

SPADE: An Efficient Algorithm for Mining Frequent Sequence, a C implementation.

Kevin Mato K5529

An implementation in C code. The description of the algorithm will be omitted. Refer to the original paper for further information, found in the references.

Assumptions made:

It's assumed that the libraries from libc are correctly implemented and correctly working and that the compiler used for the testing phase (MinGw GCC/G++) is working and linking correctly. The code is written in a way that is supposed to permit efficient compilation and efficient exploitation of the modern processors optimization techniques, for example "loop unrolling". The operative system Windows has memory limits way smaller than those in the Linux/Unix environment.

Description of input and output:

The input is passed through absolute path to the program as a start up first argument, and the second parameter for the executable will be the path for the output file.

In case of inexistence of the input file the program will be closed with an signal of system exit, meanwhile in case of absence of a file for the output generation, a new file named in the way we chose and in the correct location indexed in memory will be created. The output file and the input file will be closed and the program exit, so their modification will be saved only in that moment. It's assumed that both the files will be *.txt files correctly formed.

The output file based on the output options chosen can present a different layout based on the parameters given as input in the program startup. Please refer to the user guide for a more in-depth explanation on how to interpret this part.

The input file has strict rules relatively to its preprocessing.

An example of input:

"{C,D}",1,10-> in this line we can see that three fields can be extracted.

The first field is delimited by the characters [{ } "]. The use of double quotes is optional. The first field in any case delimited by the {,}. This field will contain the items for a transaction in a sequence. The second field "{C,D}",1,10 is the "customerID" or sequence ID and the third field "{C,D}",1,10 is the transaction id or event/element ID. These two fields need to be in this order for a correct fetching of the information. They need to be separated by commas respectively from field 1 and from field 2. The lines in dataset will present always in order the "basket", than the sid and the eid. The lines in the dataset are required to be in order od sid, and then eid. The first field doesn't need to present the element in lexicographic order. Spaces are forbidden inside the first field.

For example:

"{C,D}",1,10

"{C,D}",2,10

"{D,A}",2,15

The input file doesn't have any size limit besides those imposed by the operative system where the program will be run.

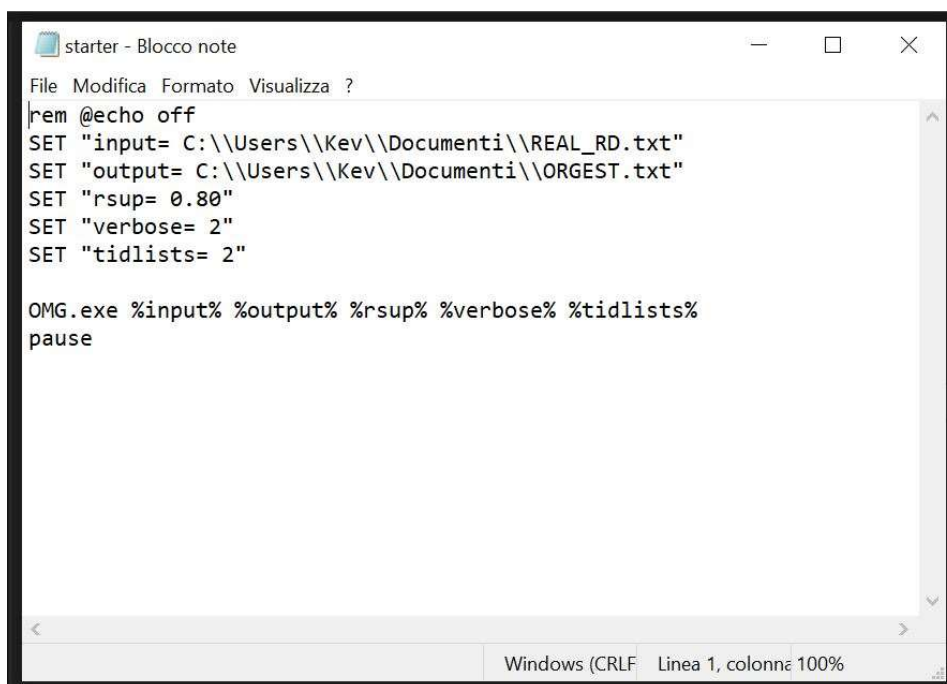
A suggestion for a fast creation of an input file: in R create a data frame with the items you want in the transaction, then the sids in order and the last field always in order but related to the sid. Save the table in a data file and then use the function "read_baskets" from "aRulesSequences" library.

This will create a data-structure in R with the transactions we need. As a last step save the data-structure as data frame in a txt file. So, the hardest part remains selecting the features we want from the original dataset.

User Guide:



To modify the parameters for the execution of the program right click on the batch file starter and open with an editor.



```
File Modifica Formato Visualizza ?
rem @echo off
SET "input= C:\\Users\\Kev\\Documenti\\REAL_RD.txt"
SET "output= C:\\Users\\Kev\\Documenti\\ORGEST.txt"
SET "rsup= 0.80"
SET "verbose= 2"
SET "tidlists= 2"

OMG.exe %input% %output% %rsup% %verbose% %tidlists%
pause

Windows (CRLF) Linea 1, colonna 100%
```

Modify the batch file variables in the file after the '=' symbol, to change the path of the input, or of the output. The rsup variable indicates the minimum relative support for a generated sequence to be valid. Verbose can have three values 0, 1, 2. Increasing the value of this parameter will make the output more verbose. With 0 no info about the supports is shown. With one only the relative support. With 2 also the absolute support. Tidlists can have only 0, 1, 2. For 0 no tidlists info will be shown. For 1 only the sids. For 2 the whole Vertical - idlist will be shown.

-Example for the Zaki dataset, with $rsup=0.5$, $verbose=2$, $tidlist=2$:

8 <{A,B}> sup: 3 rsup: 0.750

V-idlist

sid: 1, eid: 15

sid: 1, eid: 20

sid: 2, eid: 15

sid: 3, eid: 10

-with $rsup=0.5$, $verbose=1$, $tidlist=1$:

8 <{A,B}> rsup: 0.750 sid: { 1 2 3 }

Description of the execution path

I followed meticulously the description of the algorithm from the original paper [1] apart from only one step.

```
SPADE ( $min\_sup, \mathcal{D}$ ):  
   $\mathcal{F}_1 = \{ \text{frequent items or 1-sequences} \};$   
   $\mathcal{F}_2 = \{ \text{frequent 2-sequences} \};$   
   $\mathcal{E} = \{ \text{equivalence classes } [X]_{\theta_1} \};$   
  for all  $[X] \in \mathcal{E}$  do Enumerate-Frequent-Seq( $[X]$ );
```

The SPADE algorithm.

The paper [1] divides the execution of the algorithm in three phases, so did I. The first part is the generation of the valid sequences of length and size one. In one scan of the data I firstly create the first level of the results tree, composed by all the items in the dataset with their vertical-idlists. It's an iterative process where every time I encounter a new line, I check if the element in the line is already present, if yes I add a new line to its v-idlist otherwise I allocate a new element.

The relative support of each item is calculated and if it is greater or equal to the minimum relative support we chose in the beginning, the 1-length sequences are kept otherwise these non-valid sequences are pruned from the tree and erased from memory together with their tables (note that perform a \geq of the supports against the min sup, differently than the Kryszkiewicz's approach where it's used only $>$).

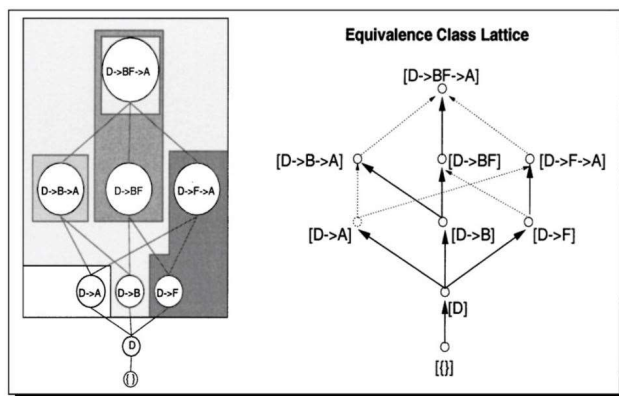
Once the valid items are found they are still represented by their original item name, so to decouple the name of each item, it is then represented by the string of the hexadecimal representation of the position of the item in the array of sons of the root. This representation isn't the fastest but is the safest in terms of memory leakage (I tried also a pure numeric representation but with scarce success in terms of memory management). The decoupling is done because the next results will be based on prefix extractions and on comparisons on the sequences representation. It's always better to compare a string logarithmically-growing length than an arbitrary ones. This representation will give us the

possibility to give back interpretable results in the output phase with constant cost, since the hex representation will be replaced with the item name just by picking the right item in the sons array of root and then checking their names.

The first step is completed and the root of the tree is returned.

In the second phase there's the generation of the sequences of length of 2. We can say that differently from the original paper [1] this is the only different part since we are creating candidate sequences since the beginning by combining the tidlists. The original paper in this second phase doesn't perform tidlists comparison, but it's shown a preprocessing phase. This preprocessing phase from my personal analysis (based on material which unluckily I can't retrieve anymore), is more expensive in terms of time.

The root of the tree is passed to the second routine of the program. With complexity $O(n^2)$ the items are combined to create candidates sequences of length 2. This complexity is necessary for a discovery of the new sequences. These candidates will be validated in terms of relative support, pruned if necessary and then written in the output. After the creation of the "2-sequences" the tables of the parents are deallocated. Note that each candidate is stored as son of the first item in the sequence created. This is done to create a recursive decomposition of class "atom" (ex. In the Zaki data set: "{D}") into smaller sub-classes (ex. $\{D\}, \{A\}, \{D\}, \{B\}, \{D\}, \{F\} \dots$).



Recursive decomposition of class $[D]$ into smaller sub-classes via θ_k .

The root is the returned again and we have an initial class decomposition.

In the third phase I create candidates of k-arbitrary length recursively. I implemented it in a "depth-first search" way since in the paper it was described as the most efficient in time and slim in terms of memory. With $O(m^2)$ with m number sequences at an arbitrary level of the tree for a given subclass.

In the validation of the sequences this time I'm not only checking if the relative support is valid but even if the number of transaction in a sequence don't exceed the number of sides.

Between one level and the other for a given subclass, only the tables of the sons or of the parents can exist. In the transition between a level and the other, the tables of the parents are always deallocated.

The creation of the sequences is based on a prefix extraction and comparison, if the prefixes for two i -length sequences ($2 \leq i$) match then we can proceed on the candidate creation and the creation of the relative table.

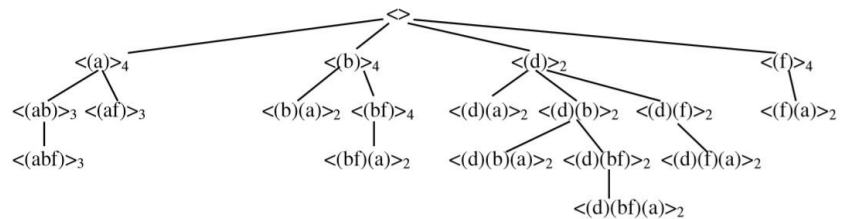
A note on the operations on the tables: the tables are created with linear cost to the appearance of an element in the dataset in order. Since the cost for adding a new line is constant. The operation of comparison between tables even if it can be optimized in different ways for many small changes it remains $O(n^2)$ a good way for doing comparisons would be doing a binary search in the tables $O(\log N)$ but as I'll show later the most important problems arise because of memory, and for external problems I didn't have the chance to implement it.

One more note on the preprocessing:--

Correctness of output

As a comparison I used the Zaki dataset with minimum relative support: 50%.

1	{D}	sup: 2	rsup: 0.500
2	{A}	sup: 4	rsup: 1.000
3	{B}	sup: 4	rsup: 1.000
4	{F}	sup: 4	rsup: 1.000
5	<{D},{A}>	sup: 2	rsup: 0.500
6	<{D},{B}>	sup: 2	rsup: 0.500
7	<{D},{F}>	sup: 2	rsup: 0.500
8	<{A,B}>	sup: 3	rsup: 0.750
9	<{A,F}>	sup: 3	rsup: 0.750
10	<{B},{A}>	sup: 2	rsup: 0.500
11	<{B,F}>	sup: 4	rsup: 1.000
12	<{F},{A}>	sup: 2	rsup: 0.500
13	<{D},{B},{A}>	sup: 2	rsup: 0.500
14	<{D},{B,F}>	sup: 2	rsup: 0.500
15	<{D},{F},{A}>	sup: 2	rsup: 0.500
16	<{D},{B,F},{A}>	sup: 2	rsup: 0.500
17	<{A,B,F}>	sup: 3	rsup: 0.750
18	<{B,F},{A}>	sup: 2	rsup: 0.500



Slide 11 of lecture notes from reference.

Output of the program

Qualitative and quantitative results from experiments

The first data set I tried to analyze is the Bach Coral Harmony data set. (link: <https://archive.ics.uci.edu/ml/datasets/Bach+Choral+Harmony>) . The objective of analyzing this dataset is finding patterns in the music of Bach which could lead to a general sentiment analysis of the compositions of the author.

After a preprocessing phase, I extracted the key, the accent in the movement and the bass note in that movement. The sid in this case will be the piece and the eid will be the movement in the piece.

Example of the data:

{F_M,3,F},1,1

{C_M,5,E},1,2

{C_M,2,E},1,3

{F_M,3,F},1,4

The total number of sides is 62, it means that the relative support has a granularity of 1.5. Maximum eid in the data set is 207 and minimum 1. Since the movements are in order and grow of one, there will be items with minimum dimension of the table of 207 lines. The number of initial items is given by:

-5 different accents

-7 bass notes

- all the possible tonalities

I ran several experiments but the only result meaningful I could calculate on my laptop was retrieved with relative support of 97%:

1 <3>

2 <5>

It means that medium accents and very strong accents are the most are the most used during movements.

The fact that I have only two results is a limit of my machine. The operative system was killing the program after a time out in Windows. When I was running it on a Linux machine It was computing, but the limit of the RAM couldn't permit the termination of the algorithm.

The second dataset I tried to analyze is the diabetes data set. Link:

<https://archive.ics.uci.edu/ml/datasets/diabetes> .

After a preprocessing phase, the data frame was composed by 4 columns (patient_id, timesec, code, value), they are related as follows: every line contains the id of the patient then the timestamp of occurrence of an event, then the code of the event and in the last column a blood measurement for that timestamp. I discretized the level of sugar is 6 categories:

"hypo"= hypoglycemic level -> 0-69

"norm_f_bm"= normal level in fasting condition and before meal->70-100

"norm_bm_bt"= normal level before meal and during bed time->100-130

"norm_am_bt"= normal level after meal and during bed time->130-140

"norm_am"= normal level after meal->140-180

"hyper" = critical level over maximal threshold-> 180-1000 (over 1000 the patient enter a phase of ketoacidosis)

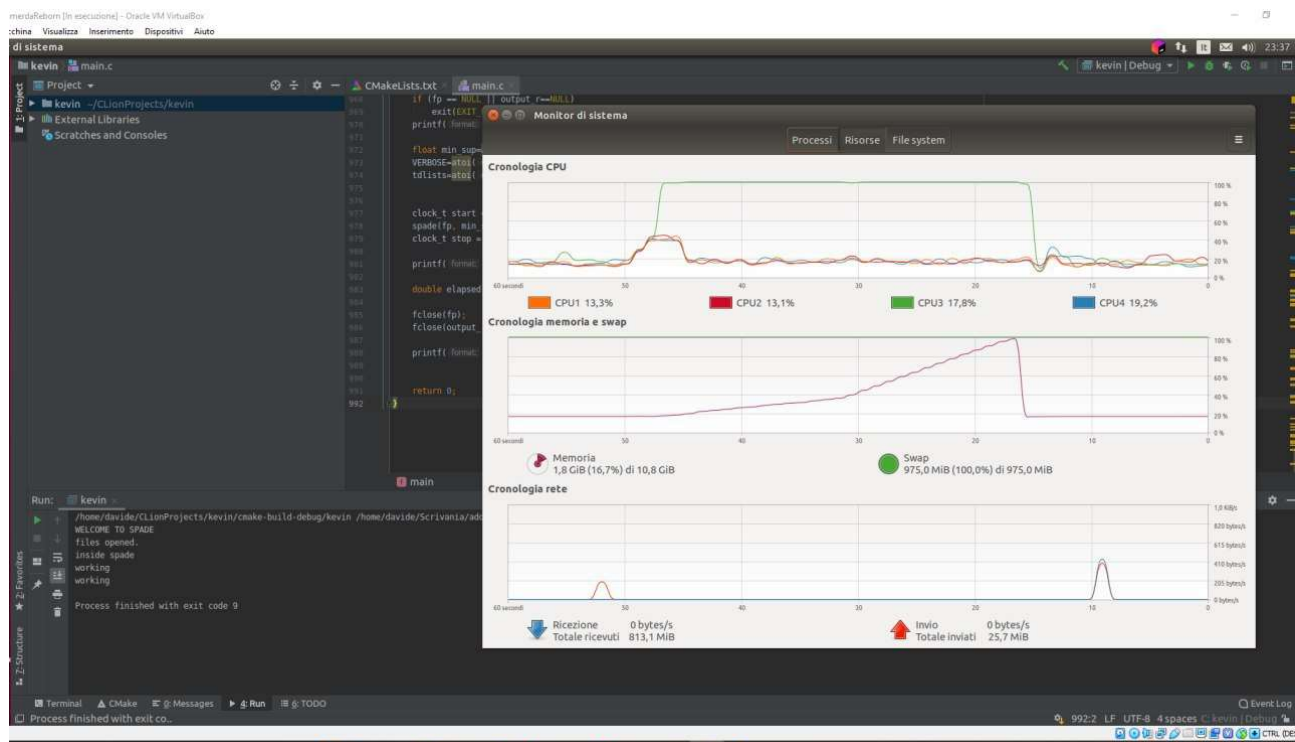
I then converted the data frame in the correct form to be inputted to the algorithm.

Unlucky I didn't have enough memory to perform the analysis with my version of the algorithm.

Issues

After the two failures I stopped the analysis, which it's guaranteed that with powerful machines can give correct results but with the ones in mine possession it's not possible.

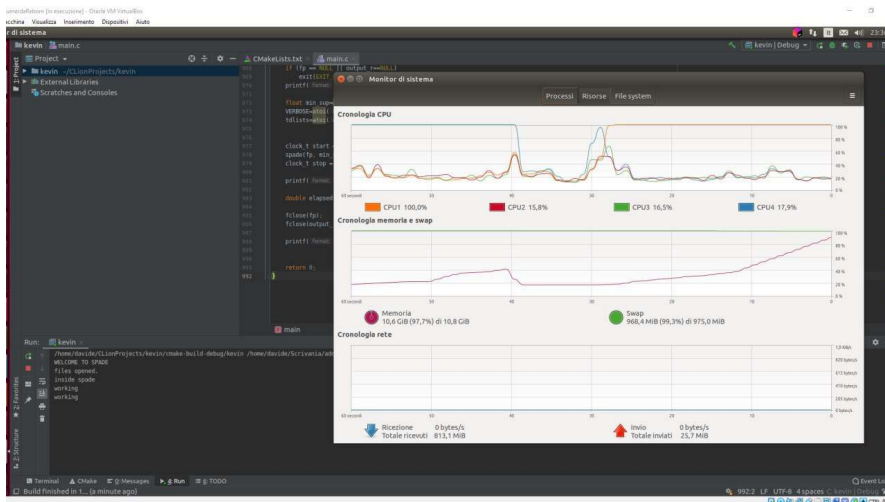
I ran the program in debug mode and I discovered that the memory is filled quickly because of the recursive nature of the algorithm. The implementation was directly inspired by the original paper [1] which explains the algorithm in recursive terms. Obviously after experimenting this is not the way to do it.



As soon as the execution enters the recursive part of the program the memory used grows linearly with the level.

Depending on the data set the behavior obviously is different:

It's possible to see from the plot the moment where tables are freed and new candidates are built.



The second issue.

The second issue found is related to the libc functions. The strlen function didn't reach the termination. After many analysis with Valgrind Memcheck I discovered some fixes.

The only real problem is the free function which doesn't seem to actually free some of the data structures in the program.



The third issue encountered is the difficulty in debugging the results.

On a small data set which I artificially made, the Zaki implementation and the program implemented have different results but surprisingly the program of this report has correct results. The correctness of the results is in terms of v-idlist. The sids given by the zaki implementation are less than the ones given by the program. This problem was encountered at 2-sequence level. It could mean that the preprocessing phase could be wrong in the original one.

The program output:

8 <{A},{N}> sup: 1 rsup: 0.250 sid: { 4 }

9 $\langle \{A,B\} \rangle$ sup: 3 rsup: 0.750 sid: { 1 2 3 }

10 $\langle \{A,E\} \rangle$ sup: 2 rsup: 0.500 sid: { 1 4 }

Zaki implementation's results:

38 $\langle \{ "A" \}, \{ "E" \} \rangle$ {1,2,3,4}



39 $\langle \{ "A", "E" \} \rangle$ {4} <- error

Conclusions

Relatively to the program, It's obvious the limited capability of the work. The way to go was to implement the algorithm in a iterative way. Following the original paper from Zaki didn't give the best results. The assumptions made in the beginning weren't actually true and the memory errors due to the library had a noticeable impact in the developing process.

A way to solve many of the issues related would have been implementing the algorithm with a language with garbage collection and the use of more sophisticated library would have made faster and more efficient the developing process.

References

- 1 **Mohammed J. Zaki** (2001) *SPADE: An Efficient Algorithm for Mining Frequent Sequences*. Machine Learning Journal, 42(1/2), Jan/Feb, pp.31-60. . special issue on Unsupervised Learning.  ([BibTeX](#))]
- 2 **Mohammed J. Zaki** (1998) *Efficient Enumeration of Frequent Sequences*. In 7th ACM International Conference on Information and Knowledge Management. Nov  ([BibTeX](#))]
- 3 **Marzena Kryszkiewicz** (2019) *Lecture Notes on Sequential Patterns*: [mkr_Lecture_Sequential_Patterns21_notes.pdf](#) (email mkr@ii.pw.edu.pl).