170025298 CS3052 Practical 2 11/05/2020

Polyreduction

Introduction

This practical mainly implemented several polyreduction programs. The program maps the positive instance of original format to the positive instance of the target format and vice versa. The polyreduction program receives four command-line arguments as the program input, and a fixed input and output format which has described in the README file and will also be described below.

Design

User Input

The polyreduction converts the program from one type to another, where the data structure could be changed such as polyreduce an adjacency matrix (the format of the graph colouring) to a cnf format output.

By consideration, the program should receive four command-line arguments as the user input, the usage of the program should be:

java Main [Input filepath] [Original type] [Target type] [output filepath]

Where the input and output path should have a fixed format that suits the original type and target type, the Dimacs CNF Format, which is introduced by CS3105, is the chosen format that is the input of the program. Dimacs format is described in the readme file, and related slides are posted on the reference, where the key advantage of this format is that Dimacs allows comment by using the keyword. It will enable multiple cnf expressions to be read from the file by the property of the expression.

Data Structure

The property of a cnf expression includes variables and clauses. As a result, the design of the cnf class should include variables, clauses, number of variables and number of clauses. Clauses can be saved with List<List<Integer>>, where the outer list saves a list of clauses and inner list saves literals in the clause. Since literals are integers from 1 to number of variables, so we need to save the number of variable and number of clauses for quick look up.

The property of a graph colouring structure includes a collection of edges, vertices,

and colours. The use of adjacency matrix is a way of storing edges and vertices. Adjacency matrix is a 2-dimensional int array, where the index represents the collection of vertex and value of array[index][index] represents whether two vertices are connected to each other. There are two important properties for adjacency matrix: the main diagonal is always 0, a undirected graph always transferred to a symmetric matrix, where array[index1][index2] is always equal to array[index2][index1].

Polyreduction

Polyreduction maps the positive instance of each problem into positive instance of target problem and maps the negative instance of the problem into negative instance of target problem.

SAT -> 3SAT
The instance of a SAT problem may contain clauses with more than 3 literals, mapping from SAT to 3SAT requires pick clauses with more than 3 literals and split the clauses until all clauses have no more than 3 elements. Algorithm for splitting a clause is for a CNF format expression with a single clause [1, 2, 3, 4] , we introduce a new variable 5 such that the clause is split into [1, 2, 5] [-5, 3, 4]. Hence the instance of the SAT problem is passed to the instance of 3SAT.

3SAT -> Colour Graphing
From the algorithm introduced from slides last year, mapping from the instance of 3SAT problem into the instance of the graph colouring problem introduces vertices and edges as following:
[x1 … xn] where each x connected to -x
[-x1 … -xn]
[y1 … yn] where yi is joined to xj and -xj if (i != j) and y connected to any other y except himself.
[C1 … Ck] where Ci is joined to every literals xj and -xj which is not in clause i
Such that we can construct an (3n+k)×(3n+k) adjacency matrix which is n+1 colourable.

Colour Graphing -> SAT
For polyreduce the instance of Graph Colouring to the instance of SAT, vertex constraint approach and edge constraint approach should be introduced. Vertex constrained approach said for each vertex, it is coloured by one of the k-colours. That is, for each vertex, it will choose the colour from the conjunction of k-colours.

$$v_{ic} = (v_{i1} \vee v_{i2} \vee \ldots \vee v_{ik})$$

Since we cannot assign two colours to a single node, so we have to make sure that there is no more colours except the chosen colour is assigned to the vertex, which is:

$$\{\neg y_{i,j}, \neg y_{i,j'}\}$$

Edge constraint approach described that for any two vertices connected by an edge, colours are always not the same. So for each edges (u, v) where u and v are two vertices of the edge, it will not assigned to the same colour k.

$$e_j = \neg(u_1 \wedge v_1) \wedge \neg(u_2 \wedge v_2) \wedge \ldots \wedge \neg(u_k \wedge v_k)$$

By the conjunction of vertex constraint approach and edge constraint approach, the satisfiability of the CNF format is the result of the k-colourable problem.

Implementation

Read Input

The input implementation includes the implementation of reading from command-line arguments and reading from the input file. Reading from command-line argument need to check the size of the argument. If the size does not satisfy the requirement, the program will print the usage of the program, or the program will assign values to the specific class by reading from files by using the scanner.

To read a CNF file, the file should have the keyword “cnf” or “gc” after the identifier “p”. By using the switch function, the program will then read the next two ints as the property of "cnf" or "gc". Finally, the program will continuously scan the next int with specific format until the end of the file.

Polyreduction

SAT -> 3SAT

To map the instance of SAT to the instance of 3SAT, what is needed is to split the clause to make sure all clauses have three literals. To achieve the goal, a split method is implemented to cut the clause recursively. The function will split the clause that is greater than three into two clauses and check the former clause since the first clause always has literals less than 3.

Graph coloring -> SAT

By the algorithm introduced above, vertex constrains approach and edge constrain approach need to be implemented. By using the addAll() function, both results of approaches are added into the clause, where different clauses in the list represent the conjunction so we can directly append two constrains together to get the result.

3SAT -> Graph Colouring follows the algorithm described in the design part and is hard coded.

Print to file:
Print to file function uses file printer to print the adjacency matrix or cnf expression to the file. The printer will print the first line, which is fixed with keyword “p” and type and two property integers. Then the printer will print the array or the list line by line but is not space sensitive. Finally, the program will close the printer.

Testing

1. SAT -> 3SAT:

```
c    clause length = 3
c
p cnf 9 1
1 2 3 4 5 6 7 8 9 0
```

=>

```
p cnf 15 7
1 2 10 0
-10 3 11 0
-11 4 12 0
-12 5 13 0
-13 6 14 0
-14 7 15 0
-15 8 9 0
```

A cnf with 9 variables is spitted into 7 clauses, which passed the test by hand calculation. As the result, the polyreduction from SAT to 3SAT passed.

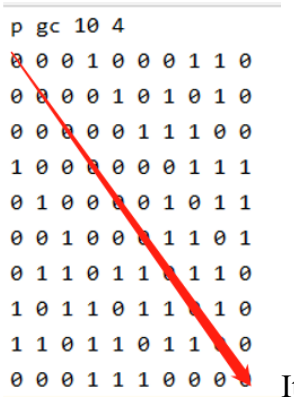
2. 3SAT -> Graph coloring:

```
p cnf 3 1
1 2 3 0
```

=>

```
p gc 10 4
0 0 0 1 0 0 0 1 1 0
0 0 0 0 1 0 1 0 1 0
0 0 0 0 0 1 1 1 0 0
1 0 0 0 0 0 0 1 1 1
0 1 0 0 0 0 1 0 1 1
0 0 1 0 0 0 1 1 0 1
0 1 1 0 1 1 0 1 1 0
1 0 1 1 0 1 1 0 1 0
1 1 0 1 1 0 1 1 0 0
0 0 0 1 1 1 0 0 0 0
```

Since the main diagonal is 0 and the clause is symmetric, so the adjacency has a correct format of reduction. Then we will check the colourable question. Since the SAT is always satisfiable, we want to check it is 4-colorable. By looking at the lower part of the graph:



It is easy to find that it is 4-colorable by counting the nearest node is always less than 4. So, the test passed.

3. Graph colouring -> SAT:
Not-colourable testing:
For an obviously no-colourable question, by putting it into WALKSAT program implemented in CS3105, it reached to a time out, which shows that this is not colourable.

```
c Test clause - Dimacs CNF format
c
p gc 5 3
0 1 1 1 1
1 0 1 1 1
1 1 0 1 1
1 1 1 0 1
1 1 1 1 0
```

=>

```
Timed out!
- UNKNOWN -
Process ends in 10004 ms.
Thread-0 is called: Thanks for using!
```

Obviously colourable testing:

```
c Test clause - Dimacs CNF format
c
p gc 5 6
0 1 1 1 1
1 0 1 1 1
1 1 0 1 1
1 1 1 0 1
1 1 1 1 0
```

=>

```
- SATISFIABLE -
Total looping: 15
result:
0: [false true ]
1: [false true ]
2: [true true ]
3: [false true ]
4: [true true ]
```

By putting the second testing into the WalkSAT, it passed obviously with a quite small number of looping.

Smaller colourable:
When number of colours is set to 5, it reaches the result:

```
- SATISFIABLE -
Total looping: 208
result:
0: [false true ]
1: [false true ]
2: [true true ]
```

4. Object transfer testing:
This test will test an simple Satisfied SAT transferred into 3SAT, and then transferred to Graph colouring and transfer to SAT.
The original SAT is a conjunction from 1 to 10

```
p cnf 10 1
1 2 3 4 5 6 7 8 9 10 0
```

3SAT:

```
p cnf 17 8
1 2 11 0
-11 3 12 0
-12 4 13 0
-13 5 14 0
-14 6 15 0
-15 7 16 0
-16 8 17 0
-17 9 10 0
```

Graph colouring(Command line output):

```
-----------------
Properties: Vertex
Colors: 11
vertices: 31
Current adjacency: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1,
--------end-------
```

SAT:

```
-----------------
Properties: CNF
Original variable range: [1,341]
New variable range: No new var introduced!
Current clause: [[-2, -1], [-3, -1], [-3, -2], [-4, -1], [-4, -2], [-4, -3], [-5, -1], [-5, -2], [-5, -3], [-5, -4], [-6, -1], [-6, -2], [-6, -3], [-6, -4], [-6, -5], [-7, -1], [-7, -2], [-7, -3
--------end--------
```

As a result, they can be passed to each other. Unluckily, whether the result is satisfiable is not testable as there are 170681 Lines of clauses. But I believe it is satisfiable!

Since three tests are all passed, hence the test passed.

Analysis:

From the last testing above, we can find that the solution to an instance of any of these problems, can be solved in polynomial time.

For poly reduce from SAT to 3SAT, the increase of clauses and number of vars are polynomial, where a clause of length more massive than three can be spat into k-2 clauses.

For poly reduce from 3SAT to Graph Colouring, the increase from the number of variables and number of clauses to the number of vertices and number of colours are polynomial too, as the number of vertices is equal to the sum of the number of variables times three plus number of clauses. Hence this poly reduce is polynomial too.

For poly reduction from Graph Colouring to 3SAT, the increase from the number of vertices and number of colours to the number of clauses and number of literals are polynomial since the number of literals is the number of vertices times number of colours. The number of clauses is equal to the number of colours plus the number of edges times number of colours. Hence the reduction is polynomial.

Extension:

Polyreduce from SAT to Clique:

Design:
Clique is the reformat of the CNF. Clique is a graph consists of literals from n different clauses, where it does not have specific literals in the clique but just set of positions [i, j] considered as literals. It constructs a graph with vertices and edges where edges are built by vertices that in different clauses and different literals.

Implementation:
Clique is an abstract graph with a set of vertices and edges. The implementation of clique is abstractly implemented same as design, but messages are printed only inside the commander but not prints to a file.

Polyreduce from SAT to Clique to Vertex Cover

Design:
The Vertex Cover edges the clique where there are no edges. By adding a new data structure where stores two vertices connected to each other but not stored in edges list, since no edges are in a clique, so this is a vertex cover.

Implementation:
Simply add a new data structure when reading from CNF and print the data to the standard output.

Conclusion

This practical mainly explored the poly reduction, which is helped to study the concept and the implementation of the poly reduction. By implementing the program, I have learnt a lot on logic and literal index calculation (see the graph colouring part).

What should be improved is that the logic of designing this program still need to be improved, as programming language should be chosen carefully and using tuples is precisely a right way if the application uses the python as a programming language.

Code Instruction:

The use of my software is IntelliJ, where the java code can be run with IntelliJ or productions in out path. Usage for the program is in the readme file.

Reference:

Reference for algorithms are in the program interfaces with Javadoc.

Dimacs CNF format
https://studres.cs.st-andrews.ac.uk/CS3105/Practicals/CS3105%20AI%20Search%20Practical%202/dimacs-cnf.pdf

Polynomial 3-SAT Encoding for K-Colorability of Graph
http://cse.vnit.ac.in/people/narendraschaudhari/wp-content/uploads/sites/15/2014/10/046-3-SAT-encoding-of-K-Colorability-encc004.pdf