First steps with Music-DFS

February 4, 2008

Music-DFS mines soundly and completely constraint-based patterns. It is an implementation of Music [1] which relies on a depth first search method[1].

This document proposes a quick help dedicated to only describe Music-DFS. In particular, we do not define here concepts about pattern mining.

1 Let's go!

First, how to mine the patterns which are present at least twice in the dataset?

```
$ music-dfs -i data.bin -q "count(x)>=2;"
# music-dfs 0.0.5
1 & 4
1 3 , 2 4 & 2
1 5 & 3
1 5 2 & 2
1 4 , 2 & 2
1 2 & 3
6 & 2
4 & 3
4 3 , 2 & 2
4 5 & 2
4 2 & 2
3 & 3
3 2 & 2
5 & 4
5 2 & 2
2 & 3
```

Description of the command line:

- `-i` : specify the input file i.e., the dataset (see Section 2.1 for a description of `data.bin`).
- `-q` : specify the constraint (here `count(x)>=2;`). Let us note that the constraint is ended by a `;` and you can use both lower and upper cases.

Description of the output:

- The first line is a comment...
- `1 & 4` says that the pattern 1 has a frequency of 4. Similarly, the frequency of the pattern 1 5 is 3.
- `1 3 , 2 4 & 2` say that all the patterns of the interval $[\{1,3\},\{1,2,3,4\}]$ have a frequency of 2.

See also `-g` in Section 3 to speed-up computation of frequent patterns.

See also Section 4.3 to discover other primitives useful to build new constraints.

[1] The intervals exploited by the solver are quite different. Indeed, the free and closed patterns depends on a very particular closure operator.

2 File formats

This section depicts the different formats used by Music-DFS.

2.1 Dataset: binary format (`.bin`)

The binary format is the input format for describing the datasets. The dataset with Music-DFS has to be specified after the option `-i`.

An example of dataset (e.g., `data.bin`):

```
$ cat data.bin
# data.bin
1 2 5 6
1 5
1 2 3 4
1 2 3 4 5
4 5
3 6
```

Description:

- `#`: comment a line
- Each number corresponds to an attribute.
- Each line (without `#`) is an object. For instance, `1 2 5 6` is the object 1, `1 5` is the objct 2 and so on.

Remarks:

- This format is also called ascii format.

2.2 Outputed patterns

The option `-o` enables the user to output the patterns in a file. Otherwise, the default output is the standard output.

Let us assume that we are interesting in the 2-frequent patterns, the output is as below:

```
$ music-dfs -i data.bin -q "count(x)>=2;" -g 2
# music-dfs 0.0.5
1 & 4
1 3 , 2 4 & 2
1 5 & 3
1 5 2 & 2
1 4 , 2 & 2
1 2 & 3
6 & 2
4 & 3
4 3 , 2 & 2
4 5 & 2
4 2 & 2
3 & 3
3 2 & 2
5 & 4
5 2 & 2
2 & 3
```

Description:

- The first line is only a comment which gives the name of the solver.

- The collection of outputted patterns (described by intervals) are exactly the patterns present in `data.bin` and satisfying $count(x) \geq 2$.
- Each line gives an interval (before the `&`) and its frequency (after the `&`).
- An interval $[X, Y]$ is represented by X (before the comma) and $Y \setminus X$ (after the comma). Whenever $Y \setminus X$ is the empty set, the comma is not printed.

Remarks:

- Any pattern satisfying the contraint is contained in one interval.
- The measures are printed after the frequency. The separator is also `&`.

See also option `-l` to swtich the output in latex mode.
See also option `-m` to print additional measures.

2.3 Translation (`.traduc`)

This file enables to convert each number (corresponding to an attribute) into a string of characters.

For instance, we can replace each number by a letter (e.g., `attributes.traduc`):

```
$ cat attributes.traduc
1 A
2 B
3 C
4 D
5 E
6 F
```

Remarks:

- This principle can be used with the latex mode (i.e., option `-l`). Be careful! some characters have special sense in latex...

2.4 Values of attributes (`.att`)

This kind of files is necessary to specify values for each attribute (or object) such price of items. Such values are used in conjunction of specific primitives (e.g., `SUM`, `MAX` and so on, see Section 4.3).

For instance, the file `values.att` gives two values DIV and $PRICE$ for the different attributes:

```
$ cat values.att
DIV     PRICE
4       15
2       10
4       5
5       20
2       100
2       40
```

- The first line enumerates the different names of values. Let us note that the tabulation is the name separator.
- The second line `4 15` specifies that the DIV value of the first attribute is 4 and its price is 15. Line `2 10` says that the DIV value of attribute 2 equals 2 and the price, 10.
- It is possible to use float...
- No comment is admited.

3 Speed-up computation

3.1 Minimal frequency threshold

You can specify a classical minimal frequency threshold by using the option `-g`. Thereby, you optimize the extraction by preserving the completeness.

By default, the threshold is fixed to 1.

For instance, the mining done in Section 1 can be improved by using `-g 2`:

```
$ music-dfs -i data.bin -q "count(x)>=2;" -g 2
# music-dfs 0.0.5
1 & 4
1 3 , 2 4 & 2
1 5 & 3
1 5 2 & 2
1 4 , 2 & 2
1 2 & 3
6 & 2
4 & 3
4 3 , 2 & 2
4 5 & 2
4 2 & 2
3 & 3
3 2 & 2
5 & 4
5 2 & 2
2 & 3
```

Let us note that you exactly obtain the same collection of patterns.

See also `-t` to specify a relative minimal frequency threshold.

3.2 Negative pruning condition w.r.t. the specialization

The anti-monotone prunings can be exploited by MUSIC-DFS. The option `-p` adds a negative pruning condition w.r.t. to the specialization [2].

```
$ music-dfs -i data.bin -q "count(x)>=2;" -p "length(x)>1;"
# music-dfs 0.0.5
1 & 4
6 & 2
4 & 3
3 & 3
5 & 4
2 & 3
```

All the intervals $[X, Y]$ whose length of X is greater than 1 are pruned.

Few remarks:

- `-g 2` is equivalent to `-p "count(X)<2;"`. Of course, this option is more general than `-g`.
- A further version will automate the computation of the pruning condition.
- The extraction presented above is equivalent to:

  ```
  $ music-dfs -i data.bin -q "count(x)>=2 and [not length(x)>1];" -p "length(x)>1;"
  ```

  More generally, `-q "CONSTRAINT and [not PRUNING];" -p "PRUNING;"` is equivalent to `-q "CONSTRAINT;" -p "PRUNING;"`.

4 Primitive-based constraints

4.1 Few words about constraints

A primitive-based constraint is a simple combination of primitives. You give your constraint after the option `-q`. Don't forget that quotes are often useful.

- X is the variable.
- A pattern is a particular set.
- A set is a list of ordered numbers. For instance, the expression `{1-3,5}` corresponds to the set $\{1, 2, 3, 5\}$.
- All the primitives (see Section 4.3) can be recursively combined.

4.2 Usual constraints

4.2.1 Emerging patterns

Emerging patterns highligth contrasts between two parts of a same dataset (e.g., according to the classes). They have a frequency significantly higher among one set of objects than another.

More formally, for comparing the objects O_1 and the objects O_2, we consider the growth rate of the pattern X:

```
(LENGTH(O2)/LENGTH(O1))*BCOUNT(X,O1)/BCOUNT(X,O2)
```

Assuming that this growth rate is equal to n, the pattern is n times more frequent in the objects O_1 than in the objects O_2.

Examples :

Mining the emerging patterns of the class corresponding to objects `{1,2,3}` with a growth rate is higher or equal to 2:

```
$ music-dfs -i data.bin -q "BCOUNT(X,{1,2,3})/BCOUNT(X,{4,5,6})>=2;"
1 & 4 & 1 2 3 4
2 , 1 & 3 & 1 3 4
1 5 & 3 & 1 2 4
1 6 , 2 5 & 1 & 1
2 6 , 1 5 & 1 & 1
5 6 , 1 2 & 1 & 1
```

The pattern `{1}` has a growth rate equal to 3 because it is 3 times more present in the objects `{1,2,3}` than in the objects `{4,5,6}`.

Most of times a coefficient is required to normalize the size of both classes. For instance, in the below example, we consider that the first class is `{1,2}` and the second class is `{3,4,5,6}`. As the classes do not contain the same number of objects, a coefficient (here 4/2) is necessary.

Mining the patterns twice more present in the objects `{1,2}` compared to the objects `{3,4,5,6}`:

```
$ music-dfs -i data.bin -q "(4/2)*BCOUNT(X,{1,2})/BCOUNT(X,{3,4,5,6})>=2;"
1 & 4 & 1 2 3 4
5 & 4 & 1 2 4 5
6 & 2 & 1 6
1 5 & 3 & 1 2 4
1 6 , 2 5 & 1 & 1
2 5 , 1 & 2 & 1 4
2 6 , 1 5 & 1 & 1
5 6 , 1 2 & 1 & 1
```

Without normalisation factor `(4/2)`, the pattern `{6}` would not be extracted.

4.2.2 Area

The area of a given pattern is its frquency times its length. Intuitively, the area represents the number of 1 covered by the pattern in the boolean matrix. Thereby, the minimal area constraint returns all the rectangles of 1 higher than a certain value.

Example: Mining the patterns having an area higher than 4:

```
$ music-dfs -i data.bin -q "COUNT(X)*LENGTH(X)>=4;"
```

4.3 More primitives...

Many primitives are implemented in MUSIC-DFS 0.0.5. Table 1 provides a brief description of all the primitives.

Primitive	Operands	Name	Comment
`count`	a pattern	frequency	`count(X)` returns the frequency of the pattern X
`length`	a set	length	`length(S)` returns the length of the set
`bcount`	a pattern, a set of objects	partial frequency	`bcount(X,O)` returns the frequency of the pattern X in the subdataset O
`subset`	two sets (affix)	subset	`S1 subset S2` returns true iff $S_1 \subset S_2$
`subseteq`	two sets (affix)	subset or equal	`S1 subseteq S2` returns true iff $S_1 \subseteq S_2$
`supset`	two sets (affix)	supset	`S1 supset S2` returns true iff $S_1 \supset S_2$
`supseteq`	two sets (affix)	supset or equal	`S1 supseteq S2` returns true iff $S_1 \supseteq S_2$
`eq`	two sets (affix)	equal	`S1 eq S2` returns true iff $S_1 = S_2$
`>`	two reals (affix)	greater than	`r1>r2` returns true iff $r_1 > r_2$
`>=`	two reals (affix)	greater than or equal to	`r1>=r2` returns true iff $r_1 \geq r_2$
`<`	two reals (affix)	greater than	`r1<r2` returns true iff $r_1 < r_2$
`<=`	two reals (affix)	less than or equal to	`r1<=r2` returns true iff $r_1 \leq r_2$
`=`	two reals (affix)	equal to	`r1=r2` returns true iff $r_1 = r_2$
`EXT`	a pattern	extension	`EXT(X)` returns the extension of the pattern X
`SUM`	a set, an ident	sum of values	`SUM(S.VAL)` returns the sum of values VAL of each element included in S
`MAX`	a set, an ident	max of values	`MAX(S.VAL)` returns the maximal value VAL of each element included in S
`MIN`	a set, an ident	min of values	`MIN(S.VAL)` returns the minimal value VAL of each element included in S
`+`	two reals (affix)	plus	`r1+r2` returns $r_1 + r_2$
`-`	two reals (affix)	minus	`r1-r2` returns $r_1 - r_2$
`*`	two reals (affix)	times	`r1*r2` returns $r_1 * \times r_2$
`/`	two reals (affix)	slash	`r1/r2` returns r_1/r_2
`union`	two sets (affix)	privation	`S1 union S2` returns $S_1 \cup S_2$
`inter`	two sets (affix)	intersection	`S1 inter S2` returns $S_1 \cap S_2$
`\`	two sets (affix)	union	`S1 \ S2` returns $S_1 \backslash S_2$
`insim`		IN SIMilarity	
`minsim`		Min SIMilarity	
`maxsim`		Max SIMilarity	
`sumsim`		Sum SIMilarity	
`svsim`		Stated Values SIMilarity	
`mvsim`		Missing Values SIMilarity	
`regexp`		REGular EXPression	

Table 1: List of the implemented primitives

5 Need some help?

You can print a short message:

```
$ music-dfs -h
Use: music-dfs -q <CONSTRAINT> -i <FILE>  [OPTION] ...
Mining with a User-SpecifIed Constraint (Depth First Search)
Example: music-dfs -i dataset.bin -o patterns -q "count(X)*length(X)>=100;"

Options:
 -h, --help                         give this message
 -q, --constraint <CONSTRAINT>      constraint to mine
 -p, --pruning <PRUNING>            negative pruning condition w.r.t. spec.
 -m, --measure <MEASURE>            measure of pattern
 -v, --version                      give the version number
 -V, --verbose ...                  verbose mode
 -i, --input <FILE>                 dataset to mine
 -o, --output <FILE>                constrained patterns
 -a, --values <FILE>                values of attributes
 -S, --similarity <FILE>            similarity matrix
 -d, --delta <NUMBER>               maximal number of exceptions
 -g, --gamma <NUMBER>               minimal absolute frequency threshold
 -t, --threshold <[0,1]>            minimal frequency threshold
 -T, --traduction <FILE> ...        traduce outputted patterns
 -l, --latex                        output in latex format
 -s, --statistics                   give several statistics

Report bug to <arnaud.soulet@info.unicaen.fr>.
```

You can easily get the version number:

```
$ music-dfs -v
 music-dfs 0.0.5
```

6 Common errors

This section enumerates few classical errors.

- Do not forget the option `-i` or `-q`
- Do not forget the semicolon at the end of a constraint/measure/pruning expression:

  ```
  $ music-dfs -i data.bin -q "count(x)>=2"
  <arguments>:1:12: expecting SEMI, found ''
  music: no constraint produced
  ```

References

[1] A. Soulet and B. Crémilleux. An efficient framework for mining flexible constraints. In T. B. Ho, D. Cheung, and H. Liu, editors, *PAKDD*, volume 3518 of *Lecture Notes in Computer Science*, pages 661–671. Springer, 2005.

[2] A. Soulet and B. Crémilleux. Exploiting virtual patterns for automatically pruning the search space. In *KDID*, Lecture Notes in Computer Science. Springer, 2005.