Internship Report

Vipul Garg

May - July 2022

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 is NP-Complete. Specifically, the 3-SAT problem is NP-complete We can use this fact and the transitivity property of reductions to show the NP-completeness of more problems.

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) 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.

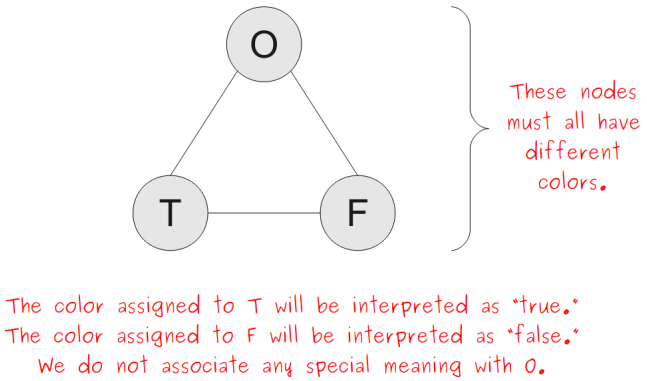
Implemented Reductions

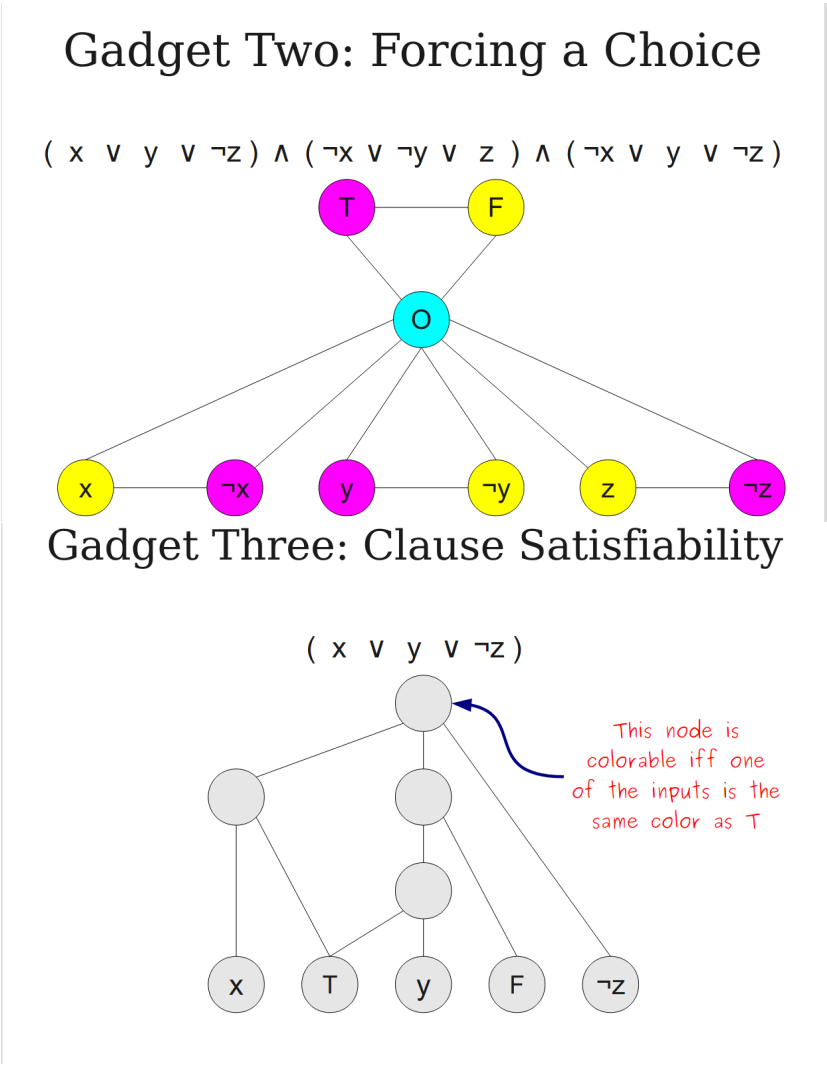
1 3SAT to 4SAT

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.

2 3SAT to 3COLOR

The reduction has been taken from CS103 course material of the Stanford University. Source : Small27.pdf.
We consider a 3SAT instance having n **variables** and m **clauses**.
We create 3 types of gadgets(pages : 25-27).



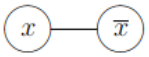


The graph obtained by combing all the 3 gadgets is the image of the 3SAT instance.
Size of the 3COLOR Instance: $3+2n+4m$ vertices and $3+3n+9m$ edges.

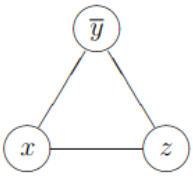
3 3SAT to Vertex Cover

The reduction has been taken from CSCI 4602 course material of the East Carolina University. Source : NPC-example.pdf.
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 $= n + 2m$

4 3SAT to Independent Set

The reduction has been taken from course material of CS 374 of the University of Illnois. Source : 23_2_0_0.pdf
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 :

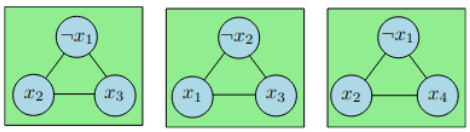


Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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 $= m$.

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. 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. The details of the additional input required are mentioned in the following paragraphs.
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. In addition to the reduction, the reduction verifier also requires an algorithm to convert a valid certificate of the 'yes' instance of problem A into a valid certificate of the reduced instance of problem B from the user. 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. In addition to the reduction, the verifier also requires an algorithm to convert a valid certificate of the 'yes' instance in the range set into a valid certificate of the pre-image. 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.
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 next section. 3-SAT was taken as the input problem for each reduction.

The reductions and the verification process were written in Clingo.

Results

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	$n = 3, m = 1$		$n = 3, m = 2$		$n = 3, m = 3$		$n = 3, m = 4$	
	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
IS	0.04s	0.01s	1.20s	0.18s	14.74s	4.68s	118.41s	77.03s

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