ParallelECLAT-ad

September 5, 2022

1 Advanced Tutorial on Implementing parallelECLAT Algorithm

In this tutorial, we will discuss two approaches to find frequent patterns in big data using parallelECLAT algorithm.

1. Advanced approach: Here, we generalize the basic approach by presenting the steps to discover frequent patterns using multiple minimum support values.

In this tutorial, we explain how the parallelECLAT (parallelECLAT) algorithm can be implemented by varying the minimum support values

Step 1: Import the parallelECLAT algorithm and pandas data frame

[1]:
```
from PAMI.frequentPattern.pyspark import parallelECLAT  as alg
import pandas as pd
```

Step 2: Specify the following input parameters

[2]:
```
inputFile = 'temporal_T10I4D100K.csv'
seperator='\t'
mumberWorkersCount=2
minimumSupportCountList = [100, 150, 200, 250, 300]
#minimumSupport can also specified between 0 to 1. E.g., minSupList = [0.005, 0.
 ↪006, 0.007, 0.008, 0.009]

result = pd.DataFrame(columns=['algorithm', 'minSup', 'patterns', 'runtime',␣
 ↪'memory'])
#initialize a data frame to store the results of parallelECLAT algorithm
```

Step 3: Execute the parallelECLAT algorithm using a for loop

[3]:
```
algorithm = 'parallelECLAT'  #specify the algorithm name
for minSupCount in minimumSupportCountList:
    obj = alg.parallelECLAT(iFile=inputFile,␣
 ↪minSup=minSupCount,numWorkers=mumberWorkersCount, sep=seperator)
    obj.startMine()
    #store the results in the data frame
```

```
    result.loc[result.shape[0]] = [algorithm, minSupCount, len(obj.
↪getPatterns()), obj.getRuntime(), obj.getMemoryRSS()]
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
22/08/24 13:58:00 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
22/08/24 13:58:01 WARN Utils: Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.


100 <class 'int'>
<class 'int'> 100


Frequent patterns were generated successfully using Parallel ECLAT algorithm
22/08/24 13:58:46 WARN Utils: Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.


150 <class 'int'>
<class 'int'> 150


Frequent patterns were generated successfully using Parallel ECLAT algorithm
22/08/24 13:59:20 WARN Utils: Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.


200 <class 'int'>
<class 'int'> 200


Frequent patterns were generated successfully using Parallel ECLAT algorithm
22/08/24 13:59:48 WARN Utils: Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.


250 <class 'int'>
<class 'int'> 250


Frequent patterns were generated successfully using Parallel ECLAT algorithm
22/08/24 14:00:11 WARN Utils: Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.
```

```
300 <class 'int'>
<class 'int'> 300
```

```
Frequent patterns were generated successfully using Parallel ECLAT algorithm
```

[4]:
```
print(result)
```

```
       algorithm  minSup  patterns    runtime     memory
0  parallelECLAT     100     27532  46.434464  223105024
1  parallelECLAT     150     19126  33.166129  224575488
2  parallelECLAT     200     13255  26.772719  224276480
3  parallelECLAT     250      7703  22.108692  223600640
4  parallelECLAT     300      4552  15.972808  223133696
```

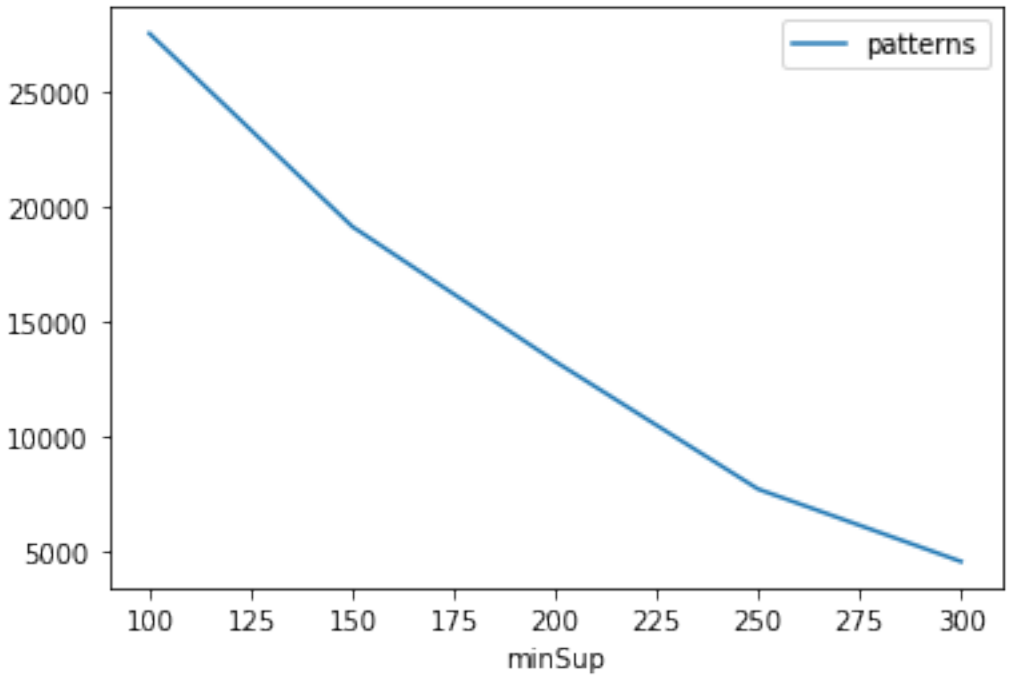
Step 5: Visualizing the results

Step 5.1 Importing the plot library

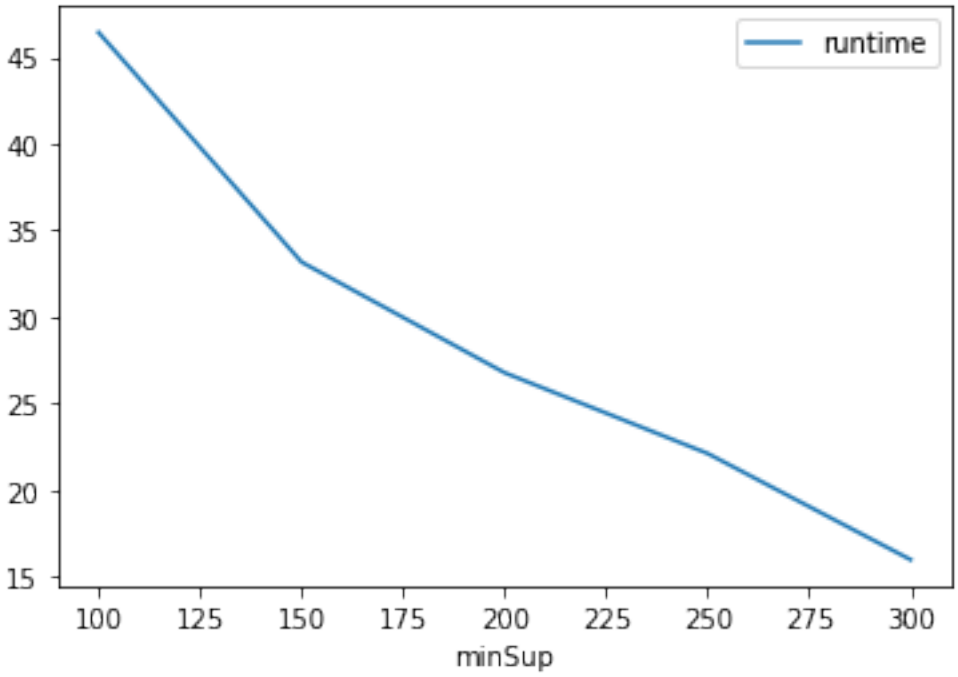
[5]:
```
from PAMI.extras.graph import plotLineGraphsFromDataFrame as plt
```

Step 5.2. Plotting the number of patterns

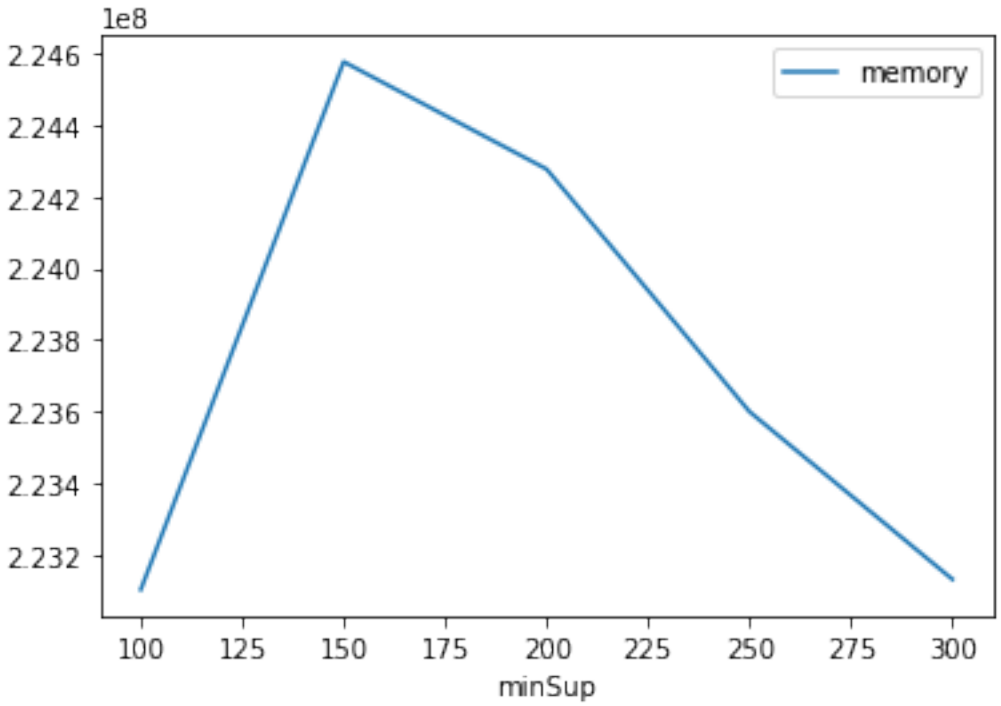
[6]:
```
ab = plt.plotGraphsFromDataFrame(result)
ab.plotGraphsFromDataFrame() #drawPlots()
```



Graph for No Of Patterns is successfully generated!



Graph for Runtime taken is successfully generated!




```
Graph for memory consumption is successfully generated!
```

1.0.1 Step 6: Saving the results as latex files

[7]:
```
from PAMI.extras.graph import generateLatexFileFromDataFrame as gdf
gdf.generateLatexCode(result)
```

```
Latex files generated successfully
```