About ModifiedGraMi

Frequent subgraph mining (FSM)

- Frequent subgraph mining in a single graph seeks to, given one graph with labeled vertices, find all frequent subgraphs that appear more than a given threshold (called minimum support threshold).
- It is a field that is reaching maturity. That is, no "new" algorithms have been proposed in the last few years, but rather algorithms to solve variations of the problem.
  - We could consider this version as a variation, because
    - We consider the in-degree of the vertices to evaluate whether two subgraphs are isomorphic (a 2-input AND is different from a 9-input AND)
- Idea of FSM algorithms:
  - Candidate subgraphs are generated by growing (extending) them in a BFS or DFS manner.
    - The goal is to not have duplicate candidate subgraphs.
  - The appearance of candidate subgraphs is counted. If the count (called support) is above the threshold, the subgraph is considered frequent.
    - The graph has to be traversed.
    - Candidate subgraphs are compared to the graph data **through isomorphism tests**.
    - This is the most computationally expensive task.

About graph isomorphism

- For graph isomorphism, it is unknown whether it can be solved in polynomial time or it is NP-complete.
- **Sub**graph isomorphism is used to find whether a subgraph exists in a supergraph. It is an NP-hard problem.
  - Detecting subgraph isomorphisms is crucial in frequent subgraph mining.
  - Thus, optimizations for FSM algorithms are directed to have more effective generation of candidates and/or navigating the search space.
- Algorithms for graph isomorphism: Ullman, SD, Nauty, VF, and VF2.
  - Although there is not one best algorithm, VF2 seems to perform best in regards to the size of the graphs to be tested. This is because the space is searched more effectively, reducing time and memory consumption.
  - Still, they are computationally costly.

Existing FSM algorithms

- gSpan
  - It is possibly the most popular FSM algorithm.
  - It traverses the graph in a DFS manner.
  - About candidate generation:
    - Subgraphs are extended through rightmost path extension.
    - Canonical representation with Minimum DFS Codes (M-DSFC) is used to give subgraphs a unique identifier. A DFS Code is a reliable way to describe a graph.
    - After generating a list of candidates, we can prune the list of duplicate candidates by taking a look at the DFS Codes present.

GraMi

- Solves FSM without enumerating all isomorphisms in the graph and instead models the problem as a Constraint Satisfaction Problem (CSP) for each candidate subgraph.
- During candidate generation, GraMi uses gSpan's DFS Codes to identify candidate subgraphs and avoid duplicates.

About ModifiedGraMi

- To each candidate subgraph corresponds a CSP.
  - In short, CSPs act as an extra restriction to more quickly discard potential replicas.
  - A CSP is represented as a tuple (X, D, C).
    - X = ordered set of variables.
      - X has a variable for each node of the subgraph.
    - D = set of domains for each variable in X.
      - It corresponds for the actual vertices in the original graph that map to the variables of the subgraph.
    - C = set of constraints, which are
      - All variables in X have to be different
      - For every variable (x) in X, the label of variable x for vertex v has to be equal to the label vertex v has in the original graph.
      - The labels of the edges between variables match the labels of the edges between vertices in the real graph.
        - This last constraint does not apply to our case, since the edges in our graph are not labeled.
  - A solution for the CSP is an assignment to the variables in X, such that all constraints in C are satisfied. Every solution is an isomorphism to this subgraph candidate and thus an instance of it in the graph!
  - Implications: choosing the right minimum support threshold.

Solution using GraMi (modifiedGraMi)

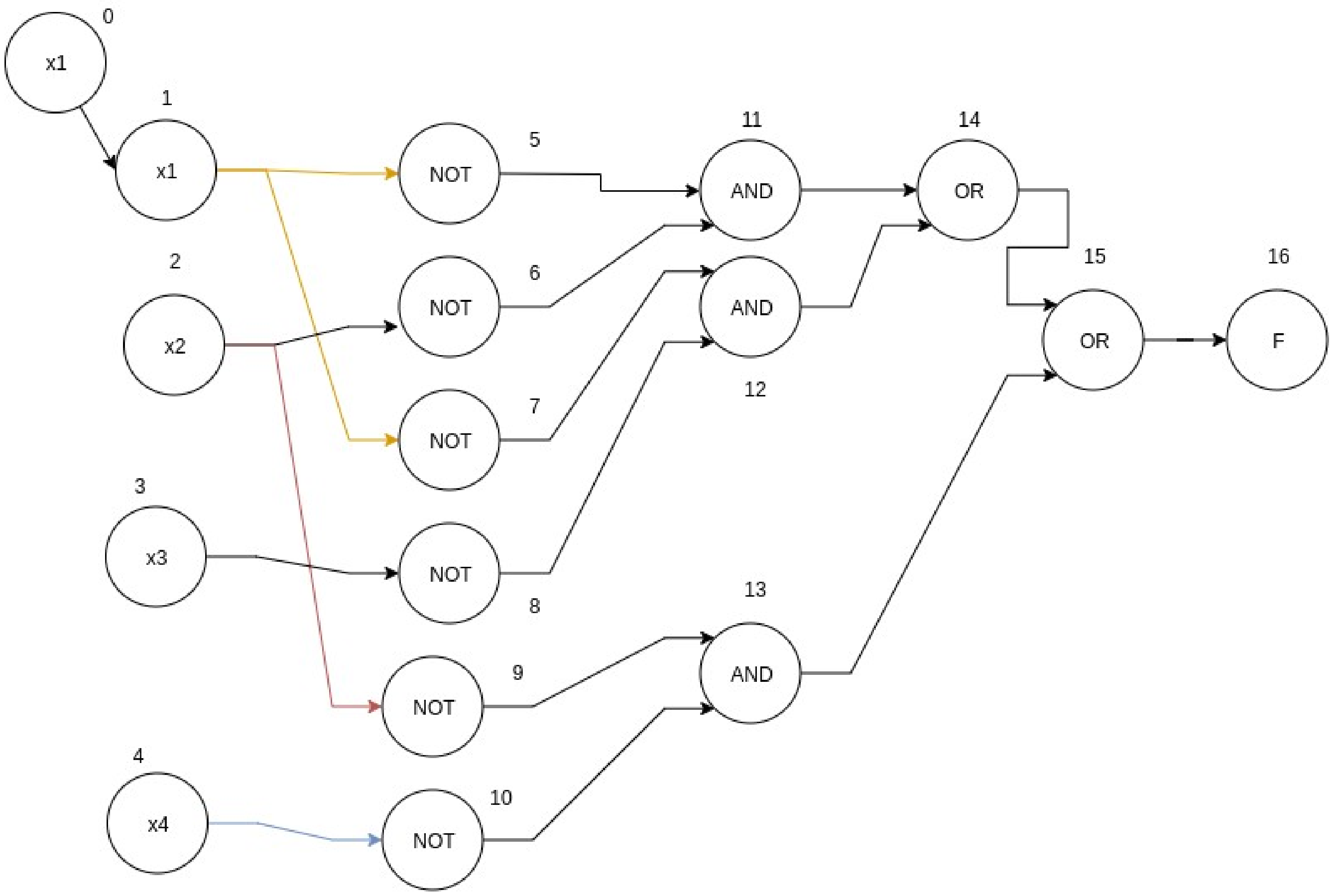
- Goal: find all replicated structures in a single graph that are of a minimum size.
- Input:
  - Graph (set of labeled vertices, set of edges)
  - Minimum size of subgraphs to find
  - (optional) minimum frequency threshold
- Output:
  - Frequent subgraphs of a minimum size and larger
    - We divide the output files by frequency (For example, the file for frequency 3 includes the subgraphs that appear exactly three times).
- Approach
  - Read subgraph and count times every edge appears.
  - Set the maximum count as initial support threshold maxFreq.
  - For freq = maxFreq to freq = 2
    - do FSM(freq);
    - for each frequentSubgraph in frequentSubgraphsFound
      - printInstances();
- Pros
  - Does not print subgraph instances of previously explored subgraphs.
  - Does not consider subgraphs that are smaller than the given minimum size
  - Constantly flushes memory.
    - Outputs found replicas for every freq value at the end of every iteration.
- Implications
  - Potential issues with memory
    - If we want to extract all subgraphs that have at least two instances, we will need to set the minimum support threshold to 2 at some point.

About ModifiedGraMi

- Reducing the threshold results in an exponential increase of the amount of candidate subgraphs (there is less we are able to prune from the graph and so the search space is not reduced much).
- If the graph is big enough and the threshold is small, we might run out of memory.
  - This is an existing limitation of the original version of GraMi and in the field of data science. It is advised to choose a meaningful frequency threshold when running frequent mining algorithms
- Thus, in modifiedGraMi, I offer the option of minThreshold to establish the minimum frequency to consider a subgraph relevant.
  - For instance, if we do not wish to find all replicas but only those that are present 10 times or more, we can set minThreshold to 10. ModifiedGraMi will iterate from maxFreq to minThreshold.

Example:
Input:

- minSize = 3
- Graph (description is in file sample1.lg)

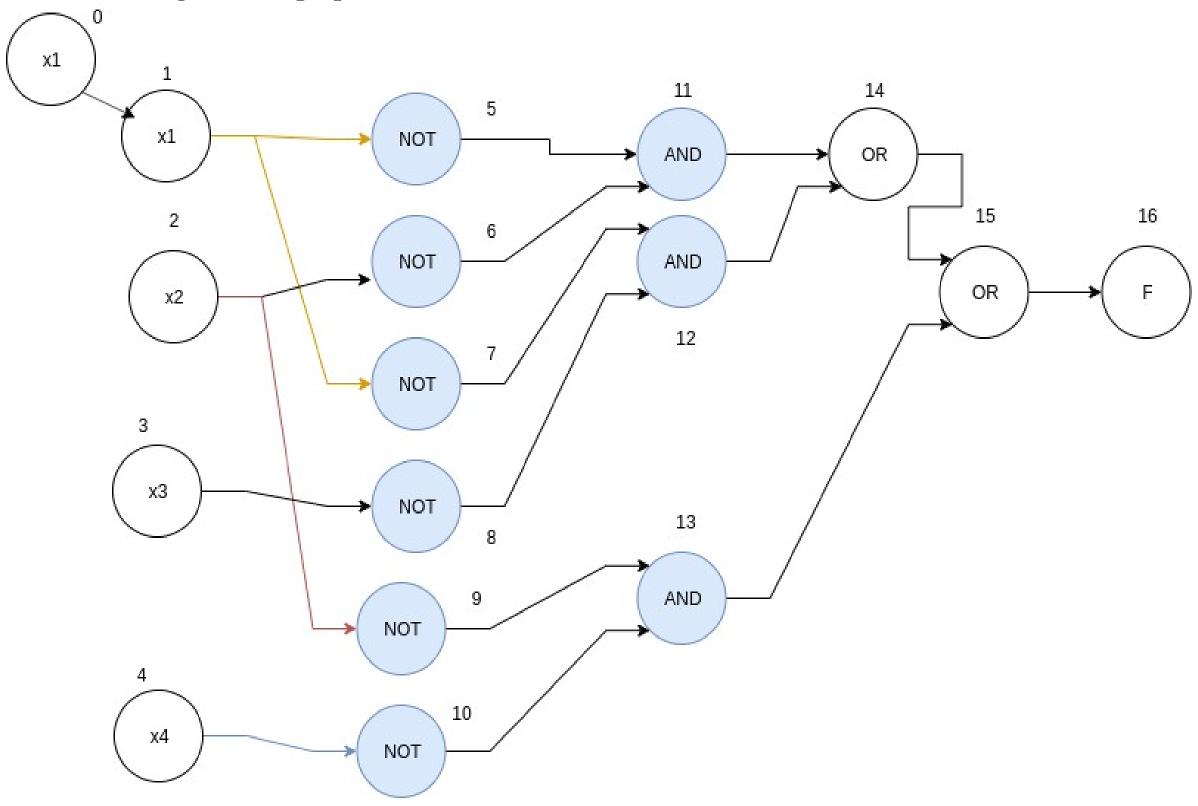


1. Read the graph and count frequency of edges
   1. Note: we count only once the edges that “share” vertices.
      1. For instance vertices 5 and 6 are both a one-input NOT that connect to the same 2-input AND (vertex 11). This NOT → AND edge is counted only once here instead of twice. This is because the focus of this FSM is to not find overlapping subgraph instances.

About ModifiedGraMi

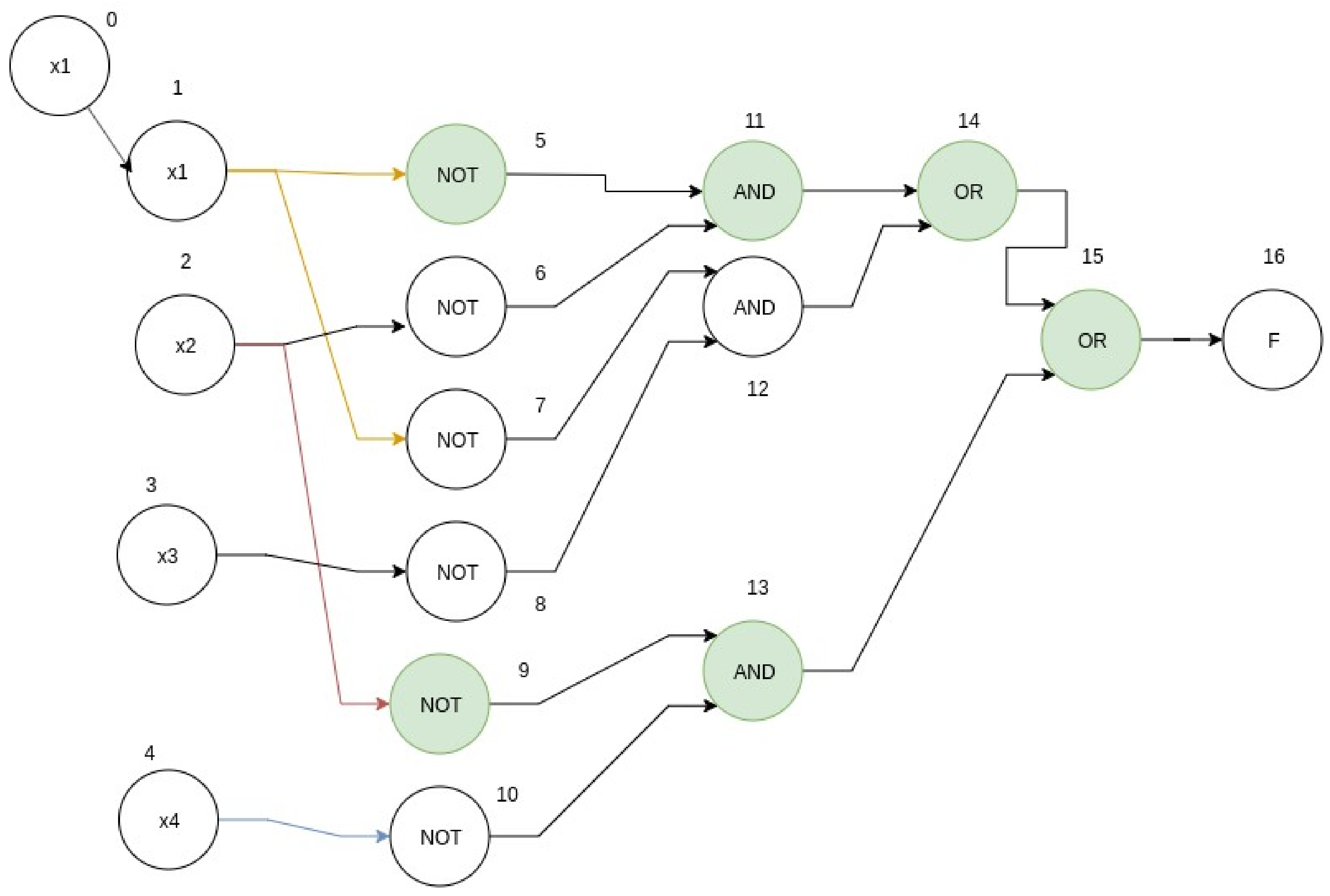
2. X1 → X1: 1
3. X1 → NOT: 1
4. X2 → NOT: 1
5. X3 → NOT: 1
6. X4 → NOT: 1
7. NOT → AND: 3
8. AND → OR: 2
9. OR → OR: 1
10. OR → F: 1

2. Set maxFreq as the maximum count of the edges
   1. maxFreq = 3
3. For freq = 3 to 2 {do FSM(freq) }
   1. freq = 3
      1. Frequent subgraphs found = 1

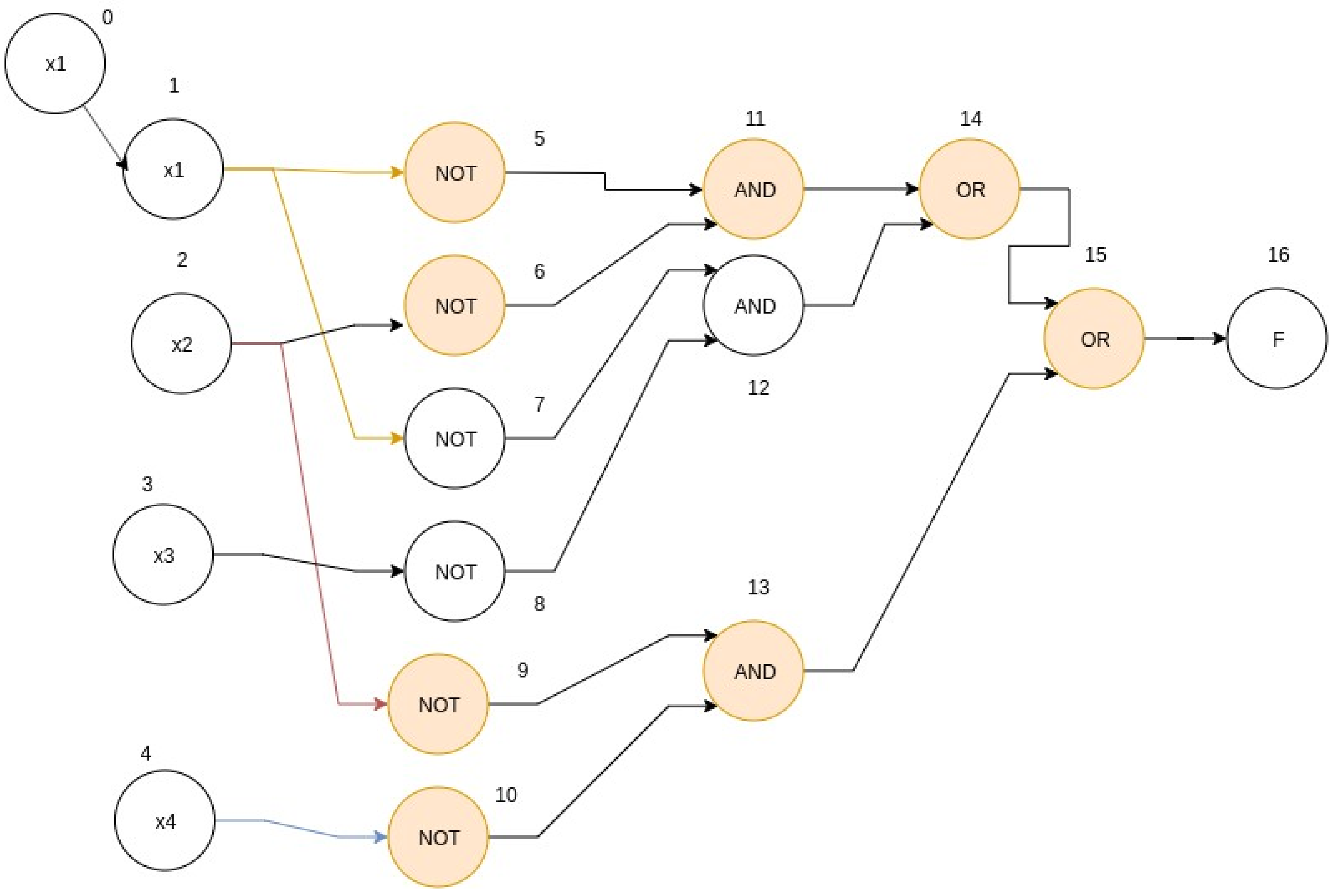


   2. freq = 2
      1. Frequent subgraphs found = 2

About ModifiedGraMi



About ModifiedGraMi



Notice how the subgraph found when freq = 3 is not considered for when freq = 2. Since we iterate from larger support thresholds to smaller ones, ModifiedGraMi keeps track of the subgraphs that were already explored and avoids processing them again for lower thresholds!

Interpreting the output:
Running ModifiedGraMi may result in more than one output file.
Output is divided by frequency. That is, every file contains frequent subgraphs that appear the same number of times inside the graph.
Running the previous example would produce two files.

- Output_mod_freq3.txt
- Output_mod_freq2.txt

Let us look at the contents of each:

1. Output_mod_freq3.txt

File contents	Meaning
0.074 1 0: Size: 3 v 0 1-NOT v 1 2-AND	Time elapsed Number of subgraphs found Frequent subgraph #0 Size of subgraph (number of vertices) Print out of the generic subgraph

v 2 1-NOT e 0 1 e 2 1 Number of instances: 3 Instances: #0: Nodes involved: [5, 6, 11] 5 -> 11 6 -> 11 #1: Nodes involved: [7, 8, 12] 7 -> 12 8 -> 12 #2: Nodes involved: [9, 10, 13] 9 -> 13 10 -> 13	Number of replicas found Print out of each isomorphic subgraph

2. Output_mod_freq2.txt

File contents	Meaning
0.204 3 0: Size: 4 v 0 1-NOT v 1 2-AND v 2 2-OR v 3 1-NOT e 0 1 e 1 2 e 3 1 Number of instances: 2 Instances: #0: Nodes involved: [5, 6, 11, 14] 5 -> 11 11 -> 14 6 -> 11 These appear to be variations of this graph: Variation #1 Nodes involved: [7, 8, 12, 14] #1: Nodes involved: [9, 10, 13, 15]	Time elapsed Number of subgraphs found (it says 3 because of the subgraph in the other file) Frequent subgraph #0 Size of subgraph (number of vertices) Print out of the generic subgraph #0 Number of replicas found Print out of each isomorphic subgraph Instance 0 We have found variations for this subgraph. This means we could pick different vertices to produce the same subgraph. This appears to happen as a result of trying to find the solution to the CSP. The algorithm might propose different vertices to solve the constraints

9 -> 13 13 -> 15 10 -> 13	and might then find out that they work as possible solutions.
1:	Frequent subgraph #1
Size: 3	Size of subgraph (number of vertices)
v 0 1-NOT v 1 2-AND v 2 2-OR e 0 1 e 1 2	Print out of the generic subgraph #1
Number of instances: 2	Number of replicas found
Instances:	Print out of each isomorphic subgraph
#0: Nodes involved: [5, 11, 14] 5 -> 11 11 -> 14	Instance 0
These appear to be variations of this graph:	
Variation #1 Nodes involved: [6, 11, 14] Variation #2 Nodes involved: [7, 12, 14] Variation #3 Nodes involved: [8, 12, 14]	We have found variations for this subgraph.
#1: Nodes involved: [9, 13, 15] 9 -> 13 13 -> 15	Instance 1
These appear to be variations of this graph: Variation #1 Nodes involved: [10, 13, 15]	We have found variations for this subgraph.

References:

1. Elseidy, M., Abdelhamid, E., Skiadopoulos, S. & Kalnis, P. (2014). GRAMI: Frequent Subgraph and Pattern Mining in a Single Large Graph. *PVLDB*, 7(7)
2. Jiang, C., Coenen, F. & Zito, M. (2004). A Survey of Frequent Subgraph Mining Algorithms. *The Knowledge Engineering Review*, 000(1), 1-31.