*Q1: All your output statistics match my statistics.*

A1: The output statistics of my own implementation of ListMiner, both for the large dataset and small dataset, are included in the submission folder, by following the format of the given output statistics. In sum, the results of my implementation are very close to the given benchmark of PSEMiner (Table 1):

|  |  |  |
| --- | --- | --- |
|  | ListMiner | PSEMiner |
| Total # of PPSEs Mined (Large) | 10437 | 9784 |
| Total # of PPSEs Mined (Small) | 6167 | 6167 |

Table 1

The accuracy is (10437-9784)/9784\*100%=93.33% for the large dataset, and 100% (Yes!!) for the small dataset.

*Q2: Your program has the "subsumed" function.*

A2: The implementation of ListMiner does have the “subsumed” function; refer to source code for detail.

*Q3: You conduct experiment to compare PSEMiner and your implementation, time performance and space* *performance*

A3: The time performance and space performance of both ListMiner and PSEMiner are summarized in Table 2 as follows (the values are obtained by averaging results of 10 experiments for each condition):

|  |  |  |
| --- | --- | --- |
|  | ListMiner | PSEMiner |
| Time (Seconds) – Large Data | 6.9534 | 18.5862 |
| Peak Space (MB) – Large Data | 266 | 43 |
| Time (Seconds) – Small Data | 3.8450 | 4.6792 |
| Peak Space (MB) – Small Data | 123 | 11 |

Table 2

*Q4: Reasoning, you have careful analysis on your experiment results*

In both cases, the ListMiner outperforms PSEMiner in terms of time consumption. This empirical result is consistent with the theoretical analysis in the IPL paper, which is to be expected. However, the ListMiner does cost much more memory. Potential source contributing to the consumption of memory possibly include:

1). The implementation of hash\_map: According to C++ documentation, the hash\_map structure is designed to hold large numbers of objects in a way that makes adds, deletes, lookups, and order less traverses efficient. It's not meant to be memory-efficient for small data structures. To avoid the penalties associated with resizing, it allocates many hash chain heads when it's first created;

2). The application of matrix is costly in terms of memory consumption: the 2D matrix in the program is simulated by vector<vector<>>, which is not the most efficient way – the boost library has an implementation of matrix, which could potentially improve efficiency.