

XEGGORA Technical Report

Mohammad Mahdi Amirian
m.amirian@gmail.com

August 2020

XEGGORA is an extension to ROCKIT [NNS13], a MAP inference engine for Markov Logic Networks. The extension contains some bug fixes, a complementary Full Constraint Aggregation (FCA) by incorporating Higher-Order Aggregations (HOA) [AS19], the support for parameterized weights syntax on multi-literal formulas and Cumulative Weights Aggregation (CUWA) [Ami20].

1- Executing XEGGORA

The installation, usage and the MLN syntax are all the same as in ROCKIT. The only change in the usage is the addition of an integer parameter: `aggregation_order` which is assigned the following values for different types of aggregations:

- 0: no aggregation
- 1: CPA
- 2: quadratic (2nd order) aggregation (deprecated)
- 1: FCA
- 2: CUWA

To run the XEGGORA main inference engine, type this in command line with the required parameters after installing and configuring Gurobi and MySQL:

```
java -jar xeggora.jar
```

to test the methods on benchmarks, a separate main class should be invoked within the package with required parameters:

```
java -cp ./xeggora.jar com.mlnengine.xeggora.test.BenchmarkTests
```

Here we describe how to run the inference engine for applying FCA and CUWA to MLN models.

2- Full Constraint Aggregation

FCA automatically chooses appropriate orders of aggregations and applies them. An HOA can be applied to sets of clauses with specific conditions outlined in [AS19]. For example, The MLN:

```
*Child(kid, person)
Kind(person)
HasFunWith(person, kid)
Happy(kid)

// Child(k, p) ^ Kind(p) ^ HasFunWith(p, k) => Happy(k)
2.3 !Child(k, p) v !Kind(p) v !HasFunWith(p, k) v Happy(k)
```

and the evidence:

```
Child(Mary, Jack)
Child(Mary, Rose)
Child(Bob, Jack)
Child(Kate, Jack)
Kind(Jack)
HasFunWith(Rose, Mary)
HasFunWith(Jack, Bob)
```

by setting the `aggregation_order` parameter to -1 (FCA) in the file `xeggora.properties` and running this in command line:

```
java -jar xeggora.jar -input prog.mln -data evidence.db -output output.db
```

the engine would replace the evidence atoms with their values and aim at finding the MAP solution by aggregating the following clauses:

```
2.3 !HasFunWith(Jack, Mary) v Happy(Mary)
2.3 Happy(Bob)
2.3 !HasFunWith(Jack, Kate) v Happy(Kate)
```

with the following log:

```
==== Start Xeggora =====Wed Feb 27 15:26:39 GMT 2019
-input prog.mln
-data evidence.db
-output output.db
-gap: use gurobi standard gap.
-debug output false
-use MAP inference
Academic license - for non-commercial use only
===== Start Standard Grounder =====Wed Feb 27 15:26:40 GMT 2019
.....PutDataIntoTables duration: 15

Number of formulas: 3
#Hard Formulas: 2
#Soft Formulas: 1
#GSW Formulas: 0
#Soft Formulas with nonzero weight: 1
#Objective Formulas: 0
#Cardinality Formulas: 0
12 evidence atoms
Aggregation order -1
===== Start Standard Solver =====Wed Feb 27 15:26:40 GMT 2019
#AggregatedHardFormulas: 2
#AggregatedSoftFormulas: 0
#AggregatedClauses[1]: 2
#CountingConstraintsAggregatingMoreThanOneClause[1]: 1
#CountingConstraintsWithMoreThanOneLiteral[1]: 0
#ConstraintsAggregatedByContingConstraintWithMoreThanOneLiteral[1]: 0
#CountingConstraintsWithOneLiteral[1]: 2
#ConstraintsAggregatedByContingConstraintWithOneLiteral[1]: 3
```

```

#ZVariables: 0
#BinaryVariables: 3
#MixedVariables: 0

#AggregatedHardFormulas: 2
#AggregatedSoftFormulas: 1
#AggregatedClauses[1]: 2
#CountingConstraintsAggregatingMoreThanOneClause[1]: 1
#CountingConstraintsWithMoreThanOneLiteral[1]: 0
#ConstraintsAggregatedByContingConstraintWithMoreThanOneLiteral[1]: 0
#CountingConstraintsWithOneLiteral[1]: 2
#ConstraintsAggregatedByContingConstraintWithOneLiteral[1]: 3
#AggregatedClauses[2]: 2
#CountingConstraintsAggregatingMoreThanOneClause[2]: 1
#CountingConstraintsWithMoreThanOneLiteral[2]: 1
#ConstraintsAggregatedByContingConstraintWithMoreThanOneLiteral[2]: 1
#CountingConstraintsWithOneLiteral[2]: 1
#ConstraintsAggregatedByContingConstraintWithOneLiteral[2]: 3
#ZVariables: 2
#BinaryVariables: 12
#MixedVariables: 2

---- Start Loop 0 ----Wed Feb 27 15:26:40 GMT 2019
add 6 constraints (total number = 6)
Optimize a model with 6 rows, 13 columns and 16 nonzeros
Variable types: 0 continuous, 13 integer (12 binary)
Coefficient statistics:
  Matrix range      [1e+00, 1e+00]
  Objective range   [2e+00, 2e+00]
  Bounds range      [1e+00, 3e+00]
  RHS range         [1e+00, 2e+00]
Found heuristic solution: objective 9.20000000

Explored 0 nodes (0 simplex iterations) in 0.00 seconds
Thread count was 1 (of 4 available processors)

Solution count 1: 9.2

Optimal solution found (tolerance 1.00e-04)
Best objective 9.200000000000000e+00, best bound 9.200000000000000e+00, gap 0.0000%
ILP took 9 Milliseconds.
#AggregatedHardFormulas: 0
#AggregatedSoftFormulas: 0
#AggregatedClauses[1]: 2
#CountingConstraintsAggregatingMoreThanOneClause[1]: 1
#CountingConstraintsWithMoreThanOneLiteral[1]: 0
#ConstraintsAggregatedByContingConstraintWithMoreThanOneLiteral[1]: 0
#CountingConstraintsWithOneLiteral[1]: 2
#ConstraintsAggregatedByContingConstraintWithOneLiteral[1]: 3
#AggregatedClauses[2]: 2
#CountingConstraintsAggregatingMoreThanOneClause[2]: 1
#CountingConstraintsWithMoreThanOneLiteral[2]: 1
#ConstraintsAggregatedByContingConstraintWithMoreThanOneLiteral[2]: 1
#CountingConstraintsWithOneLiteral[2]: 1
#ConstraintsAggregatedByContingConstraintWithOneLiteral[2]: 3
#ZVariables: 2
#BinaryVariables: 12
#MixedVariables: 2

Total ILP Time: 9
Dispose model
Dispose environment
Database connection terminated
Xeggora runtime was 727 milliseconds.

```

The model is converted to the following ILP:

```
Maximize
    2.3 z0 + 2.3 z1
Subject To
    R0: HasFunWith|d|b + HasFunWith|c|e >= 2
    R1: Kind|c >= 1
    R2: - ~HasFunWith|c|b_v_Happy|b - HasFunWith|c|b + Happy|b >= -1
    R3: - ~HasFunWith|c|f_v_Happy|f - HasFunWith|c|f + Happy|f >= -1
    R4: - z0 + ~HasFunWith|c|b_v_Happy|b + ~HasFunWith|c|f_v_Happy|f + Happy|e
        >= 0
    R5: Happy|b - z1 - Kind|d >= -1
Bounds
    z0 <= 3
Binaries
    HasFunWith|d|b HasFunWith|c|e Kind|c ~HasFunWith|c|b_v_Happy|b
    HasFunWith|c|b Happy|b ~HasFunWith|c|f_v_Happy|f HasFunWith|c|f Happy|f
    Happy|e z1 Kind|d
Generals
    z0
End
```

and Gurobi is invoked to solve it. The identifiers: b, c, d, e and f in this ILP correspond to Mary, Jack, Rose, Bob and Kate in the MLN, respectively. The result would be translated as the MAP inference solution:

```
Happy("Mary")
Kind("Rose")
Happy("Bob")
Kind("Jack")
Happy("Kate")
HasFunWith("Jack", "Kate")
HasFunWith("Jack", "Bob")
HasFunWith("Jack", "Mary")
HasFunWith("Rose", "Mary")
```

Note that by activating CPI, another valid result would be generated as follows, because CPI may reach easy-to-access solutions in short times:

```
HasFunWith("Jack", "Bob")
HasFunWith("Rose", "Mary")
Happy("Bob")
Kind("Jack")
```

This solution is the result of two ILP solving iterations:

iteration 1:

```
Maximize

Subject To
    R0: HasFunWith|d|b + HasFunWith|c|e >= 2
    R1: Kind|c >= 1
Bounds
Binaries
    HasFunWith|d|b HasFunWith|c|e Kind|c
End
```

with the solution:

```
# Objective value = 0
HasFunWith|d|b 1
HasFunWith|c|e 1
Kind|c 1
```

This solution doesn't satisfy all constraints, so in the second iteration, the following ILP is generated and solved:

```
Maximize
  2.3 z0
Subject To
  R0: HasFunWith|d|b + HasFunWith|c|e >= 2
  R1: Kind|c >= 1
  R2: - z0 + Happy|e >= 0
Bounds
Binaries
  HasFunWith|d|b HasFunWith|c|e Kind|c z0 Happy|e
End
```

with the solution:

```
# Objective value = 2.3
z0 1
HasFunWith|d|b 1
HasFunWith|c|e 1
Kind|c 1
Happy|e 1
```

which satisfies all the constraints.

3- CUmulative Weights Aggregation

To apply CUWA to Grounding-Specific Weighted formulas, we should first introduce them to the inference engine in the syntax of parameterized weights as mentioned in [Ami20]. As no learning engine currently support this syntax, the learned weights should be manipulated. For example, the formula with the following learned GSWs:

```
-1.3 !Kind(Jack) v !HasFunWith(Jack, Mary) v Happy(Mary)
-1.3 !Kind(Rose) v !HasFunWith(Rose, Mary) v Happy(Mary)
2.1 !Kind(Jack) v !HasFunWith(Jack, Bob) v Happy(Bob)
-0.5 !Kind(Jack) v !HasFunWith(Jack, Kate) v Happy(Kate)
1.1 !Kind(Jack) v !HasFunWith(Jack, Alice) v Happy(Alice)
```

would be introduced to XEGGORA as the MLN:

```
Child(kid, person)
Kind(person)
Happy(kid)

//NormalKid(+k, float_): Child(k, p) ^ Kind(p) ^ HasFunWith(p, k) => Happy(k)
*NormalKidP(kid, float_)
*NormalKidN(kid, float_)
conf: !NormalKidP(k, conf) v !Child(k, p) v !Kind(p) v !HasFunWith(p, k) v Happy(k)
cNEG: !NormalKidN(k, cNEG) v !Child(k, p) v !Kind(p) v !HasFunWith(p, k) v Happy(k)
```

with the learned weights as the following first four lines in the evidence:

```
NormalKidN(Mary, -1.3)
NormalKidP(Bob, 2.1)
NormalKidN(Kate, -0.5)
NormalKidP(Alice, 1.1)
Child(Mary, Jack)
Child(Mary, Rose)
Child(Bob, Jack)
Child(Kate, Jack)
Child(Alice, Jack)
```

The commented line in the MLN shows our standard syntax for parameterized weights, but it has not yet been recognized by Xeggora but accepted temporarily with minor format differences: Instead of

each evidential predicate (NormalKid in the example) one should define two distinct predicates for positive and negative weights if any (e.g. NormalKidP and NormalKidN), and the real variable for assigning negative weights must end with “NEG” (e.g. cNEG). By setting the aggregation_order parameter to -2 and running XEGGORA, the Cumulative Weights Aggregation procedure is called and the MLN is converted to the following MILP:

```
Maximize
    z0 + z1 - 0.5 z2
Subject To
    R0: - ~HasFunWith|f|c_v_Happy|c - HasFunWith|f|c + Happy|c >= -1
    R1: - ~HasFunWith|f|e_v_Happy|e - HasFunWith|f|e + Happy|e >= -1
    R2: - z0 + 2.1 ~HasFunWith|f|c_v_Happy|c + 1.1 ~HasFunWith|f|e_v_Happy|e
        - 3.2 Kind|f >= -3.2
    R3: - 2 ~Kind|g_v_~HasFunWith|g|b - Kind|g - HasFunWith|g|b <= -2
    R4: - Kind|f - 2 ~Kind|f_v_~HasFunWith|f|b - HasFunWith|f|b <= -2
    R5: - z1 - 1.3 ~Kind|g_v_~HasFunWith|g|b - 1.3 ~Kind|f_v_~HasFunWith|f|b
        >= 0
    R6: - z1 - 2.6 Happy|b >= 0
    R7: - Kind|f - 3 z2 - HasFunWith|f|d + Happy|d <= -2
Bounds
    z0 <= 3.2
    -2.6 <= z1 <= 0
Binaries
    ~HasFunWith|f|c_v_Happy|c HasFunWith|f|c Happy|c ~HasFunWith|f|e_v_Happy|e
    HasFunWith|f|e Happy|e Kind|f ~Kind|g_v_~HasFunWith|g|b Kind|g
    HasFunWith|g|b ~Kind|f_v_~HasFunWith|f|b HasFunWith|f|b Happy|b z2
    HasFunWith|f|d Happy|d
End
```

with the solution:

```
# Objective value = 3.2
z0 3.2
z1 0
z2 0
~HasFunWith|f|c_v_Happy|c 1
HasFunWith|f|c 0
Happy|c 1
~HasFunWith|f|e_v_Happy|e 1
HasFunWith|f|e 0
Happy|e 1
Kind|f 1
~Kind|g_v_~HasFunWith|g|b 0
Kind|g 1
HasFunWith|g|b 1
~Kind|f_v_~HasFunWith|f|b 0
HasFunWith|f|b 1
Happy|b 0
HasFunWith|f|d 1
Happy|d 0
```

and the following output solution is generated:

```
Happy("Bob")
Kind("Jack")
Kind("Rose")
Happy("Alice")
HasFunWith("Rose", "Mary")
HasFunWith("Jack", "Kate")
HasFunWith("Jack", "Mary")
```

The MILP shows that the first two clauses have been cumulatively aggregated with respect to Happy(Mary), while the third and the fifth clauses are cumulatively aggregated with respect

to !Kind(Jack). However, the first two clauses can also be classically aggregated (not necessarily by cumulative aggregation). Other alternatives would also exist. The decision to select which target sets to aggregate is somehow heuristic. Again, by activating CPI, the smaller MILP:

```
Maximize
  z0 - 0.5 z1
Subject To
  R0: - 2 ~Kind|f_v_~HasFunWith|f|b - Kind|f - HasFunWith|f|b <= -2
  R1: - 2 ~Kind|g_v_~HasFunWith|g|b - Kind|g - HasFunWith|g|b <= -2
  R2: - z0 - 1.3 ~Kind|f_v_~HasFunWith|f|b - 1.3 ~Kind|g_v_~HasFunWith|g|b
      >= 0
  R3: - z0 - 2.6 Happy|b >= 0
  R4: - Kind|f - 3 z1 - HasFunWith|f|d + Happy|d <= -2
Bounds
  -2.6 <= z0 <= 0
Binaries
  ~Kind|f_v_~HasFunWith|f|b Kind|f HasFunWith|f|b ~Kind|g_v_~HasFunWith|g|b
  Kind|g HasFunWith|g|b Happy|b z1 HasFunWith|f|d Happy|d
End
```

with the solution:

```
# Objective value = 0
z0 0
z1 0
~Kind|f_v_~HasFunWith|f|b 0
Kind|f 1
HasFunWith|f|b 1
~Kind|g_v_~HasFunWith|g|b 0
Kind|g 1
HasFunWith|g|b 1
Happy|b 0
HasFunWith|f|d 1
Happy|d 0
```

in one iteration and a simpler valid final solution is resulted:

```
HasFunWith("Rose", "Mary")
HasFunWith("Jack", "Kate")
HasFunWith("Jack", "Mary")
Kind("Jack")
Kind("Rose")
```

4- BUG Fixes

Among several minor issues, the ROCKIT system has two major bugs in the implementation; One for negative weights and the other in the SQL generation. Formulas with negative weights are not correctly converted to ILPs despite the perfect illustration in the related article [NNS13]. For example, the MLN:

```
Child(kid, person)
*Kind(person, kid)
Rich(person)
Happy(kid)

-1.5 !Kind(p, k) v !Rich(p) v !Child(k, p) v Happy(k)
10 Happy(k)
14 !Child(k, p)
```

with the evidence:

```
Kind(B, C)
```

would be converted to:

```
Maximize
  - 1.5 z0 + 10 z1 + 14 z2
Subject To
  R0: - z0 + Happy|c - Child|c|b <= -1
  R1: - z0 - Rich|b <= -1
  R2: Happy|c - z1 >= 0
  R3: - Child|c|b - z2 >= -1
Bounds
Binaries
  z0 Happy|c Child|c|b Rich|b z1 z2
End
```

with the solution:

```
Optimal objective: 12.5
z0 1
Happy|c 0
Child|c|b 0
Rich|b 1
z1 0
z2 1
```

and the output:

```
Rich("B")
```

But the correct ILP is:

```
Maximize
  - 1.5 z0 + 10 z1 + 14 z2
Subject To
  R0: - 3 z0 - Rich|b + Happy|c - Child|c|b <= -2
  R1: Happy|c - z1 >= 0
  R2: - Child|c|b - z2 >= -1
Bounds
Binaries
  z0 Rich|b Happy|c Child|c|b z1 z2
End
```

with the solution:

```
# Objective value = 22.5
z0 1
z1 1
z2 1
Rich|b 1
Happy|c 1
Child|c|b 0
```

and the output:

```
Rich("B")
Happy("C")
```

This is resolved in XEGGORA. Moreover, XEGGORA has added the aggregation of hard formulas.

The second bug is a logical faulty SQL generation for performing the subtraction operation on SQL tables in CPI iterations. As described in [Noe14], the system generates queries to perform subtraction on tables for grounding and violated constraints extraction. However, the queries fail on the **LEFT JOIN** part in specific cases where the output is conditioned with a non-null comparison (having CPI activated and the `convert_string_values` parameter deactivated). Suppose for example the MLN:


```
Is(adjective, person)
Smokes(person)
0.1 Is("Healthy", p) v Smokes(p)
```

with evidence:

```
Is(Athlete, John)
```

The formula would be internally transformed to:

```
0.1 Smokes(p) v !person_typePred(p) v !varString0(varString0) v Is(varString0, p)
!Is_observed(v0, v1) v Is(v0, v1)
```

and the SQL query for subtraction would be correctly generated as:

```
SELECT (1 * 0.1) as weight, xx.varString0, xx.p FROM (
  SELECT x1.field0 as `varString0`, x0.field0 as `p` FROM
    `person_typePred` x0
  INNER JOIN `varString0` x1
  LEFT JOIN `Smokes` y0
  ON y0.field0 = x0.field0
  WHERE y0.field0 IS NULL
) as xx
```

with the output:

```
0.1, b, d
```

which finally results in:

```
Is("Healthy", "John")
Is("Athlete", "John")
```

But when a parameter is added to the Smokes predicate:

```
Is(adjective, person)
Smokes(person, cigarette)
0.1 Is("Healthy", p) v Smokes(p, "Marlboro")
```

with formulas:

```
0.1 Smokes(p, "c") v !person_typePred(p) v !varString0(varString0) v
Is(varString0, p)
!Is_observed(v0, v1) v Is(v0, v1)
```

with the same evidence, it generates this query:

```
SELECT (1 * 0.1) as weight, xx.varString0, xx.p FROM (
  SELECT x1.field0 as `varString0`, x0.field0 as `p` FROM
    `person_typePred` x0
  INNER JOIN `varString0` x1
  LEFT JOIN `Smokes` y0
  ON y0.field0 = x0.field0
  WHERE y0.field1 = 'c' AND y0.field0 IS NULL
) as xx
```

with the null output which results in the wrong solution:

```
Is("Athlete", "John")
```

This failure is because the comparison: `y0.field1 = 'c'` is incorrectly placed in conjunction with the other null-comparison condition in the **WHERE** part. To solve, we refactored the SQL generation script to generate the following query instead:

```
SELECT (1 * 0.1) as weight, xx.varString0, xx.p FROM (
```

```

SELECT x1.field0 as `varString0`, x0.field0 as `p` FROM
`person_typePred` x0
INNER JOIN `varString0` x1
LEFT JOIN (
    SELECT * FROM `Smokes`
    WHERE field1 = 'c'
) as y0
ON y0.field0 = x0.field0
WHERE y0.field0 IS NULL
) as xx

```

The new query has the correct output:

0.1, b, d

with the correct final solution:

```

Is("Healthy", "John")
Is("Athlete", "John")

```

5- Bug Report

Any bug report of the system would be highly appreciated. In case, please create an issue in the XEGGORA page on GitHub¹, attaching your model and system parameters you have set. Before reporting any bug, please test your model with the underlying ROCKIT system first. If the bug remains in ROCKIT, you had better report the bug via ROCKIT webpages² and mention that in reporting to XEGGORA webpage too.

6- Academic Citation

Please cite us for using XEGGORA and FCA as [AS19], and for using CUWA as [Ami20].

References

- [AS19] Amirian, M. M. and Shiry Ghidary, S. Xeggora: Exploiting immune-to-evidence symmetries with Full Aggregation in Statistical Relational Models, *Journal of Artificial Intelligence Research*, 66, pp. 33-56, 2019.
- [Ami20] Amirian, Mohammad Mahdi. Lifting Untied Formulas for MAP Inference in Markov Logic Networks, *submitted to International Journal of Approximate Reasoning*, 2020.
- [NNS13] Noessner, J., Niepert, M. and Stuckenschmidt, H. RockIt: Exploiting Parallelism and Symmetry for MAP Inference in Statistical Relational Models. *Proceedings of AAAI*, pp. 739-745, 2013.
- [Noe14] Noessner, Jan. Efficient Maximum A-Posteriori Inference in Markov Logic and Application in Description Logics. *PhD diss., University Mannheim*, 2014.

¹ <https://github.com/amirian/xeggora>

² <https://code.google.com/archive/p/rockit>, <https://github.com/jnoessner/rockIt>