CS 514

Quiz 4 – Network Flow, Linear Programming, and NP Completeness

Answers to Question1 and Question2

Q1. To solve 3SAT **we need to determine whether a given 3-CNF formula has at least one satisfying assignment. For this problem we will consider a similar problem we call** Twice3SAT**. To solve** Twice3SAT **we need to determine whether a given** 3-CNF **formula has at least two distinct satisfying assignments.**

Show that Twice3SAT **is NP-Complete.**
(5 points) NP: Demonstrate that Twice3SAT **belongs to NP.**

(10 points) NP-Hard Reduction: Describe a reduction demonstrating that Twice3SAT **is NP-Hard.**

(5 points) Running Time: Demonstrate that your reduction runs in polynomial time.

Answer –

A. NP: Demonstrate that Twice3SAT belongs to NP (5 points):

To show that Twice3SAT is in NP, we need to prove that, given a potential solution, we can verify its correctness in polynomial time.

A solution to Twice3SAT is two distinct satisfying assignments for a given 3-CNF formula. To verify this solution, we can do the following:

1. For each clause in the 3-CNF formula, check if at least one of the assignments makes the clause true.
2. This checking process involves evaluating each clause, and for each clause, checking if it is satisfied by either of the two assignments.

Since the number of clauses is polynomial in the size of the input, and each clause can be checked in polynomial time, the overall verification process is polynomial. Thus, Twice3SAT is in NP.

Explanation in detail –

1. For each clause in the 3-CNF formula: A clause in 3-CNF consists of three literals (variables or their negations) joined by logical OR operations. For example, a clause might look like (x1 ∨ ¬ x2 ∨ x3).

2. Check if at least one of the assignments makes the clause true: For each clause, we evaluate the truth value by substituting the truth values of the variables according to the two distinct satisfying assignments. If, for at least one of the assignments, the clause evaluates to true, then the clause is satisfied.

3. Checking process for each clause involves evaluating each clause twice: For each clause, we need to perform the evaluation twice, once for each of the two distinct satisfying assignments. We evaluate the clause with the truth values assigned by the first satisfying assignment. Then, we evaluate the same clause with the truth values assigned by the second satisfying assignment.

4. Overall Time Complexity: The number of clauses in the 3-CNF formula is polynomial in the size of the input. For each clause, evaluating it twice (for the two assignments) can be done in polynomial time.

Example: Let's consider a 3-CNF formula with three clauses:

1. (x1 ∨ ¬ x2 ∨ x3)
2. (¬ x1 ∨ x2 ∨ ¬ x3)
3. (x2 ∨ ¬ x3 ∨ x4)

And let's assume two satisfying assignments:

1. x1 = True, x2 = False, x3 = True, x4 = True
2. x1 = False, x2 = True, x3 = False, x4 = True

Now, verify each clause:

1. For clause 1: (True ∨ True ∨ True)- Satisfied by the first assignment.
2. For clause 2: (False ∨ True ∨ False) - Not satisfied by the second assignment.
3. For clause 3: (False ∨ True ∨ True) - Satisfied by the second assignment.

At least one assignment satisfies each clause, so the formula is satisfied by both assignments.

The verification process involves polynomial-time operations, as it checks each clause in polynomial time. Therefore, Twice3SAT is in NP.

B. NP-Hard Reduction: Describe a reduction demonstrating that Twice3SAT is NP-Hard:

To demonstrate that Twice3SAT is NP-hard, we need to perform a polynomial-time reduction from a known NP-hard problem, such as SAT.

Reduction Steps:

Given an instance of SAT with variables x1, x2, …, xn and clauses C1, C2, …, Cm, we construct an instance of Twice3SAT:

1. Introduce new variables : For each original variable xi, we introduce two new variables yi and zi.
2. Replace clauses: We replace each clause Cj in SAT with the following three clauses in 3-CNF format:

- (xa ∨ ya ∨ za)
- (¬ xa ∨ ¬ ya)
- (¬ xa ∨ ¬ za)

Explanation of Reduction:

1. **Satisfiability in the original SAT instance:** If there is a satisfying assignment for the original SAT instance, we use it for the new Twice3SAT instance by setting yi and zi to True/False accordingly.
2. **Two distinct satisfying assignments in the original SAT instance:** If there are two distinct satisfying assignments for the original SAT instance, we use them for the new Twice3SAT instance by setting yi and zi to True/False accordingly.

Polynomial Time Complexity:

The construction involves a constant number of operations for each variable and each clause in the original SAT instance. Therefore, the reduction runs in polynomial time.

Explaining these two reductions in detail –

1. If there is a satisfying assignment for the original SAT instance: Suppose we have an instance of SAT with variables x1,x2, ..., xn and clauses C1, C2, ..., Cm and we know that there exists a satisfying assignment for this SAT instance.

Original Satisfying Assignment: Let's say the satisfying assignment is x1 = v1, x2 = v2, ... xn = vn, where each vi is either True or False.

Construction of Twice3SAT Assignment: For each variable xi in the original SAT instance, introduce two new variables yi and zi in the Twice3SAT instance. We set yi and zi to vi (True or False) according to the satisfying assignment of the original SAT instance.

The original SAT instance is satisfiable, and we've created a corresponding assignment for the new Twice3SAT instance.

2. Two distinct satisfying assignments in the original SAT instance:

Now, let's consider the scenario where there are two distinct satisfying assignments for the original SAT instance.

Distinct Satisfying Assignments: Suppose the two satisfying assignments are A and B such that A has x1 = a1, x2 = a2, ..., xn = an and B has x1 = b1, x2 = b2, ..., xn = bn where each ai and bi are True or False.

Construction of Twice3SAT Assignments: For each variable xi in the original SAT instance, introduce two new variables yi and zi in the Twice3SAT instance. Set yi and zi to ai for assignment A and bi for assignment B.

The original SAT instance has two distinct satisfying assignments, and we've created corresponding assignments for the new Twice3SAT instance.

We can conclude that the construction ensures that if the original SAT instance is satisfiable, the new Twice3SAT instance is also satisfiable, and if the original SAT instance has two distinct satisfying assignments, the new Twice3SAT instance has two distinct satisfying assignments. Therefore, we have a polynomial-time reduction from SAT to Twice3SAT, establishing that Twice3SAT is NP-hard. Since Twice3SAT is also in NP (as demonstrated in the first part), it is NP-complete.

C. Running Time: Demonstrate that your reduction runs in polynomial time :

The reduction from SAT to Twice3SAT is polynomial-time because each step of the reduction involves a constant number of operations, and the number of variables and clauses in the new instance is polynomial in the size of the original SAT instance.

Detailed Time Complexity Analysis:

Introducing new variables: For each of the n original variables, we introduce two new variables, resulting in 2n new variables. This operation is clearly polynomial in the size of the original SAT instance.

1. **For each of the n original variables:** We introduce two new variables (yi and zi) for each original variable (xi). So, for n variables, we introduce 2n new variables (y1, z1, y2, z2, ...yn, zn).
2. **Total New Variables:** The total number of new variables introduced is 2n, which is linear in the size of the original SAT instance.
3. **Time Complexity:** The operation of introducing new variables is clearly polynomial in the size of the original SAT instance because the number of new variables is linear in the size of the original instance.

Replacing Clauses:

1. **For each of the m original clauses:** We replace it with three clauses in 3-CNF format. So, for m original clauses, we have 3m clauses in the new Twice3SAT instance.
2. **Total New Clauses:** The total number of new clauses introduced is 3m, which is linear in the size of the original SAT instance.
3. **Time Complexity:** The operation of replacing clauses is also polynomial in the size of the original SAT instance because the number of new clauses is linear in the size of the original instance.

Overall Time Complexity:

1. **Combining Both Operations:** The overall time complexity is determined by the sum of the time complexities of introducing new variables and replacing clauses.
2. **Linear Time Complexity:** Both operations result in a linear number of new variables and clauses in the new Twice3SAT instance concerning the size of the original SAT instance.
3. **Polynomial Time:** Since the total number of new variables and clauses is polynomial in the size of the original SAT instance, the overall time complexity of the reduction is polynomial.

Hence, we conclude that the reduction from SAT to Twice3SAT runs in polynomial time because each step involves a constant number of operations, and the overall growth in the number of variables and clauses is polynomial in the size of the original SAT instance. This polynomial time complexity ensures the efficiency of the reduction.

Q2. Solve the following linear program using SIMPLEX:

Maximize 120x + 100y

Subject to - 2x + 2y + u = 8
5x +3y + v = 15
x >=0, y>=o, u>=0 and v>=0

Answer-

Step 1. Introduce slack variables s1 and s2 in the standard form and write everything on the RHS side.

Optimization function - z – 120x – 100y + 0u + 0v + 0s1 + 0s2 = 0

Constraints – 2x + 2y + u + 0v + s1 + 0s2 = 8

5x + 3y + 0u + v + 0s1 + s2 = 15

Step 2. Creating the initial tableau –

	x	y	u	v	s1	s2	RHS	Ratio
z	-120	-100	0	0	0	0	0	
s1	2	2	1	0	1	0	8	
s2	5	3	0	1	0	1	15	

Step 3. Considering the most negative value in the row of z, that will be our key column – Here, 120 is the most negative value.

	x	y	u	v	s1	s2	RHS	Ratio
z	-120	-100	0	0	0	0	0	
s1	2	2	1	0	1	0	8	
s2	5	3	0	1	0	1	15	

Step 4. We calculate the ratio column by dividing a row's RHS entry with the value in the key column.

	x = KEY COLUMN	y	u	v	s1	s2	RHS	Ratio
z	-120	-100	0	0	0	0	0	0
s1	2	2	1	0	1	0	8	4
s2	5	3	0	1	0	1	15	3

Step 5. The least positive ratio row will be considered as our key row – Here, 3 is the least positive ratio

	x = KEY COLUMN	y	u	v	s1	s2	RHS	Ratio
z	-120	-100	0	0	0	0	0	0
s1	2	2	1	0	1	0	8	4
s2	5	3	0	1	0	1	15	3

Step 6. The intersection of the key column and the key row will be our key value – Here, it is 5.

	x = KEY COLUMN	y	u	v	s1	s2	RHS	Ratio
z	-120	-100	0	0	0	0	0	0
s1	2	2	1	0	1	0	8	4
s2 = KEY ROW	5	3	0	1	0	1	15	3

Step 7. We make the key value entry as 1 by doing row operation –

R3 = R3/5.

This changes the values of elements in the key row leading to the below table.

	x = KEY COLUMN	y	u	v	s1	s2	RHS	Ratio
z	-120	-100	0	0	0	0	0	
s1	2	2	1	0	1	0	8	
s2 = KEY ROW	1	3/5	0	1/5	0	1/5	3	

Step 8. We make the values of elements (other than key row) in key column equal to zero. To do this, we perform row operations

R2 = R2 – 2R3

R1 = R1 + 120R3

This leads us to a new table. In this table, the slack variable which is in the key row (here, s2) leaves and the basic variable in the key column (here, x) enters key row.

	x	y	u	v	s1	s2	RHS	Ratio
z	0	-28	0	24	0	24	360	
s1	0	4/5	1	-2/5	1	-2/5	2	
x	1	3/5	0	1/5	0	1/5	3	

Step 9. Now, iteration 1 is over and iteration 2 starts. Here the most negative value in the row of basic variable z is -28, so that will be our key column.

	x	y = KEY COLUMN	u	v	s1	s2	RHS	Ratio
z	0	-28	0	24	0	24	360	
s1	0	4/5	1	-2/5	1	-2/5	2	
x	1	3/5	0	1/5	0	1/5	3	

Step 10. We calculate the ratios and the one with least positive ratio(Here, 5/2) will be our key row.

	x	y = KEY COLUMN	u	v	s1	s2	RHS	Ratio
z	0	-28	0	24	0	24	360	
s1 = KEY ROW	0	4/5	1	-2/5	1	-2/5	2	5/2
x	1	3/5	0	1/5	0	1/5	3	5

Step 11. We make the key value(here, 4/5) as 1 by performing the row operation

R2 = (5/4)*R2.

	x	y = KEY COLUMN	u	v	s1	s2	RHS	Ratio
z	0	-28	0	24	0	24	360	
s1 = KEY ROW	0	1	5/4	-1/2	5/4	-1/2	5/2	
x	1	3/5	0	1/5	0	1/5	3	

Step 12. We make the corresponding entries in the key column as 0, considering the key value. We do this by performing the below row operations –

R3 = R3 – (3/5)R2

R1 = R1 + 28R2.

This leads us to yet another new table. In this table, the slack variable which is in the key row (here, s1) leaves and the basic variable in the key column (here, y) enters key row.

	x	y	u	v	s1	s2	RHS	Ratio
z	0	0	35	10	35	10	430	
y	0	1	5/4	-1/2	5/4	-1/2	5/2	
x	0	0	-3/4	1/2	-3/4	1/2	3/2	

As we have all positives and zeroes in the z row, we have got our final values. The optimal values of each basic variable will be in the RHS row

So the optimal values are - Z = 430

X = 5/2

Y = 3/2

U = 0

V = 0