Internship Report

Vipul Garg
Mentor : Professor Abdallah Saffidine

May - July 2022

1 Introduction

It is always a topic of interest to check if a particular problem has a polynomial time algorithm. But if we have not been able to find a polynomial time algorithm for a particular problem, it doesn't imply that a polynomial time algorithm doesn't exist. It is widely believed that P≠NP. If it is true, we would not be able to find a polynomial time algorithm for some problems ever. In that case, it is essential to be able to prove that a problem does not have a polynomial time algorithm. Reduction[2] is a tool that helps in identifying NP-Complete problems. If we are able to identify if a problem is NP-Complete, we can redirect our research work more towards finding approximate algorithms for the problem instead of polynomial time algorithms.
In our research work, we present implementations of some reductions in Answer Set Programming(ASP)[1] and a verification process that can verify the correctness of reductions. We also test the efficiency of this verification process and discuss areas where more work could be done.
One would be able to provide an instance of the input problem of these reductions and obtain the reduced instance of the output problem. One would also be able to check the correctness of any implemented reduction.

2 NP-completeness

In complexity theory, a problem is NP-complete when it is both in NP and NP-hard. A problem is in NP if the correctness of each solution of the problem can be verified in polynomial time and the problem can be solved by a brute-force algorithm by trying all possible solutions. A problem is NP-hard if it is at least as hard as the hardest problems in NP. Precisely, every problem in NP can be *reduced* to the given NP-hard problem in polynomial time. Consequently, using an algorithm to solve the NP-Hard problem, we can solve every problem in NP. Hence, NP-complete represents the hardest problems in NP. If some NP-complete problem has a polynomial time algorithm, all problems in NP have a polynomial time algorithm. To prove that a problem is NP-complete, we show that it is in NP and that every problem in NP can be *reduced* to the given

problem. Since it is difficult to show that every problem in NP can be reduced to the given problem, we instead show that a *known* NP-complete problem can be reduced to the given problem. In 1973, Cook showed that the Boolean Satisfiability Problem [3] is NP-Complete. Specifically, the 3-SAT [4] problem is NP-complete We can use this fact and the transitivity property of reductions to show the NP-completeness of more problems.

3 The Notion of Reduction

If problem Y is reducible to problem X, it means that if we have an algorithm to solve X, the same algorithm can used to solve Y. As a result, problem X is at least as hard as problem Y. If we are able to find a polynomial time algorithm to solve X, we would be able to find a polynomial time algorithm to solve Y as well.
For decision problems, reduction(Karp)[5] of problem Y to problem X means to create a mapping from instances of problem Y to instances of problem X such that :-

1. Every instance of problem Y has an image.
2. The image of an instance of problem Y should be attainable in polynomial time.
3. Every 'yes' instance of problem Y is mapped to a 'yes' instance of X.
4. Every 'no' instance of problem Y is mapped to a 'no' instance of X.

We implement a selection of NP-complete reductions in a logic-based programming language Clingo and further test the correctness of these reductions using an automatic reduction verification process.

4 Reductions in ASP

Answer Set Programming(ASP) is a form of declarative programming which is based on the stable model semantics of logic programming. An answer set program is just knowledge written in the form of Prolog style facts and rules[6]. The search for a stable model that satisfies the knowledge is done by answer set solvers. The ASP tool used to implement the reuctions is Clingo.

4.1 Defining a Problem

In ASP, to define a problem, we must define the set of rules that the solution of the problem must satisfy. One such example is that of 3SAT:

specifications > 3-Sat.lp

```
1  %Specification
2  bool(true;false).
3  ns(sat3,basis(var)).
4  ns(sat3,candidate(set(V,B))) :- ns(sat3,base(var,V)) , bool(B).
5  ns(sat3,satisfied(pos(V))) :- ns(sat3,base(var,V)) , ns(sat3,solution(set(V,true))).
6  ns(sat3,satisfied(neg(V))) :- ns(sat3,base(var,V)) , ns(sat3,solution(set(V,false))).
7
8  ns(sat3,assigned(V)) :- ns(sat3,solution(set(V,_))).
9  ns(sat3,incorrect) :- ns(sat3,base(var,V)) , not ns(sat3,assigned(V)).
10 ns(sat3,incorrect) :- ns(sat3,solution(set(V,B1))) , ns(sat3,solution(set(V,B2))) , B1 != B2.
11 ns(sat3,incorrect) :- ns(sat3,solution(X)) , not ns(sat3,candidate(X)).
12 ns(sat3,incorrect) :- ns(sat3,instance(clause(X,Y,Z))) , not ns(sat3,satisfied(X)) ,not ns(sat3,satisfied(Y)) , not ns(sat3,satisfied(Z)).
13 ns(sat3,correct) :- not ns(sat3,incorrect).
```

- ns/2 is predicate that is used throughout all programs. The first argument is the name of the problem a particular fact belongs to while the second argument is the fact itself.
- candidate/1 is a predicate that contains a fact that is *potentially* the part of solution of any instance of that problem. In the case of 3SAT, the fact is of the form set(V,B), which is true if variable V has been assigned the boolean value B.
- solution/1 is a predicate that contains a fact that is part of solution of an instance of that problem.
- instance/1 is a predicate that contains a fact that is part of a particular instance of that problem.
- In lines 9-12, we check if a particular solution is a valid certificate for a particular instance. In case we don't find the solution to be incorrect, we conclude it to be correct through line 13.

In similar fashion, many decision problems can be defined in ASP.

4.2 Defining a Reduction

Reductions in ASP can be written by deriving facts for the second problem using the facts of the instance of the first problem. Algorithm to convert certificates from one problem to another can be done in similar way. One such reduction is 3SAT-4SAT:

Reductions > 3SAT-4SAT.lp

```
1
2  %Creating reduced instance of 4SAT. Simple enough reduction.
3  ns(sat4,base(var,X)) :- ns(sat3,base(var,X)).
4  ns(sat4,instance(clause(X,Y,Z,X))) :- ns(sat3,instance(clause(X,Y,Z))).
5  %%REDUCTION COMPLETE
6
7  % Solution of 4 SAT INSTANCE IN TERMS OF 3 SAT AND VICE VERSA
8  ns(sat4,solution(set(V,B))) :- ns(sat3,solution(set(V,B))), direct.
9
10 ns(sat3,solution(set(V,B))) :- ns(sat4,solution(set(V,B))), not direct.
11
12
13 input(sat3).
14 output(sat4).
```

- In lines 2 and 3, we find the set of variables and clauses for the 4SAT problem to create the reduced instance.
- In line 8, we write the method to find a valid certificate of the reduced instance in terms of the certificate of the input instance. The fact 'direct' signifies that the conversion is from input to output problem.
- In line 10, we write the method to find a valid certificate of the pre-image instance in terms of the certificate of the output instance. The fact 'not direct' signifies that the conversion is from output to input problem.
- In lines 13 and 14, we let the verifier know that the input problem was 3SAT and the output problem was 4SAT.

In similar fashion, multiple reductions were implemented in ASP.

5 Implemented Reductions

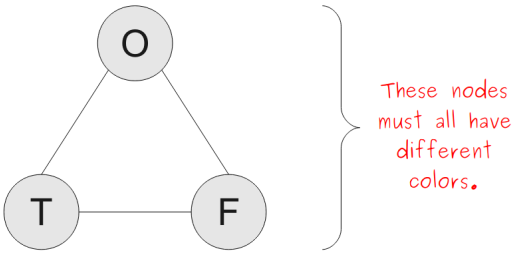
5.1 3SAT to 4SAT

We consider a 3SAT instance having n **variables** and m **clauses**.
Let a clause of the 3SAT instance be (a $\vee$ b $\vee$ c). We produce a clause of the 4SAT instance by simply repeating the first literal in the clause to get the clause : (a $\vee$ b $\vee$ c $\vee$ a). Each clause of the 3SAT is replicated in similar fashion to obtain the 4 SAT instance which is the image of the given 3SAT instance.
Size of the 4SAT Instance: n variables and m clauses.

5.2 3SAT to 3COLOR

The reduction has been taken from CS103 course material of the Stanford University [7].
We consider a 3SAT instance having n **variables** and m **clauses**.
We create 3 types of gadgets(pages : 25-27).

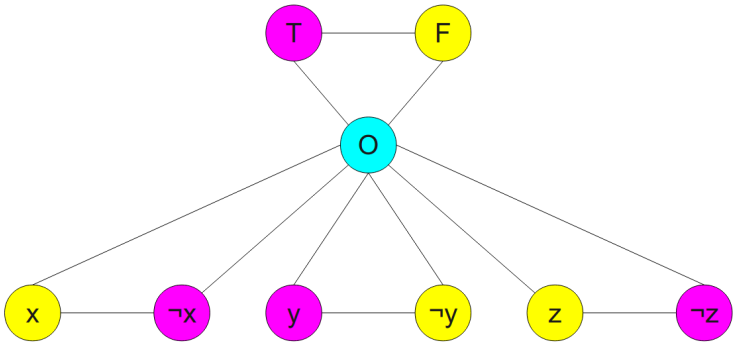
Gadget One: Assigning Meanings

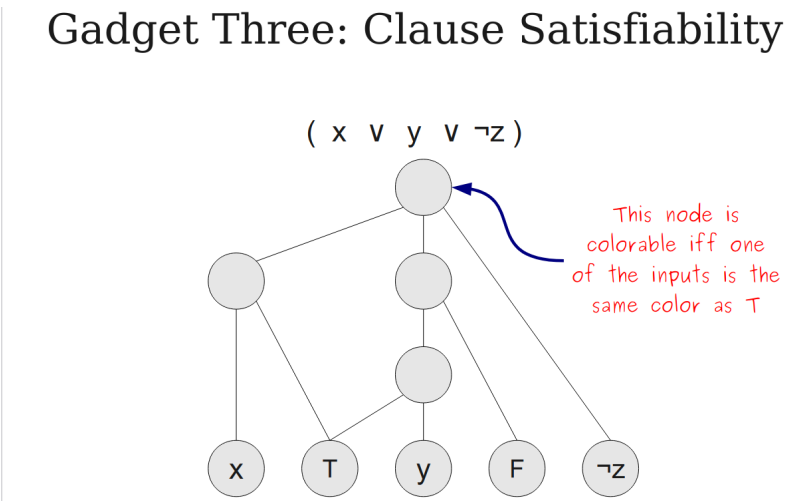


The color assigned to T will be interpreted as "true."
The color assigned to F will be interpreted as "false."
We do not associate any special meaning with O.

Gadget Two: Forcing a Choice

(x ∨ y ∨ ¬z) ∧ (¬x ∨ ¬y ∨ z) ∧ (¬x ∨ y ∨ ¬z)



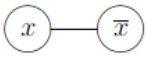


The graph obtained by combing all the 3 gadgets is the image of the 3SAT instance.
Size of the 3COLOR Instance: $3+2n+4m$ vertices and $3+3n+9m$ edges.

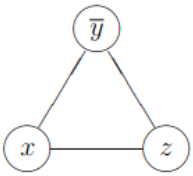
5.3 3SAT to Vertex Cover

The reduction has been taken from CSCI 4602 course material of the East Carolina University [8].
We consider a 3SAT instance having n **variables** and m **clauses**.
We create 2 types of gadgets(page : 5).

1. Vertex Gadget:



2. Clause Gadget:



To complete the graph, we connect vertices of gadget 1 with vertices of gadget 2 having the same label. In this graph, we need to find a vertex cover of at least k vertices where $k = n + 2m$. The graph and the value of k is the image of the 3SAT instance.

Size of the Vertex Cover Instance: $2n+3m$ vertices and $n+6m$ edges and minimum size of vertex cover $= n+2m$

5.4 3SAT to Independent Set

The reduction has been taken from course material of CS 374 of the University of Illnois[9].
We consider a 3SAT instance having n **variables** and m **clauses**.
We create clause gadgets by connecting all the 3 literals in a clause to form a triangle in the following way :

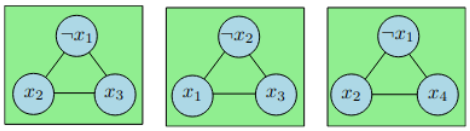


Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

To complete the graph, we connect 2 vertices if they label complementary literals. In this graph, we need to find an independent set of at least k vertices where $k = m$. The graph and the value of k is the image of the 3SAT instance. The number of edges in this graph depend on the number of pairs of complementary literals in the CNF. The number of edges would always be greater than or equal to $3m$ since every clause gadget would contribute to 3 edges. It can also be shown that the number of edges are always less than or equal to $3m + \frac{3m^2}{4}$.
Size of the Independent Set Instance: $3m$ vertices and m' edges where $3m \leq m' \leq 3m + \frac{3m^2}{4}$and minimum size of independent set $= m$.

6 Reduction Verification Process

To aid in our reduction verification process, we bound the size of the instances of problem A and our assumption is if there is a bug in the reduction, the bug should appear in the reduction of these bounded size instances.
To check if a reduction from problem A to problem B is correct, we try to find counter examples to prove that it isn't correct. If we are not able to find a counter example, we conclude that the reduction is indeed correct. A valid counter example is one in which a 'yes' instance of problem A is mapped to a 'no' instance of problem B or a 'no' instance of problem A is mapped to a 'yes' instance of problem B.
One way to proceed is to take all possible instances(bounded) of problem A and their corresponding images. Lets take an instance of problem A and its image instance. If A is a 'yes' instance, finding a counter example is same as checking if there exists a candidate in solution space of instance A such that it is a solution of problem A while at the same time; all candidate solutions in solution space of the image instance are invalid candidates. To do this, we would need to implement the nested quantifier-

if there exists ... such that for all ...

This is something that is not possible to implement in ASP without using disjunctive logic program[10]. Hence, this method of verification runs into trouble. But we can implement a variant of this method by asking for some more input from the user in addition to the reduction algorithm. This would also help us in creating a more efficient reduction verification process.
In addition to the reduction, the reduction verifier also requires the user to input an algorithm :-

1. to convert a valid certificate of a 'yes' instance of problem A into a valid certificate of the reduced instance of problem B

2. to convert a valid certificate of a 'yes' instance of problem B in the range set into a valid certificate of the pre-image instance of problem A.

To find counter examples of the first type, we select all the 'yes' instances among the bounded size instances of problem A through **brute-force**. Our ASP tool(Clingo) is able to find patterns in the data to eliminate a lot of repetitive cases which effectively increases the speed of the brute-force process. The verifier would check if the instance obtained by reducing the given 'yes' instance is a 'yes' instance. To do so, it would create a certificate for the reduced instance using the algorithm provided by the user and check if that is indeed a valid certificate. If we are to find a counter example, then there would exist a 'yes' instance of problem A whose image would have no valid certificate.
To find counter examples of the second type, we select all the 'yes' instances in the range set of the mapping through **brute-force**. Again, the process is sped up due to faster brute-force search of Clingo. The verfier would check if the pre-image of this reduced instance is a 'yes' instance. To do so, it would create a certificate for the pre-image using the algorithm provided by the user and check if that is indeed a valid certificate. If we are to find a counter example, then there would exist a 'yes' instance in range set whose pre-image would have no valid certificate. This relies on the idea of contra-positive.

7 Testing and Results

Different bounds were taken on the input problem size for different reductions and the time taken for the verifier to run for each case was recorded in table in the tables given below. 3-SAT was taken as the input problem for each reduction. The reductions and the verification process were written in Clingo.

Target	Instance Size		Sol space	$n = 2, m \leq 6$		$n = 2$	
	$n' =$	m'		Y→N	N→Y	Y→N	N→Y
4SAT	n	m	2^n	0.00s	0.00s	0.00s	0.00s
3COL	$3 + 2n + 4m$	$3 + 3n + 9m$	$3^{n+2n+4m}$	0.00s	0.00s	0.00s	0.00s
VC	$2n + 3m$	$n + 6m$	2^{2n+3m}	6.40s	22.68s	22.63s	895.61s
IS	$3m$	$[3m, 3m + \frac{3m^2}{4}]$	2^{3m}	2.51s	6.96s	11.05s	195.44s

Target	$n = 3, m = 1$		$n = 3, m = 2$		$n = 3, m = 3$		$n = 3, m = 4$		$n = 3, m \leq 4$	
	Y→N	N→Y	Y→N	N→Y	Y→N	N→Y	Y→N	N→Y	Y→N	N→Y
VC	0.28s	0.02s	4.72s	0.78s	86.17s	39.46s	962.46s	444.64s	1075.25s	890.48s
IS	0.04s	0.01s	1.20s	0.18s	14.74s	4.68s	118.41s	77.03s	232.14s	191.40s

Target	$n = 10$		$n = 25$		$n = 50$		$n = 75$		$n = 100$	
	Y→N	N→Y	Y→N	N→Y	Y→N	N→Y	Y→N	N→Y	Y →N	N→Y
4SAT	0.05s	0.05s	0.53s	0.57s	4.93s	5.13s	18.75s	19.31s	49.15s	49.29s

Target	$n = 3$		$n = 5$		$n = 7$		$n = 9$		$n = 10$	
	Y→N	N→Y	Y→N	N→Y	Y→N	N→Y	Y→N	N→Y	Y →N	N→Y
3COL	0.04s	0.00s	1.06s	0.05s	10.17s	0.52s	79.64s	3.05s	175.97s	9.42s

8 Conclusion

The reduction from 3SAT to 4SAT was relatively simple and hence the verifier was able to quickly verify reductions up to even a very large size of the 3SAT instance.

The reduction from 3SAT to 3COL was verified at a much faster rate than reductions to Vertex Cover and Independent Set. This might come as a surprise because the former reduction has a larger size and the solution space of the reduced instance is significantly larger! This can be attributed to the larger complexity of the reduction algorithm from 3SAT to VC/IS. Another reason could be that vertex cover and independent set both have threshold values for the maximum and minimum sizes respectively for the size of their solution set of vertices which might increase the time required to check valid certificates for both of these problems. Further research can be done in this area.

It was seen in the above tests that the verifier is able to verify reductions faster when we break the verification process into smaller parts. This fact was supported by the following 2 observations-

1. In each case mentioned in the table above, the verifier ran faster if we verified the 'yes' to 'no' and 'no' to 'yes' cases separately instead of clubbing

the two together in a single program run. Time taken to verify 'yes' to 'no' cases + time taken to verify 'no' to 'yes' cases $\leq$ time taken to verify both of them in a single run of the verifier.

2. One can see in table 2 that the time taken to verify the case where $m \leq 4$ is greater than $\sum_{i=1}^{4} t(i)$ where $t(i)$ is the time taken to verify the case where $m = i$.

The number of reductions can be added and more analysis can be done for the time taken to verify these reductions. More patterns can emerge if we take a larger number of problems that have similar type of reductions. This can help in devising correct and optimal reductions of several problems.

References

[1] ASP

[2] Reductions

[3] Boolean Satisfiability Problem

[4] 3SAT

[5] Karp Reductions

[6] Prolog

[7] CS103 Stanford University

[8] CSCI 4602 East Carolina University

[9] CS374 University of Illnois

[10] Disjunctive Logic Programs versus Normal Logic Programs *by Heng Zhang, Yan Zhang*