Assignment 1

for the course DM840

Fall 2023

1 Formose Cycle

1.1 Depiction of the Four Rules as provided by MØD

The four rules were provided to mød as graph transformation rules in the following form:

- aldol_addition_forward.gml

```
rule [
    ruleID "Aldol Addition ->"
        left [
            edge [ source 1 target 2 label "=" ]
            edge [ source 2 target 3 label "-" ]
            edge [ source 3 target 4 label "-" ]
            edge [ source 5 target 6 label "=" ]
        ]
        context [
            node [ id 1 label "C" ]
            node [ id 2 label "C" ]
            node [ id 3 label "O" ]
            node [ id 4 label "H" ]
            node [ id 5 label "O" ]
            node [ id 6 label "C" ]
        ]
        right [
            edge [ source 1 target 2 label "-" ]
            edge [ source 2 target 3 label "=" ]
            edge [ source 5 target 6 label "-" ]
            edge [ source 4 target 5 label "-" ]
            edge [ source 6 target 1 label "-" ]
        ]
]
```

- aldol_addition_backward.gml

```
rule [
    ruleID "Aldol Addition <-"
        left [
            edge [ source 1 target 2 label "-" ]
            edge [ source 2 target 3 label "=" ]
            edge [ source 5 target 6 label "-" ]
            edge [ source 4 target 5 label "-" ]
            edge [ source 6 target 1 label "-" ]
        ]
        context [
            node [ id 1 label "C" ]
            node [ id 2 label "C" ]
            node [ id 3 label "O" ]
            node [ id 4 label "H" ]
```

```
            node [ id 5 label "O" ]
            node [ id 6 label "C" ]
        ]
        right [
            edge [ source 1 target 2 label "=" ]
            edge [ source 2 target 3 label "-" ]
            edge [ source 3 target 4 label "-" ]
            edge [ source 5 target 6 label "=" ]
        ]
]
```

- keto_enol_forward.gml

```
rule [
    ruleID "Keto-enol isomerization ->"
        left [
            edge [ source 1 target 4 label "-" ]
            edge [ source 1 target 2 label "-" ]
            edge [ source 2 target 3 label "=" ]
        ]
        context [
            node [ id 1 label "C" ]
            node [ id 2 label "C" ]
            node [ id 3 label "O" ]
            node [ id 4 label "H" ]
        ]
        right [
            edge [ source 1 target 2 label "=" ]
            edge [ source 2 target 3 label "-" ]
            edge [ source 3 target 4 label "-" ]
        ]
]
```

- keto_enol_backward.gml

```
rule [
    ruleID "Keto-enol isomerization <-"
        left [
            edge [ source 1 target 2 label "=" ]
            edge [ source 2 target 3 label "-" ]
            edge [ source 3 target 4 label "-" ]
        ]
        context [
            node [ id 1 label "C" ]
            node [ id 2 label "C" ]
            node [ id 3 label "O" ]
            node [ id 4 label "H" ]
        ]
        right [
            edge [ source 1 target 4 label "-" ]
            edge [ source 1 target 2 label "-" ]
            edge [ source 2 target 3 label "=" ]
        ]
]
```

The rules for the forward reaction and backward reaction pairs (aldol addition and keto enol) have the same context with the left and right sides interchanged.

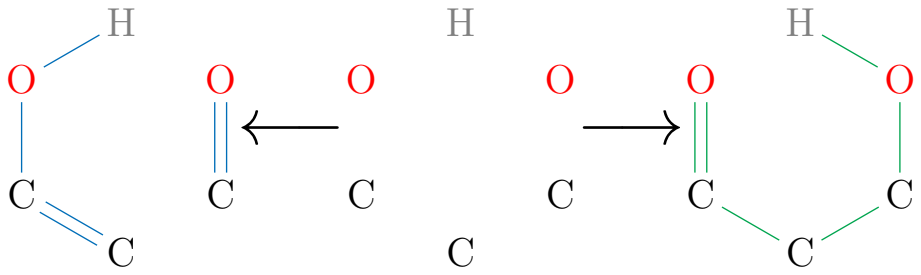


Figure 1: aldol addition forward reaction

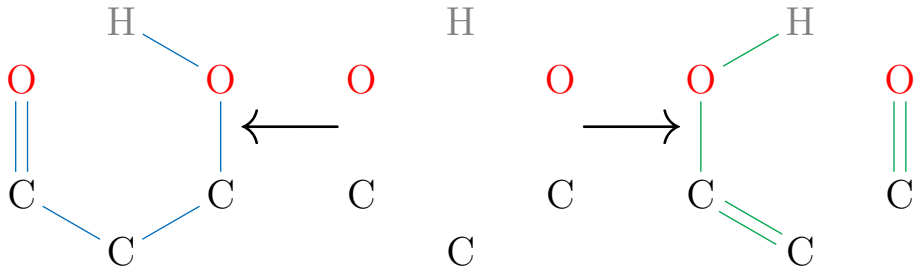


Figure 2: keto-enol tautomerism backward reaction

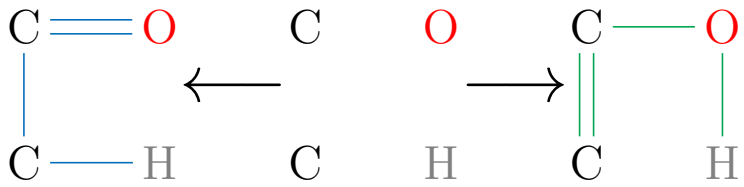


Figure 3: keto-enol tautomerism forward reaction

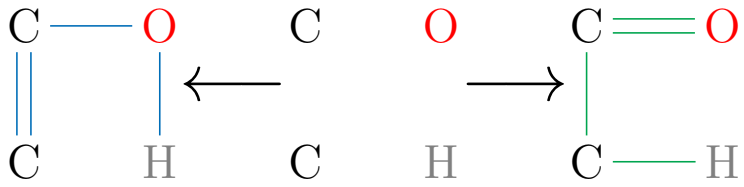


Figure 4: keto-enol tautomerism backward reaction

1.2 Rules in Chemical Form

1.3 Derivation Graph with the Auto-catalytic Cycle

The derivation graph for the auto-catalytic cycles was too large to display and too complicated to discern something meaningful, it was move to the end of this document as Figure 11.

1.4 Objective value to the solution for Auto-catalysis

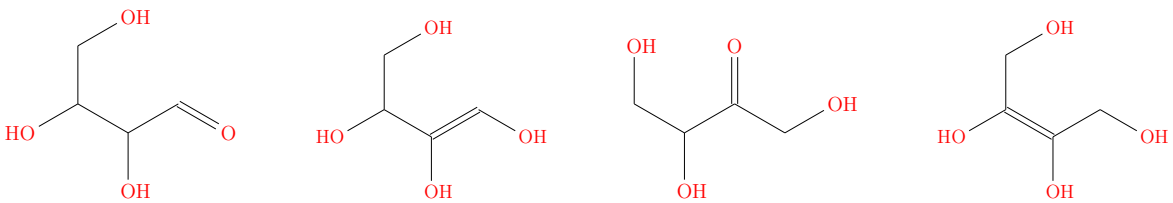


Figure 5: Compounds which are auto-catalytic in the formose cycle.

The solution to the auto-catalytic cycles for tautomers of four carbon compounds are:

1. One flow with `Objective value (integral): 14`

Vertex/Graph	In	Out	OA
Glycolaldehyde	2	0	0
g_{10}	1	2	1

2. One flow with `Objective value (integral): 13`

Vertex/Graph	In	Out	OA
Glycolaldehyde	2	0	0
g_{11}	1	2	1

3. Three flows with `Objective value (integral): 14`

Vertex/Graph	In	Out	OA
Glycolaldehyde	2	0	0
g_{12}	1	2	1

4. Four flows with `Objective value (integral): 15`

Vertex/Graph	In	Out	OA
Glycolaldehyde	2	0	0
g_{13}	1	2	1

2 Reindeer Game

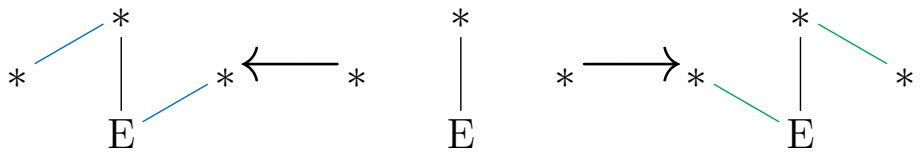


Figure 6: Reindeer move where a reindeer hops to a vacant space one place ahead.

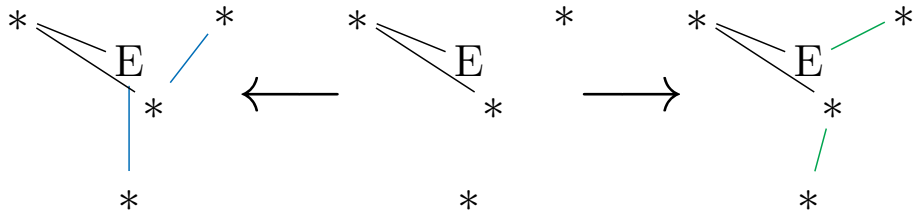


Figure 7: Reindeer jump where a reindeer leaps over its neighbour reindeer to a vacant rock two places ahead.

The directionality underlying the game (i.e., some reindeer are supposed to jump only from left to right (resp. right to left)) was ignored while deriving the solution that is presented here because the graphs used to model the game are *undirected* graphs. There is no notion of left–right (or backwards–forwards) in undirected graphs. Although one may argue that the graph has an `S`-end and a `T`-end, utilising them to encore directionality into the rules for reindeer moves would result in the loss-of-generality of the context for the rules. The rules would need to specify all 9(=1+3+1+3+1) vertices of the graph. The smallest number of moves required is 15. Both the solutions that were obtained for the reindeer game were symmetric, as depicted in Figure 8. The solutions are essentially mirror images of each other due to the underlying symmetry in the arrangement of the reindeer. One solution can be transformed to the other by exchanging the `B` and `R` labels on the reindeer.

`Objective value (integral): 16`

Vertex/Graph	In	Out
g_{0}	1	0
g_{1}	0	1

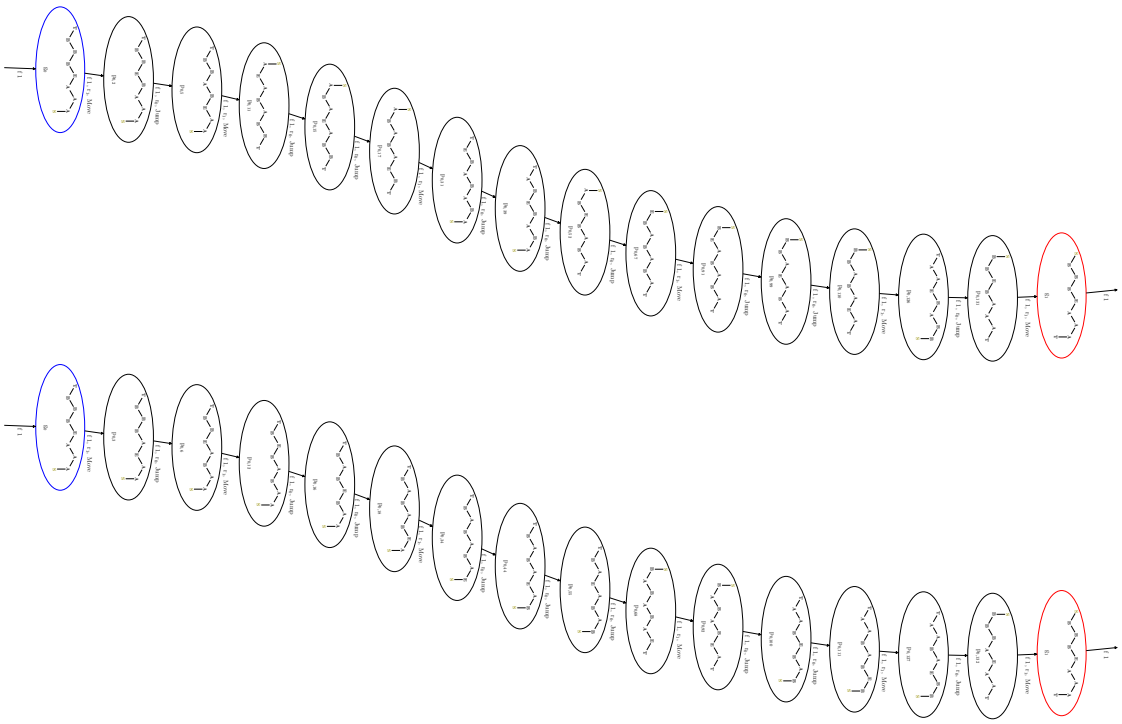


Figure 8: Sequence of moves leading to the transpose of the three trios of reindeer.

3 Catalan Game

3.1 Depiction of the Rules as provided by MØD

The name of the rules were provided in `rules.py` giving an indication of what each rule should do. A single move in the Catalan game in modelled by multiple graph grammar rules. The rules were specified as graph transformation rules as:

- markForConversion.gml was already defined.
- removeInterREdge.gml

```
rule [
    ruleID "Remove Inter R-Node Edges"
        left [
            edge[source 0 target 1 label "-"]
        ]
        context [
            node[id 0 label "R"]
            node[id 1 label "R"]
        ]
        right [
        ]
]
```

- reattachExternalEdge.gml

```
rule [
    ruleID "Reattach External Edges"
        left [
            edge[source 0 target 1 label "-"]
        ]
        context [
            node[id 0 label "0"]
```

```
            node[id 1 label "R"]
            node[id 2 label "A"]
            edge[source 1 target 2 label "-"]
        ]
        right [
            edge[source 0 target 2 label "-"]
        ]
]
```

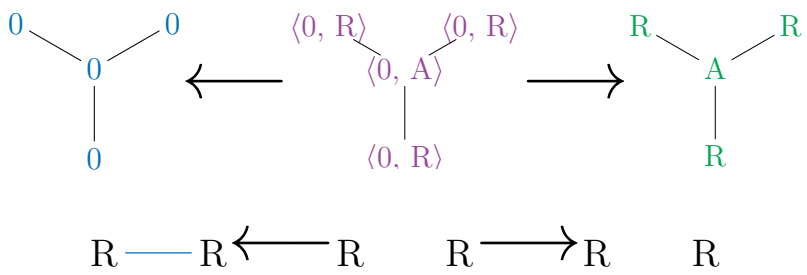
- removeAlreadyAttached.gml

```
rule [
    ruleID "Remove attached edge"
        left [
            edge [source 0 target 1 label "-"]
        ]
        context [
            node [ id 0 label "R"]
            node [ id 1 label "0"]
        ]
        right [
        ]
]
```

- removeSingleR.gml

```
rule [
    ruleID "Remove R Nodes"
        left [
            node[id 0 label "A"]
            node[id 1 label "R"]
            node[id 2 label "R"]
            node[id 3 label "R"]
            edge[source 0 target 1 label "-"]
            edge[source 0 target 2 label "-"]
            edge[source 0 target 3 label "-"]
        ]
        context [
        ]
        right [
            node[id 0 label "0"]
        ]
]
```

3.2 Mimicking a Catalan Move



A single Catalan move removes the three neighbours and the edges incident on them for a vertex with *exactly* three neighbours. This is accomplished by the defined graph transformation rules in the following manner:

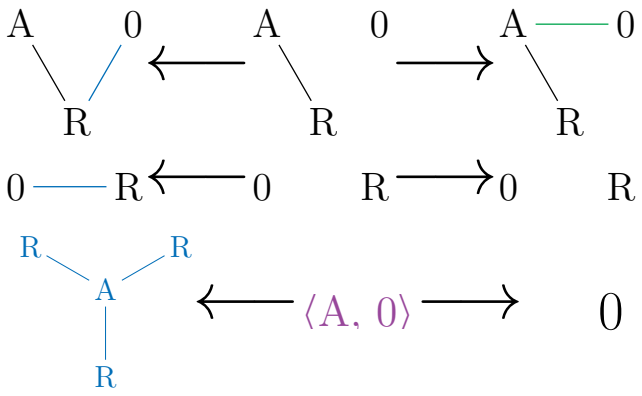


Figure 9: Graph transformation rules minicking a single move of the Catalan game.

1. `markForConversion.gml` identifies a vertex with exactly three neighbours, with the constraint defined by `constrainAdj`. The central vertex (with exactly three neighbours) is marked `A`, while its three neighbours are marked `R` (to be removed).

2. `removeInterR.gml` removes any edges with a vertex marked `R` as the source and a vertex marked `R` as the target.

3. `reattachExternalEdge.gml` removes an edge incident on a vertex with label `R`, and the other vertex of the edge having label `0`, and adds an edge between that vertex labelled `0` and the vertex labelled `A`, 'reattaching' the rest of the graph with the vertex labelled `A`.

4. `removeAlreadyAttached.gml` removes an edge between a vertex labelled `R` and a vertex labelled `0` from loops of the type `A---R---0---A`, which might have been formed after the application of the rule in `reattachExternalEdge.gml`.

5. `removeSingleR.gml` removes the vertices labelled `R` and the edges with a vertex labelled `R` as the source and a vertex labelled `A` as the target, and relabels the vertex marked `A` to `0` so that the next move of the Catalan game might be implemented.

The strategy for solving the Catalan game was

```
strategy = (addSubset(level) >> repeat[steps](
    mark >> repeat(revive(removeInterR))
        >> repeat(revive(reattachExternal))
        >> repeat(revive(removeAttached))
        >> removeR_unmarkA))
```

The rule named `mark` had a constraint defined, that is, it would be applied only on a vertex labelled `0` with exactly three neighbouring vertices with labels `0`. None of the three neighbouring vertices should have a prior marked node as a neighbour. The `removeInterR` rule removes any edges present between the three neighbouring vertices to be collapsed, so that there are no loops in the collapsed graph. The `reattachExternal` rule connects the remaining external part of the graph (incident edges from the rest of the graph on the nodes to be removed) to the central node. This ensures that only the three neighbours of the central node are removed and preserves the number of connected components in the graph to *one*.

In the process of reattaching external nodes in the graph to the central node, one might encounter loops of three vertices of the type `A---R---0---A`. This would happen when an external vertex labelled `0` is connected to two or more vertices to be removed (labelled `R`). In such cases, the application of the rule `reattachExternal` would fail because the left hand side would not match (an isomorphism, *exact match* is required rather than a *subgraph* isomorphism) the graph present. To remove an 'external´ edge `R---0` in these cases, the rule `removeAlreadyAttached` is used. The rule `removeAlreadyAttached` removes the edge `R---0` from 3-loops formed after the application of the rule `reattachExternal`, thus isolating all nodes labelled `R`, such that they only have a single

edge with a vertex labelled `A`. The rule `removeSingleR` (contrary to it's name) removes the three neighbouring vertices (labelled `R`) of the vertex labelled `A`. It also resets the label of the vertex `A` to `0`, preserving the incident edges from all vertices, except the ones that were removed (labelled `R`). This entire sequence of moves is iterated for every 'move´ of the Catalan game.

3.3 Solving the Catalan Game

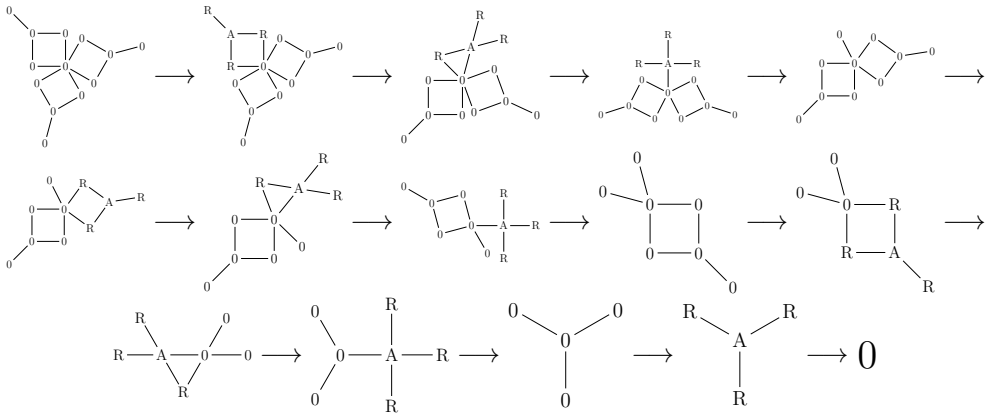


Figure 10: Sequence of moves solving level 1 of the Catalan game.

Level 1 of the Catalan game took 4 steps to converge. The graph transformation rules defined could solve all the levels of Catalan game provided, with a maximum of *ten* steps. All, except one (level 25), were solved within a reasonable time.

The `solve.sh` script for solving the different levels of the Catalan Game was modified so that the script tries to read the input graph for each level of the game and solves it by applying the strategy for a maximum of 10 iterations. All the levels of the game were solved in 10 iterations of the strategy.

```
#!/bin/bash
for levelNum in {1..56};
    do
        if test "x$levelNum" = "x"; then
                echo "No level number specified.";
                exit 1;
        fi;
        steps=10
        if test "x$steps" = "x"; then
                steps=1000000000
                echo "Be carefull, the number of steps is defaulted to a lot."
        fi;
        mod -e "levelFile = 'levels/level_$levelNum.gml'" -e "steps = $steps" -f doIt.py
        mv "summary/summary.pdf" "results/result_$levelNum.pdf"
done
```

Since PostMØD overwrites the `summary.pdf` files that are generated after solving each level of the Catalan game, the `summary.pdf` was moves to the `results` directory after appending the level number to it, so that the result of solving each level of the Catalan game was preserved.

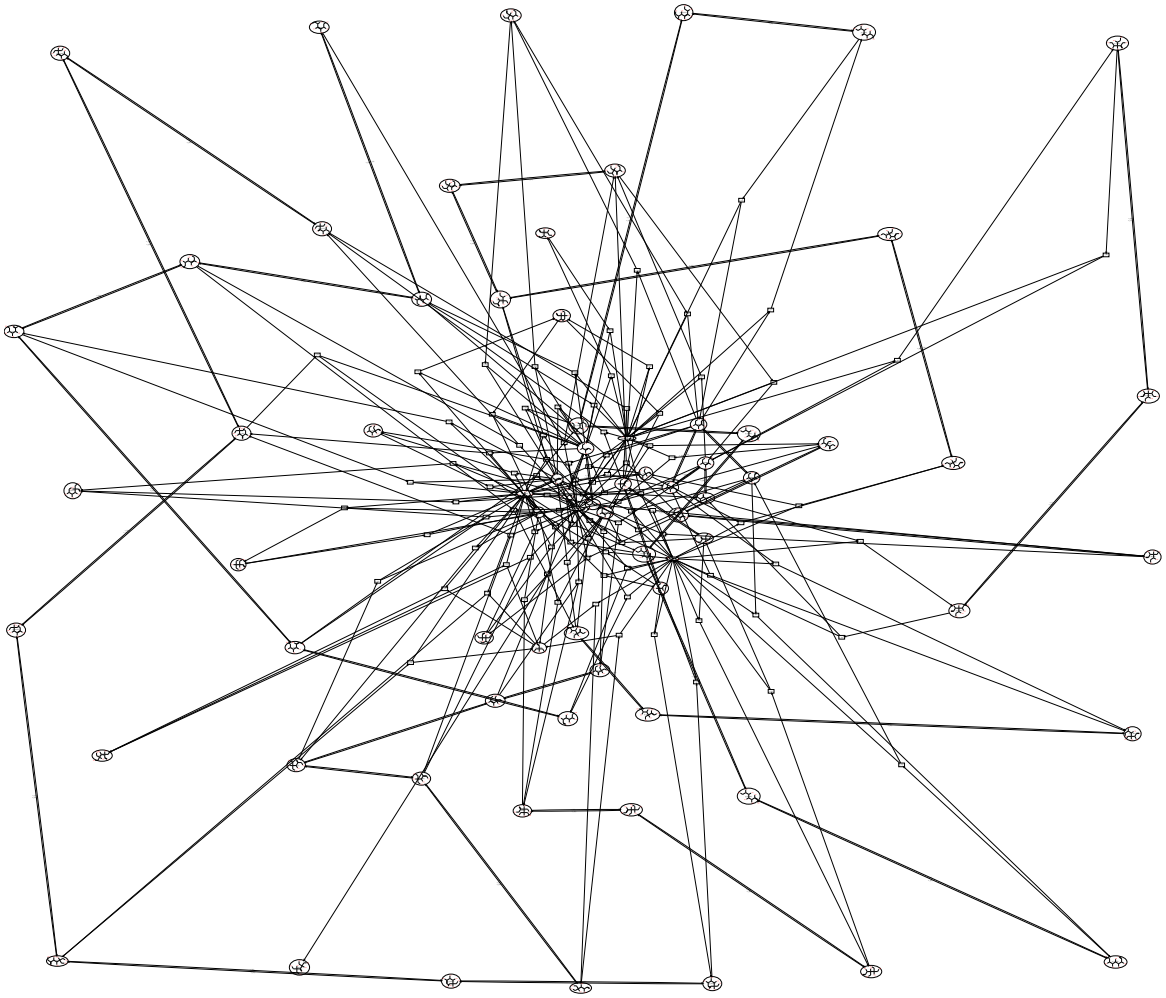


Figure 11: Caption