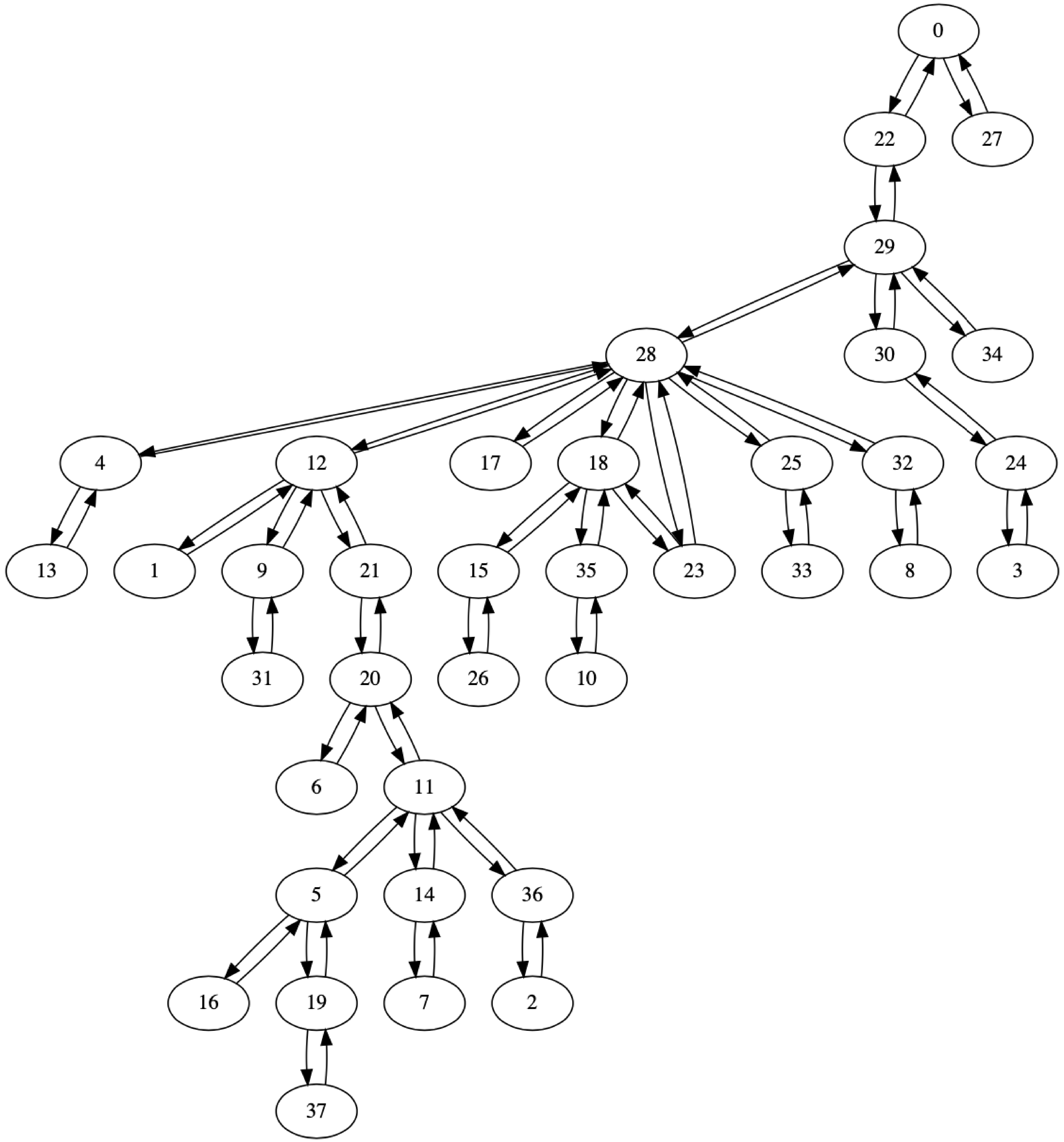
Get intuition

Ground true DAG of data-4:



Get intuition

Topology graph of data-4:



Get intuition

Statistics of data-4:

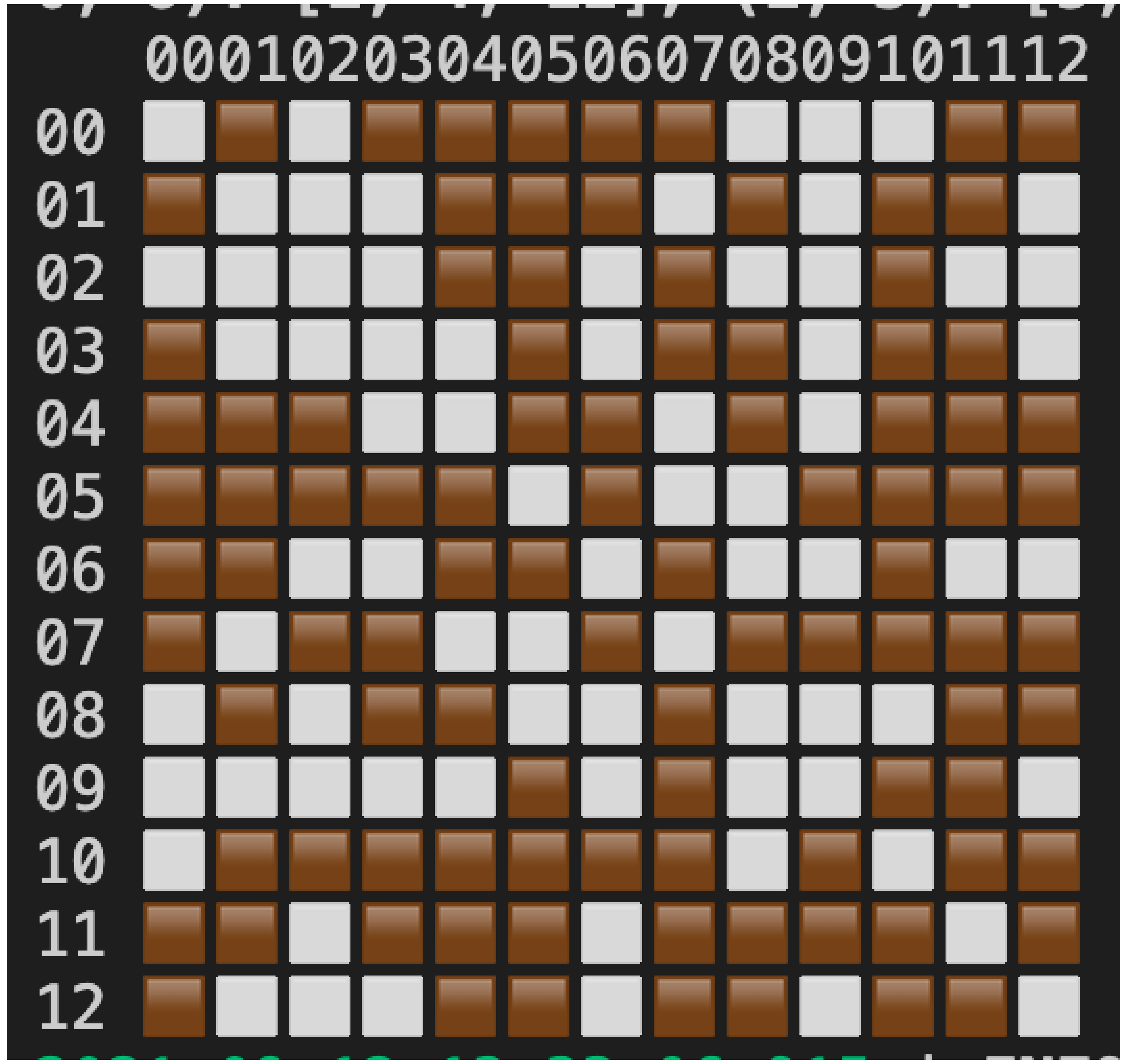
	device_id		duration	
	count	mean	count	mean
alarm_id				
0	3156	19.107414	3156	13.869455
1	1447	17.953697	1447	55.198341
2	4326	18.575127	4326	24.649561
3	1857	18.478729	1857	27.588045
4	20106	20.148911	20106	19.436188
5	5808	19.356921	5808	24.266357
6	4881	18.904118	4881	35.641262
7	12112	19.592305	12112	23.346929
8	3572	18.844345	3572	33.975644
9	1499	18.354236	1499	56.389593
10	9800	19.650714	9800	25.252347
11	22652	20.066572	22652	19.337277
12	2112	18.659091	2112	29.479640

Time interval between events

count	93327.000000
mean	6.479679
std	12.537947
min	0.000000
25%	1.000000
50%	2.000000
75%	6.000000
max	223.000000

PC-Algorithm do not work

* construct feature by windowing
* should be explored further



Estimate DAG by causal effect

Calculate the treatment effect of event i on event j

$$Y(x_j|do(x_i = 1)) - Y(x_j|do(x_i = 0) > E(Y(x_j))$$

Treatment effect: Summation inside a specified window

$$x_1, x_2, x_3, \ldots \boxed{x_j, \ldots x_k, \ldots x_j \ldots,} x_n \ldots$$

Estimate DAG by causal effect

Windowing

```
# apply sliding window, [N, num_events] --> [N, num_events, win_size]
data_view = np.lib.stride_tricks.sliding_window_view(data_onehot, delta_index, axis=0)
```

Sum over the window

```
# sum over the sliding window, [N, num_events, win_size-1] --> [N, num_events]
data_final = np.sum(data_view[:, :, 1:], axis=-1)
# clip values
data_final_clipped = np.clip(data_final, a_min=0, a_max=1)
```

Estimate DAG by causal effect

Iterate over all edge pairs

Apply same method to effect time

```
effect_matrix = np.zeros((num_events, num_events), dtype=np.float32)
avg_effect_times = np.zeros((num_events, num_events), dtype=np.float32)
std_effect_times = np.zeros((num_events, num_events), dtype=np.float32)

for i in range(num_events):
    for j in range(num_events):
        if i == j:
            continue
        cause_rows = cause_events[:, i] == 1
        selected_data = data_final_clipped[cause_rows]
        effect_matrix[i, j] = np.mean(selected_data[:, j])
```

Estimate DAG by causal effect

Treatment effect matrix

```
#### effect_matrix:
-----------------------------------------------------------------------------------------------
          0      1      2      3      4      5      6      7      8      9     10     11     12
00 [ 0.415  0.178  0.566  0.224  2.079  0.683  0.631  1.363  0.480  0.181  1.179  2.582  0.262]
-----------------------------------------------------------------------------------------------
          0      1      2      3      4      5      6      7      8      9     10     11     12
00 [ 0.038  0.007  0.036  0.009  0.145  0.052  0.062  0.067  0.051  0.008  0.093  0.097  0.014]
-----------------------------------------------------------------------------------------------
          0      1      2      3      4      5      6      7      8      9     10     11     12
00 [-0.000  0.025  0.059  0.047 -0.365  0.059 -0.015 -0.168  0.176  0.033  0.138  0.011  0.044]
01 [-0.127 -0.000  0.247  0.014 -0.044 -0.215  0.664  0.220 -0.118  0.001 -0.410 -0.023 -0.020]
02 [-0.009 -0.024 -0.000 -0.014 -0.022 -0.107  0.042  0.223  0.019 -0.007 -0.306  0.428 -0.013]
03 [ 0.239  0.055 -0.036 -0.000 -0.183 -0.101  0.013 -0.254 -0.063  0.066  0.561 -0.265  0.080]
04 [-0.096 -0.002 -0.079  0.012 -0.000 -0.123 -0.061  0.012 -0.109  0.009 -0.192 -0.221 -0.002]
05 [-0.080 -0.028 -0.138 -0.030 -0.071 -0.000 -0.206  0.545 -0.095 -0.023  0.352 -0.135 -0.043]
06 [-0.068  0.005  0.071 -0.003 -0.091 -0.130 -0.000 -0.079 -0.025 -0.004 -0.327  0.715  0.002]
07 [-0.117 -0.027 -0.128 -0.027  0.980 -0.093 -0.096 -0.000 -0.138 -0.025 -0.094 -0.170 -0.046]
08 [ 0.027 -0.026 -0.027 -0.044 -0.640  0.488 -0.138 -0.130 -0.000 -0.033  0.566  0.176 -0.062]
09 [-0.028  0.035 -0.039  0.060  1.098 -0.044 -0.035 -0.262 -0.013 -0.000 -0.208 -0.542  0.100]
10 [-0.005 -0.017 -0.114 -0.016 -0.177  0.181 -0.169  0.069 -0.093 -0.016 -0.000  0.300 -0.025]
11 [-0.054 -0.003 -0.052 -0.012  0.021  0.040 -0.051  0.068 -0.068 -0.008 -0.062 -0.000 -0.015]
12 [ 0.317  0.008  0.234  0.012 -0.505  0.046  0.051 -0.244  0.528  0.008 -0.019 -0.274 -0.000]
```

Estimate DAG by causal effect

Estimate DAG by deviation detect

mean value along rows:

standard value along rows:

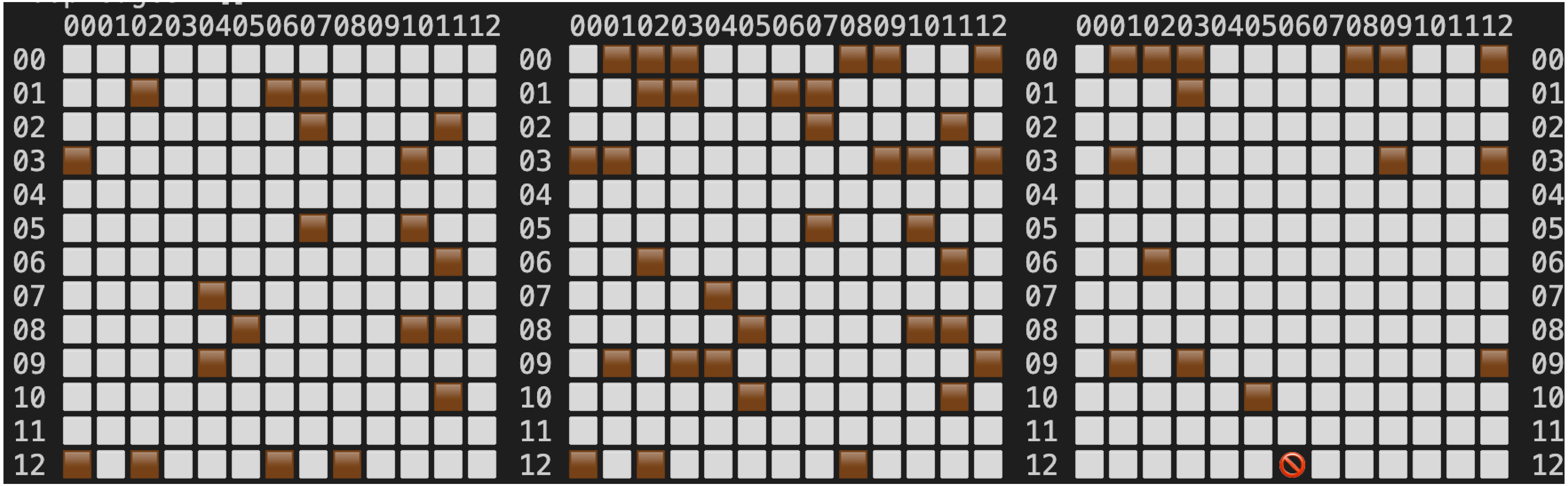
```
#### effect_matrix:
-----------------------------------------------------------------------------------------------
          0      1      2      3      4      5      6      7      8      9     10     11     12
00 [ 0.415  0.178  0.566  0.224  2.079  0.683  0.631  1.363  0.480  0.181  1.179  2.582  0.262]
-----------------------------------------------------------------------------------------------
          0      1      2      3      4      5      6      7      8      9     10     11     12
00 [ 0.038  0.007  0.036  0.009  0.145  0.052  0.062  0.067  0.051  0.008  0.093  0.097  0.014]
-----------------------------------------------------------------------------------------------
          0      1      2      3      4      5      6      7      8      9     10     11     12
00 [-0.000  0.025  0.059  0.047 -0.365  0.059 -0.015 -0.168  0.176  0.033  0.138  0.011  0.044]
01 [-0.127 -0.000  0.247  0.014 -0.044 -0.215  0.664  0.220 -0.118  0.001 -0.410 -0.023 -0.020]
02 [-0.009 -0.024 -0.000 -0.014 -0.022 -0.107  0.042  0.223  0.019 -0.007 -0.306  0.428 -0.013]
03 [ 0.239  0.055 -0.036 -0.000 -0.183 -0.101  0.013 -0.254 -0.063  0.066  0.561 -0.265  0.080]
04 [-0.096 -0.002 -0.079  0.012 -0.000 -0.123 -0.061  0.012 -0.109  0.009 -0.192 -0.221 -0.002]
05 [-0.080 -0.028 -0.138 -0.030 -0.071 -0.000 -0.206  0.545 -0.095 -0.023  0.352 -0.135 -0.043]
06 [-0.068  0.005  0.071 -0.003 -0.091 -0.130 -0.000 -0.079 -0.025 -0.004 -0.327  0.715  0.002]
07 [-0.117 -0.027 -0.128 -0.027  0.980 -0.093 -0.096 -0.000 -0.138 -0.025 -0.094 -0.170 -0.046]
08 [ 0.027 -0.026 -0.027 -0.044 -0.640  0.488 -0.138 -0.130 -0.000 -0.033  0.566  0.176 -0.062]
09 [-0.028  0.035 -0.039  0.060  1.098 -0.044 -0.035 -0.262 -0.013 -0.000 -0.208 -0.542  0.100]
10 [-0.005 -0.017 -0.114 -0.016 -0.177  0.181 -0.169  0.069 -0.093 -0.016 -0.000  0.300 -0.025]
11 [-0.054 -0.003 -0.052 -0.012  0.021  0.040 -0.051  0.068 -0.068 -0.008 -0.062 -0.000 -0.015]
12 [ 0.317  0.008  0.234  0.012 -0.505  0.046  0.051 -0.244  0.528  0.008 -0.019 -0.274 -0.000]
```

DAG[i, j] = 1 if effect_delta[i, j] > effect_std[j] * scale

Estimate DAG by causal effect

DAG estimated:

Ground True | Estimated | Estimated − Ground True



*Recall most edges
*Some false positive edges

Estimate DAG by causal effect

Estimate DAG by sorting

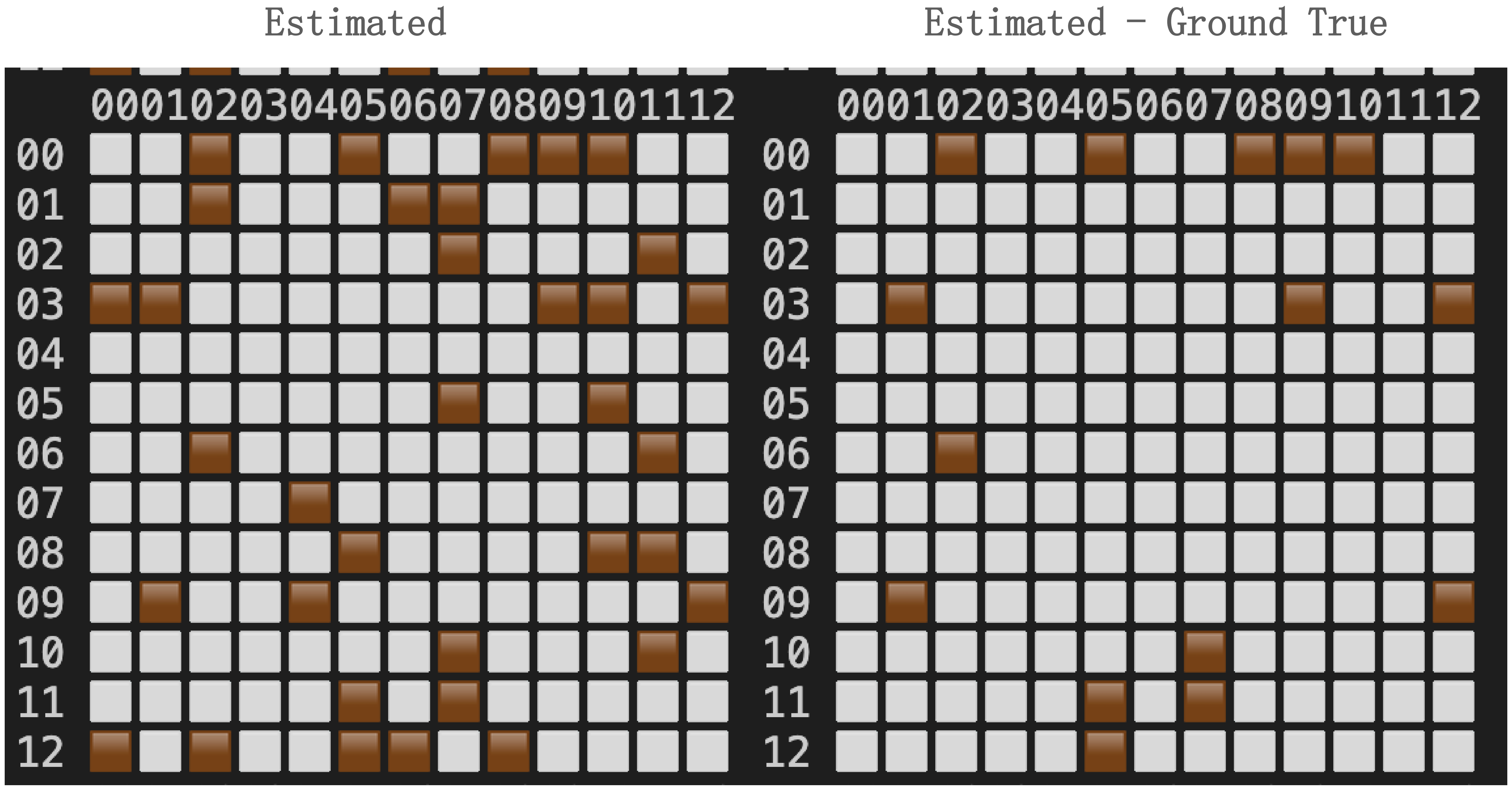
```
#### effect_matrix:
-------------------------------------------------------------------------------------------
         0      1      2      3      4      5      6      7      8      9     10     11     12
00 [ 0.415  0.178  0.566  0.224  2.079  0.683  0.631  1.363  0.480  0.181  1.179  2.582  0.262]
-------------------------------------------------------------------------------------------
         0      1      2      3      4      5      6      7      8      9     10     11     12
00 [ 0.038  0.007  0.036  0.009  0.145  0.052  0.062  0.067  0.051  0.008  0.093  0.097  0.014]
-------------------------------------------------------------------------------------------
         0      1      2      3      4      5      6      7      8      9     10     11     12
00 [-0.000  0.025  0.059  0.047 -0.365  0.059 -0.015 -0.168  0.176  0.033  0.138  0.011  0.044]
01 [-0.127 -0.000  0.247  0.014 -0.044 -0.215  0.664  0.220 -0.118  0.001 -0.410 -0.023 -0.020]
02 [-0.009 -0.024 -0.000 -0.014 -0.022 -0.107  0.042  0.223  0.019 -0.007 -0.306  0.428 -0.013]
03 [ 0.239  0.055 -0.036 -0.000 -0.183 -0.101  0.013 -0.254 -0.063  0.066  0.561 -0.265  0.080]
04 [-0.096 -0.002 -0.079  0.012 -0.000 -0.123 -0.061  0.012 -0.109  0.009 -0.192 -0.221 -0.002]
05 [-0.080 -0.028 -0.138 -0.030 -0.071 -0.000 -0.206  0.545 -0.095 -0.023  0.352 -0.135 -0.043]
06 [-0.068  0.005  0.071 -0.003 -0.091 -0.130 -0.000 -0.079 -0.025 -0.004 -0.327  0.715  0.002]
07 [-0.117 -0.027 -0.128 -0.027  0.980 -0.093 -0.096 -0.000 -0.138 -0.025 -0.094 -0.170 -0.046]
08 [ 0.027 -0.026 -0.027 -0.044 -0.640  0.488 -0.138 -0.130 -0.000 -0.033  0.566  0.176 -0.062]
09 [-0.028  0.035 -0.039  0.060  1.098 -0.044 -0.035 -0.262 -0.013 -0.000 -0.208 -0.542  0.100]
10 [-0.005 -0.017 -0.114 -0.016 -0.177  0.181 -0.169  0.069 -0.093 -0.016 -0.000  0.300 -0.025]
11 [-0.054 -0.003 -0.052 -0.012  0.021  0.040 -0.051  0.068 -0.068 -0.008 -0.062 -0.000 -0.015]
12 [ 0.317  0.008  0.234  0.012 -0.505  0.046  0.051 -0.244  0.528  0.008 -0.019 -0.274 -0.000]
```

Simply sort all effect value and select the top k edges

Remove bi-directional edges by removing the less significant one

Estimate DAG by causal effect

DAG estimated:

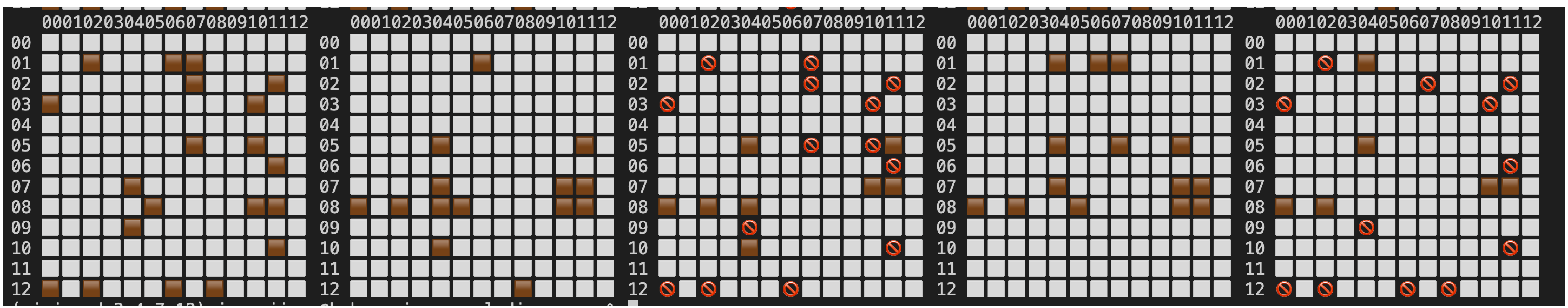


*Recall all edges
*Some false positive edges (less than checking std value)

Estimate DAG by causal effect

Apply same process to effect time, Got:

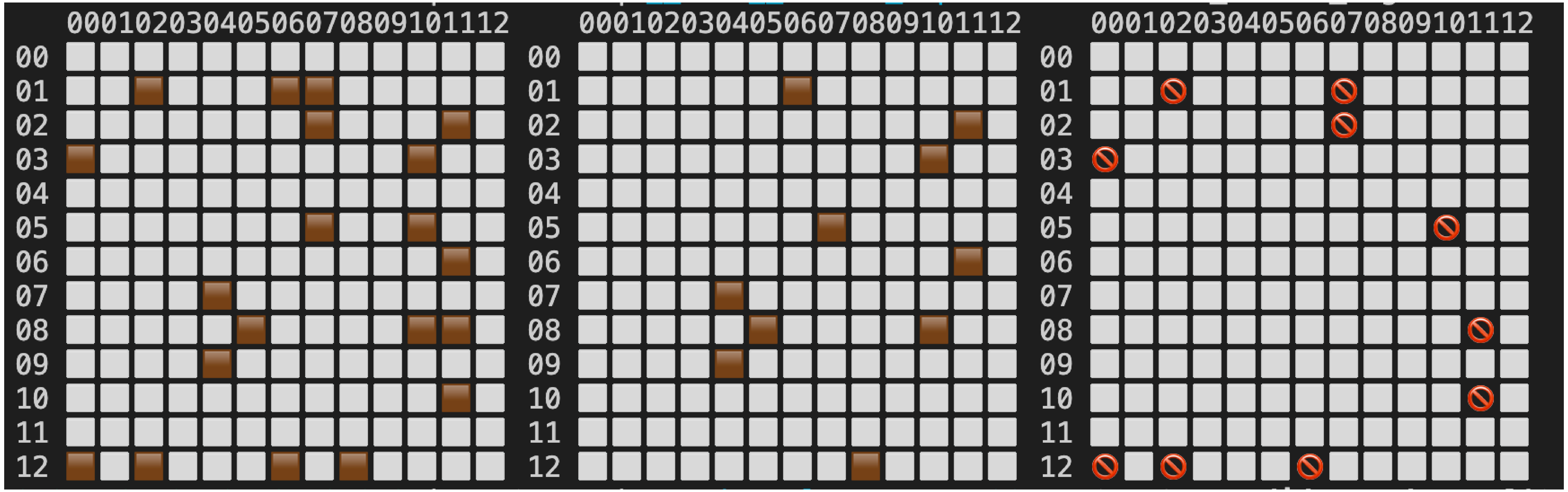
Ground True | Estimated by mean effect time | Delta | Estimated by std effect time | Delta



Estimate DAG by causal effect

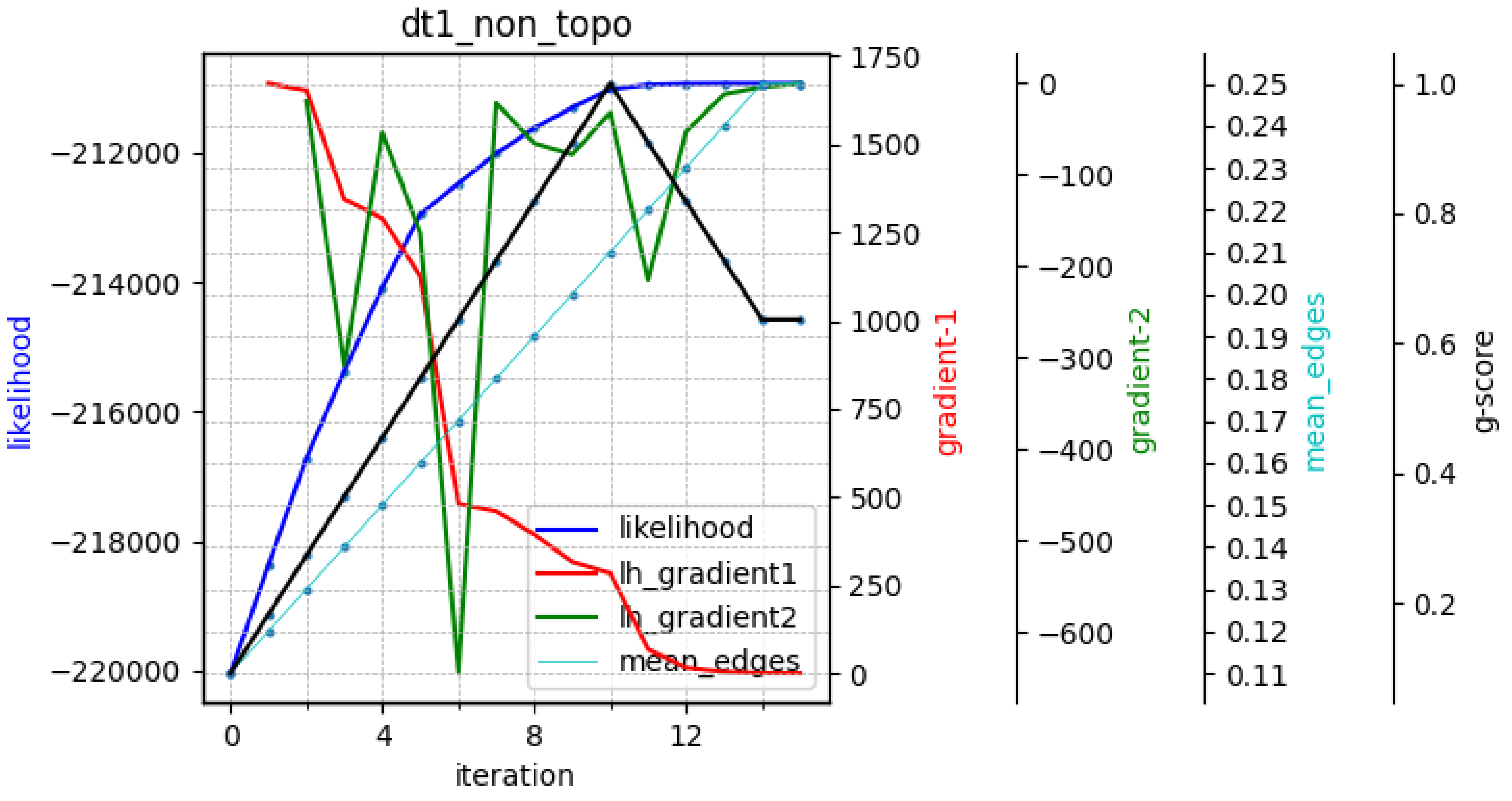
What these DAG can do?

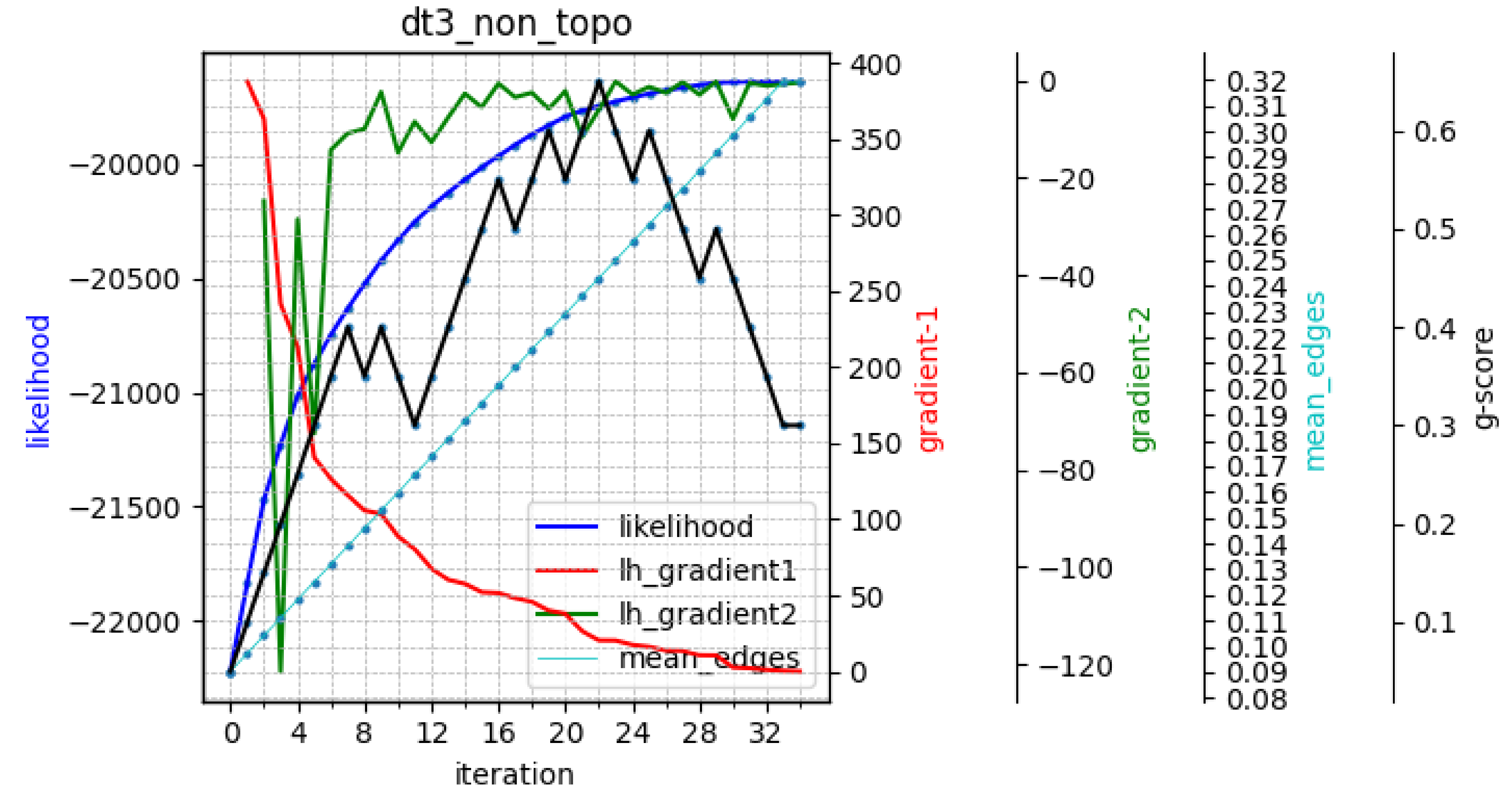
* Give some intuition
* Used as initial edge_mat for TTPM
* Used as candidate edges for TTPM



TTPM method

Run the baseline method





TTPM method

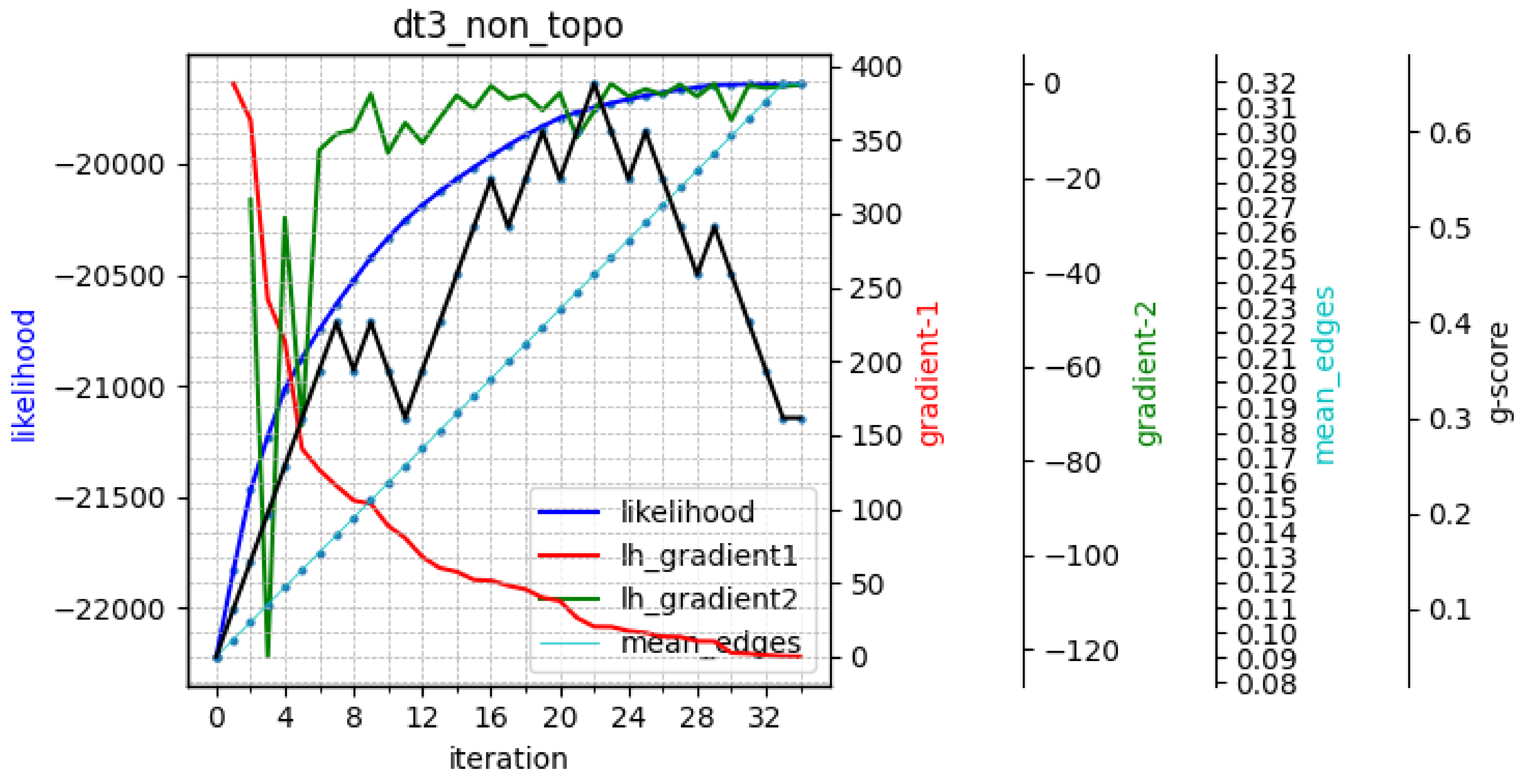
Hyper-parameters

* **max_hop**
  * For data without topo_mat: 0
  * For data with topo_mat: 2 is better 1 (that's what the paper tell us too)
* **max_iterations**
  * the larger the better, util the likelihood converges
  * > 40 for phase1, > 60 for phase 2
* **delta**
  * roughly around 0.01 ~ 0.04 (default 0.10)
* **epsilon**
  * 2.0 ~ 4.0 (default 1.0)
* **penalty method:**
  * "BIC" is better than "AIC"

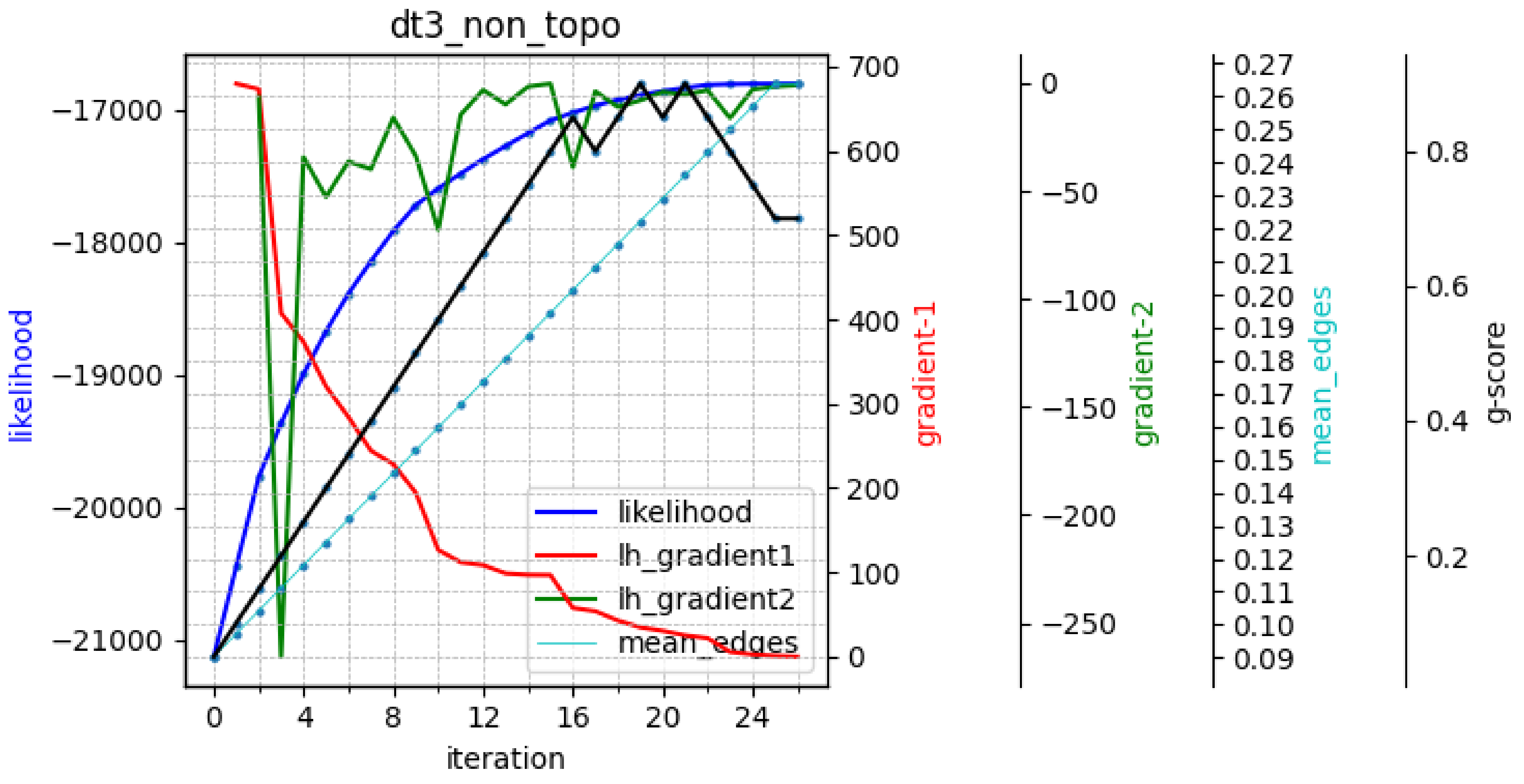
TTPM method

Hyper-parameters: delta

delta: 0.10
(default)



delta: 0.02

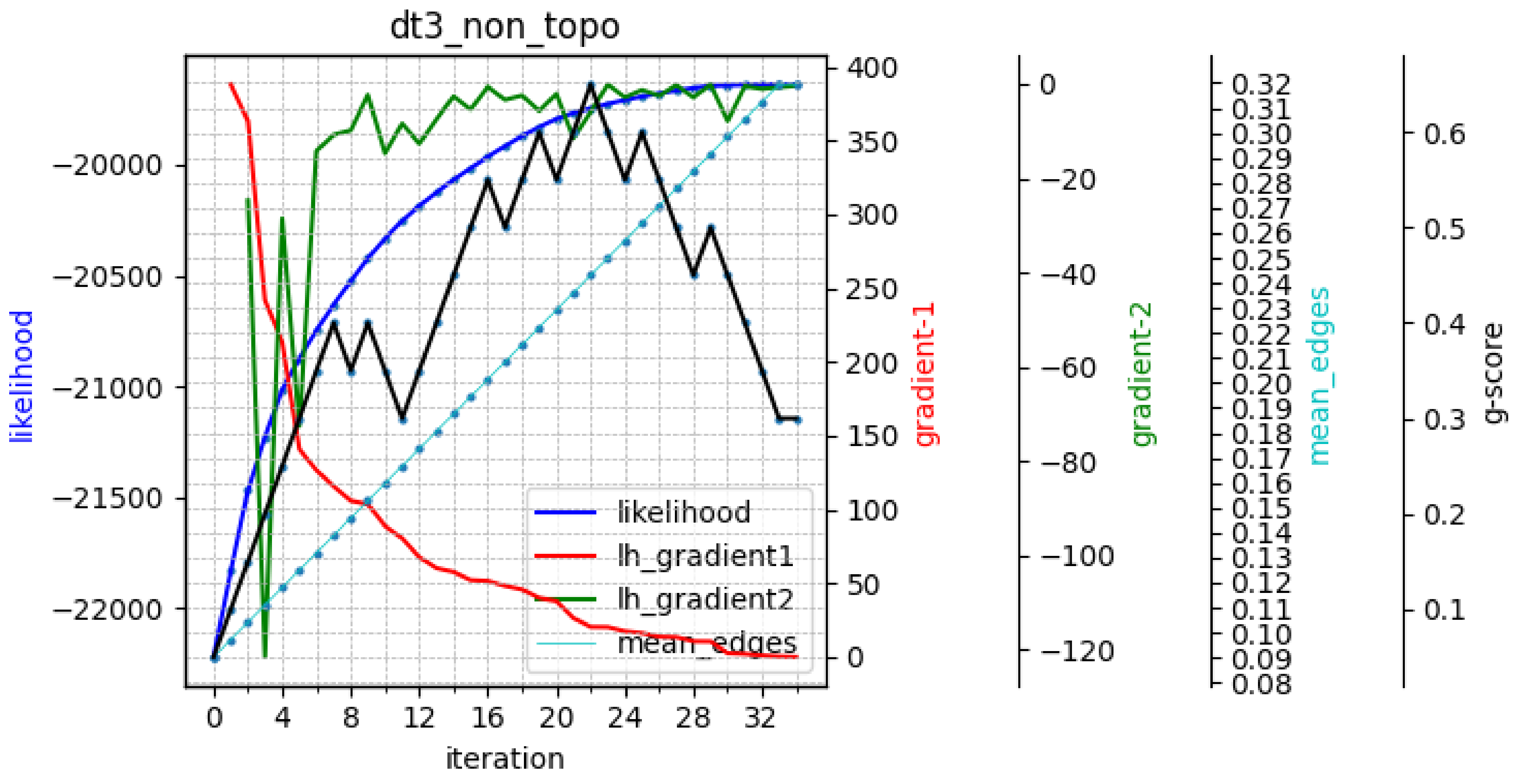


*higher likelihood
*higher g-score

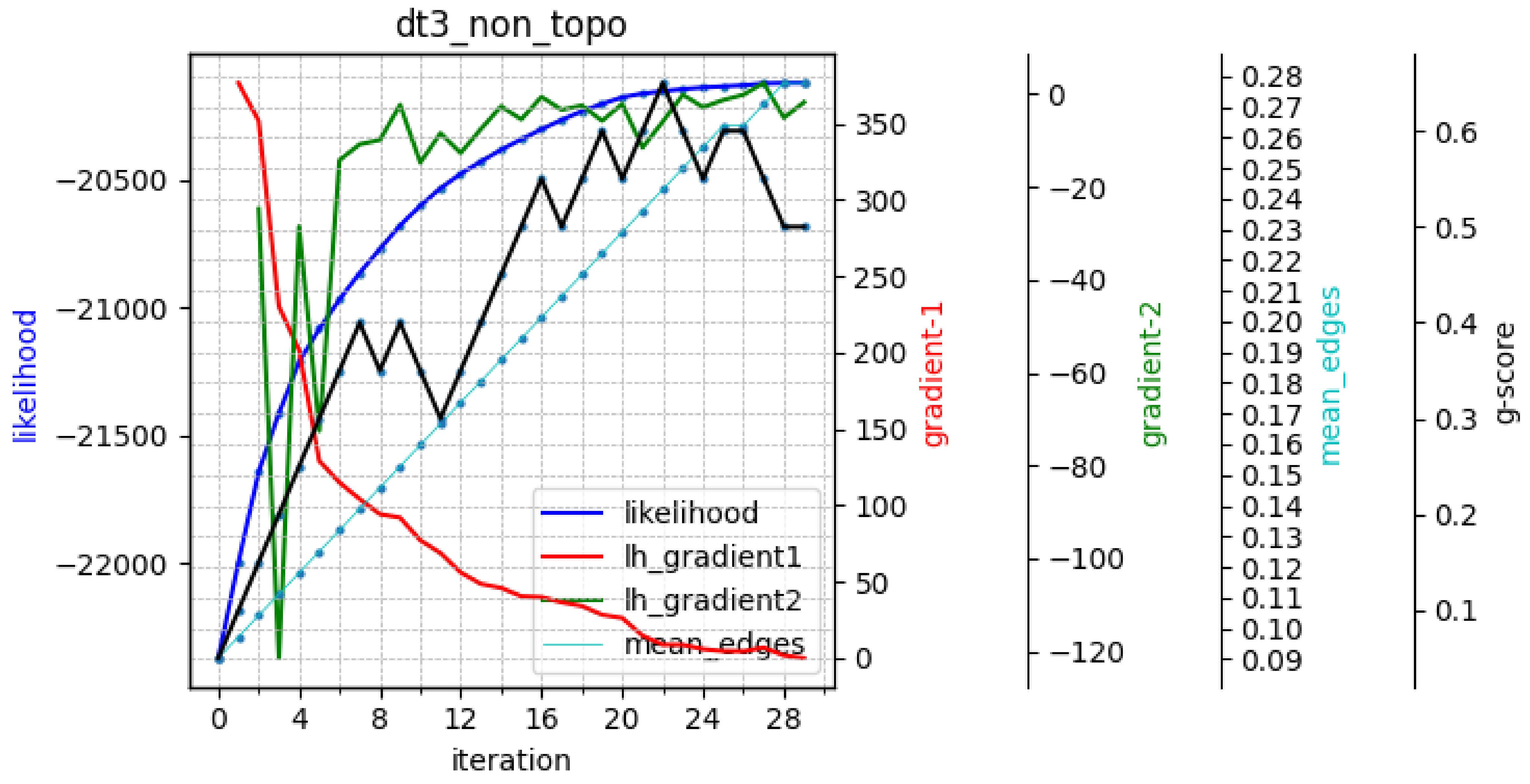
TTPM method

Hyper-parameters: epsilon

epsilon: 1.0
(default)



epsilon: 4.0

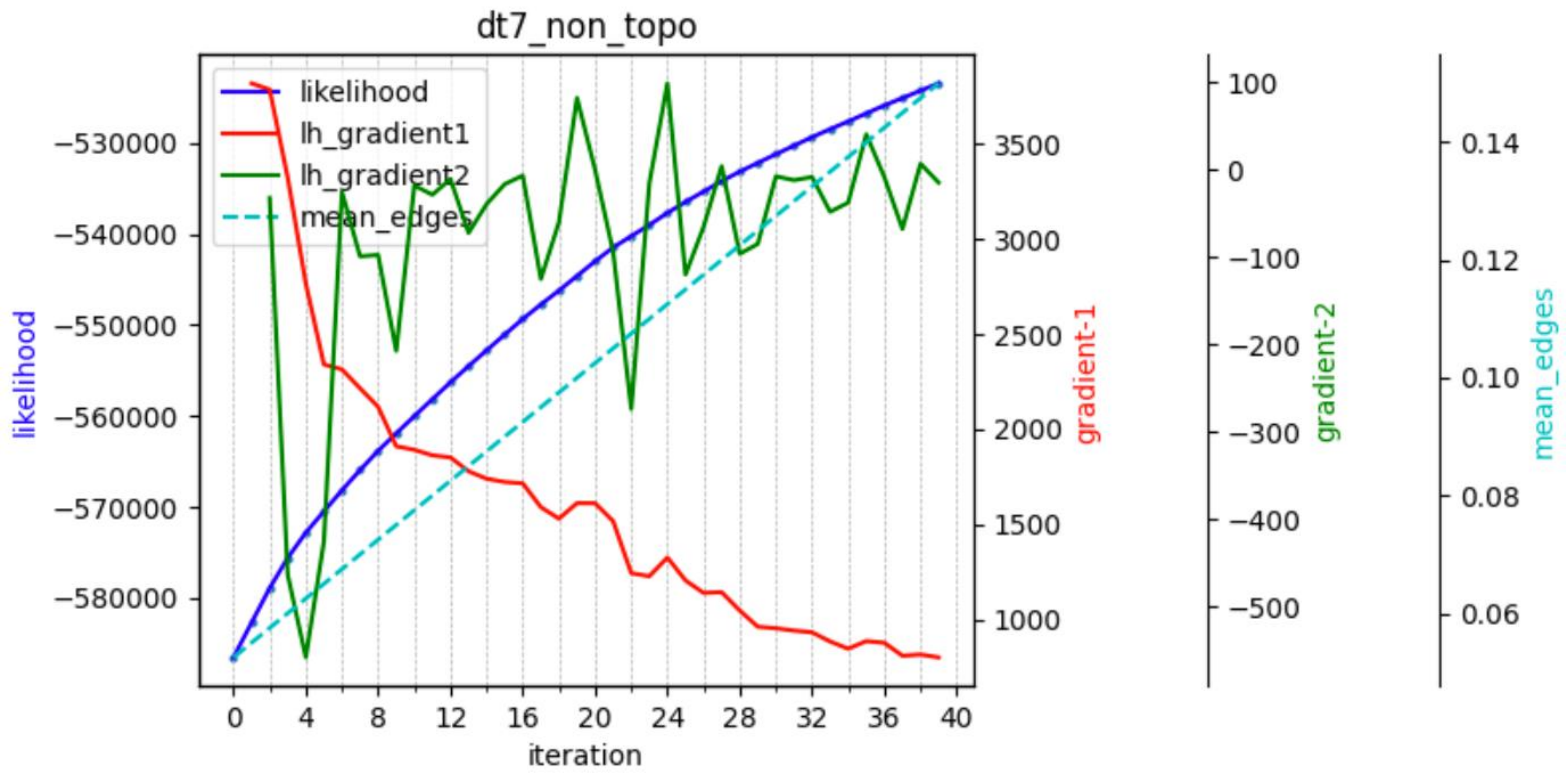


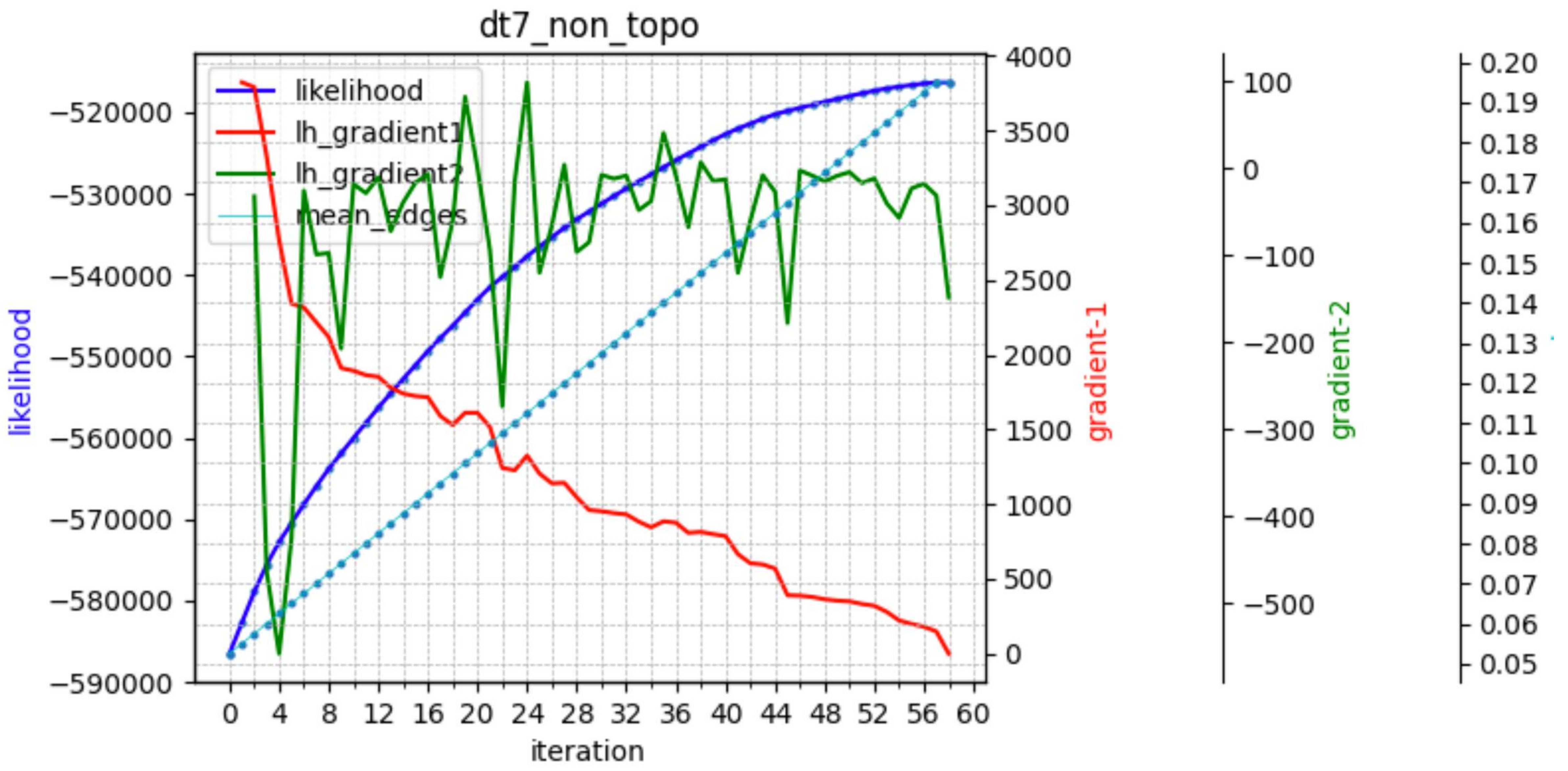
*Converge more earlier

TTPM method

Key details of phase 2

* max_iteration
  * much larger then phase 1
* delta
* epsilon

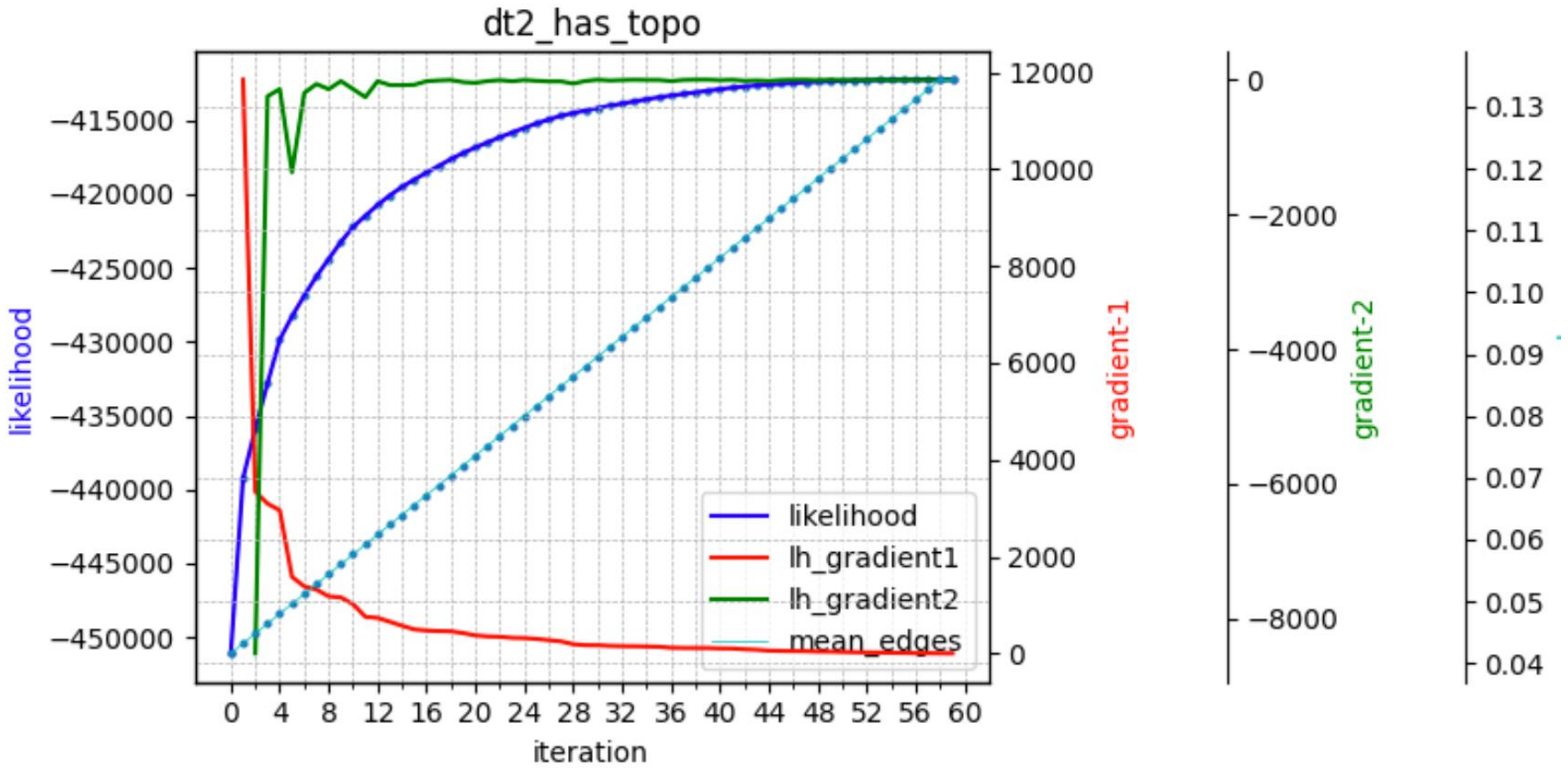




TTPM method

Phase 2: data-1, 2 are hard to learn

*max_iteration
  *much larger then phase 1
*likelihood changes too smoothly



TTPM method

TTPM is time consuming

$$\mathcal{O}\left(Km^3 E^{\frac{|\mathbf{V}|(|\mathbf{V}|+1)}{2}}\right).$$

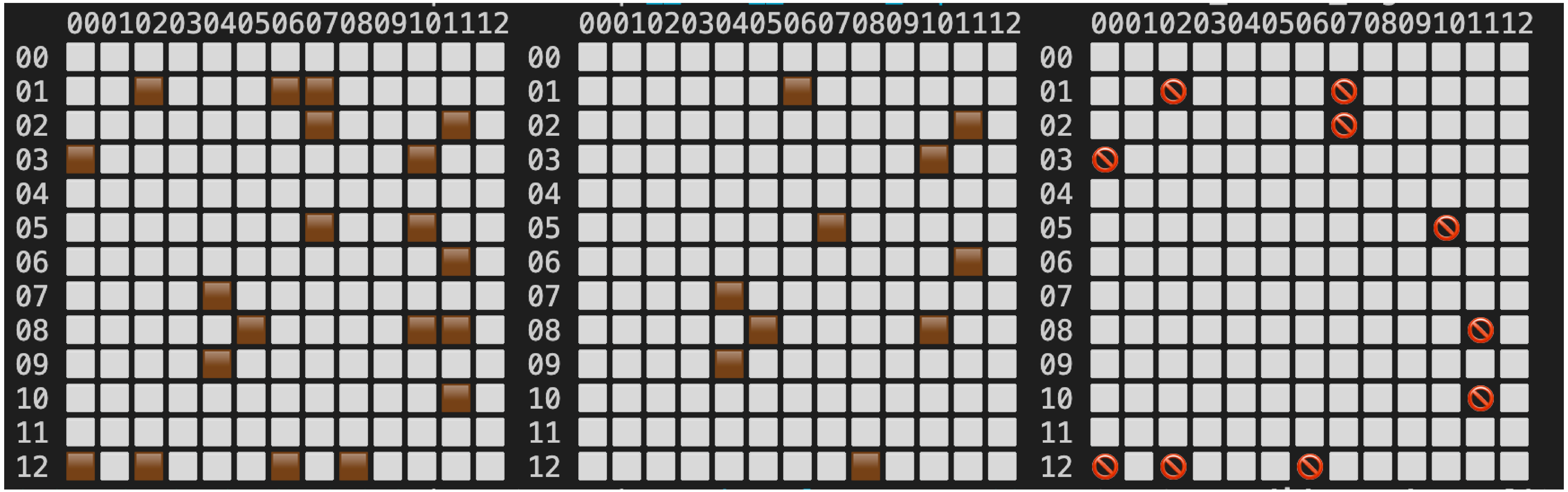
```python
def _one_step_change_iterator(self, edge_mat):

        return map(lambda e: self._one_step_change(edge_mat, e),
                   product(range(len(self._event_names)),
                           range(len(self._event_names))))
```

TTPM method

Improving speed

Use the DAG estimated by treatment effect as initial edge mat

Only keep edges with significant treatment effect (Use the top 6% edges)



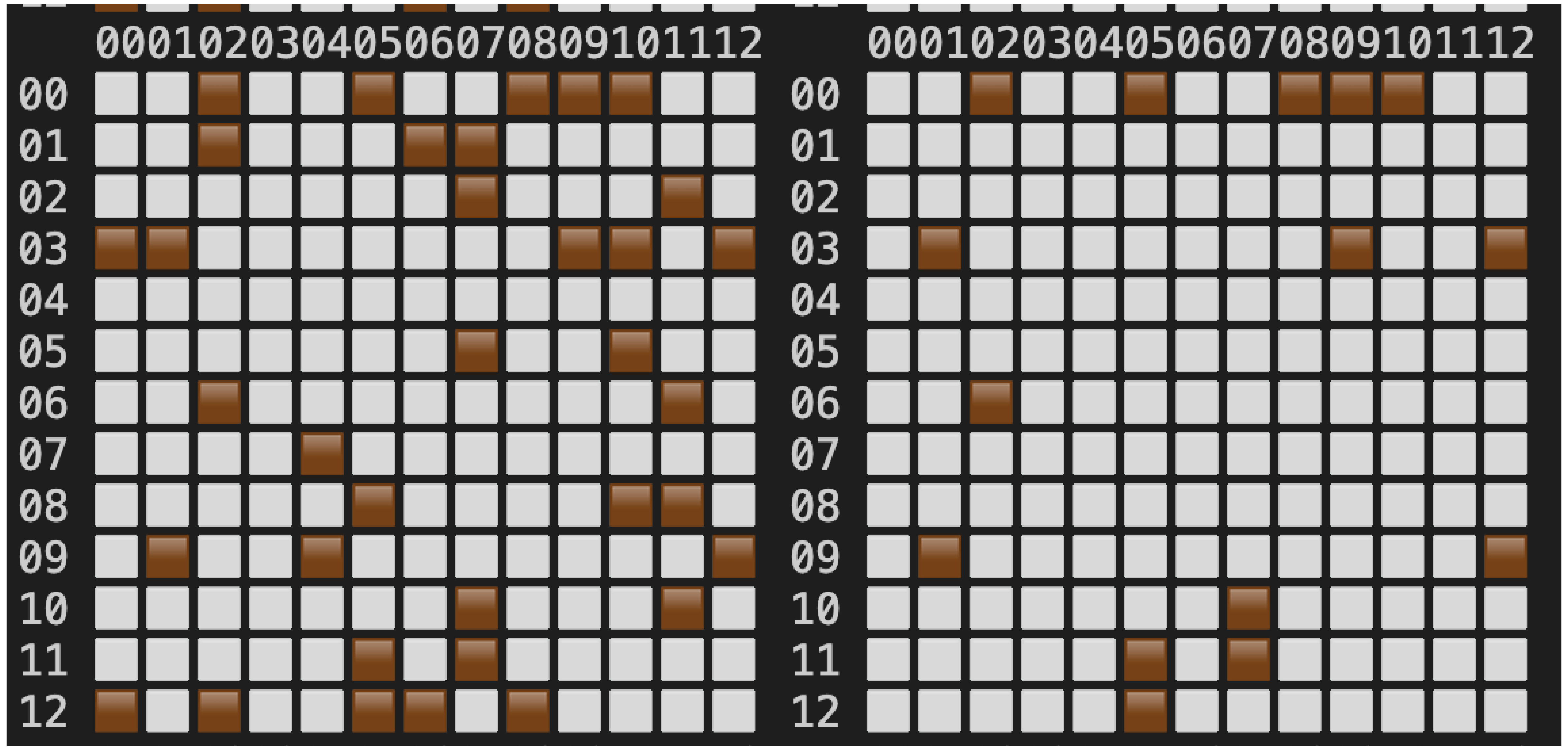
TTPM method

Improving speed

Use the DAG estimated by treatment effect as candidate edges

Keep most edges with significant treatment effect (Use the top 25% edges)

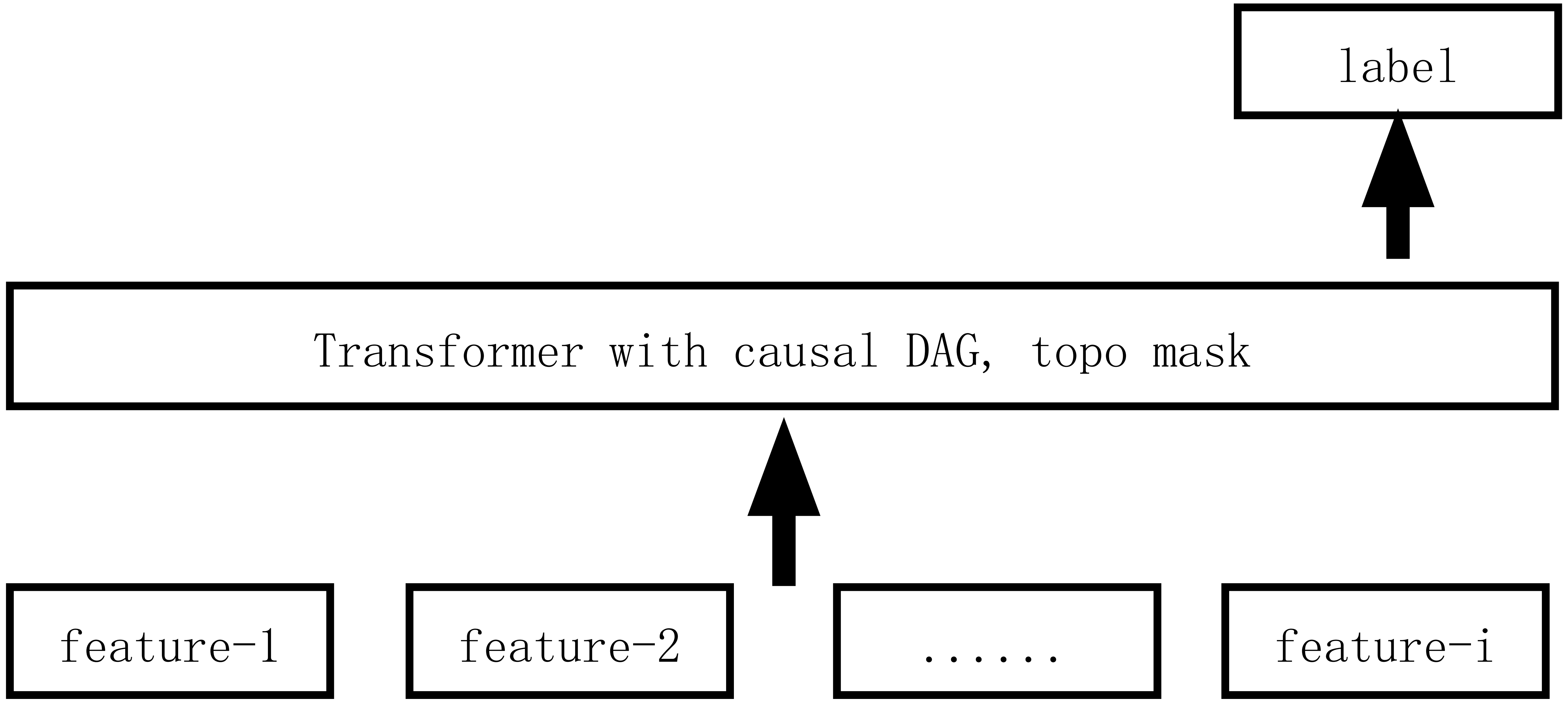
Recover to search around all edges when likelihood converges



10x speed up in data [1, 2, 3, 4] in phase 1, not tested for phase2 yet

Failed effort

Try to estimate DAG by gradient but failed



Q & A

Thanks!