CMSC 641, Design and Analysis of Algorithms, Spring 2010

Sandipan Dey,
Homework Assignment - 6

March 30, 2010

Finding satisfying assignments

Let ϕ be a formula with n boolean variables $x_1, x_2, \ldots x_n$. We are given a polynomial time algorithm SAT that can decide satisfiabilty of a formula, i.e., $SAT(\phi)$ returns $true$ or $false$ depending on whether or not the formula ϕ is satisfiable. We can use the following logic to find a satisfying assignment in polynomial time, by using the algorithm SAT.

Algorithm 1 SATASSIGN(ϕ): Find the satisfying assigments ($SATLVAL[1 \ldots n]$) for the $n - variable$ boolean formula ϕ

```
1: if SAT(φ)=false then
2:    return null {not satisfiable}
3: end if
4: SATVAL[1...n] ← {false, false, ..., false}
5: for i = 1 to n do
6:    if SAT(φ ∧ x_i)=true then
7:       φ ← SUBSTITUTE(φ, x_i, true) {traverse φ and substitute x_i with
         true in all the clauses}
8:       STATVAL[i] ← true
9:    else if SAT(φ ∧ x_i)=false then
10:      φ ← SUBSTITUTE(φ, x_i, false) {traverse φ and substitute x_i with
         false in all the clauses}
11:      STATVAL[i] ← false
12:   end if
13: end for
14: return SATVAL
```

Analysis

- If the forumula is not satisfiable, line $1-3$ of the algorithm executes which just invokes ploynomial time algorithm SAT and returns null, taking polynomial time only.
- If the formula is satisfiable, the for loop $5-13$ executes for n times and in i^{th} iteration it assigns all the x_is in ϕ either true or false, which is again polynomial time (precisely, if ϕ has m clauses and each clause has at most one instance of a variable, this takes at most $O(m)$ time).
- Hence the overall algorithm for SATASSIGN is a ploynomial time algorithm.

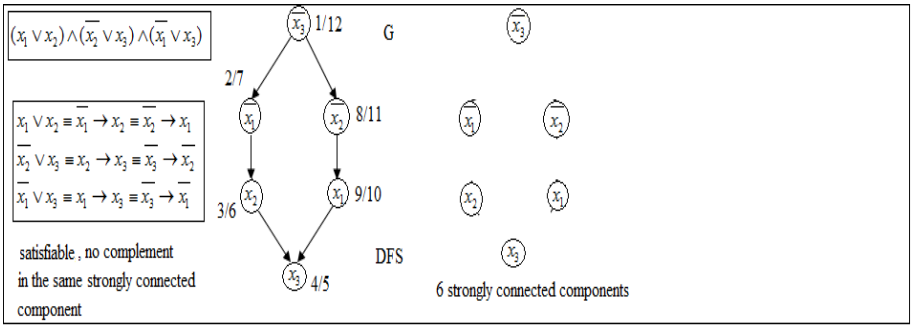
$2CNF - SAT \in P$

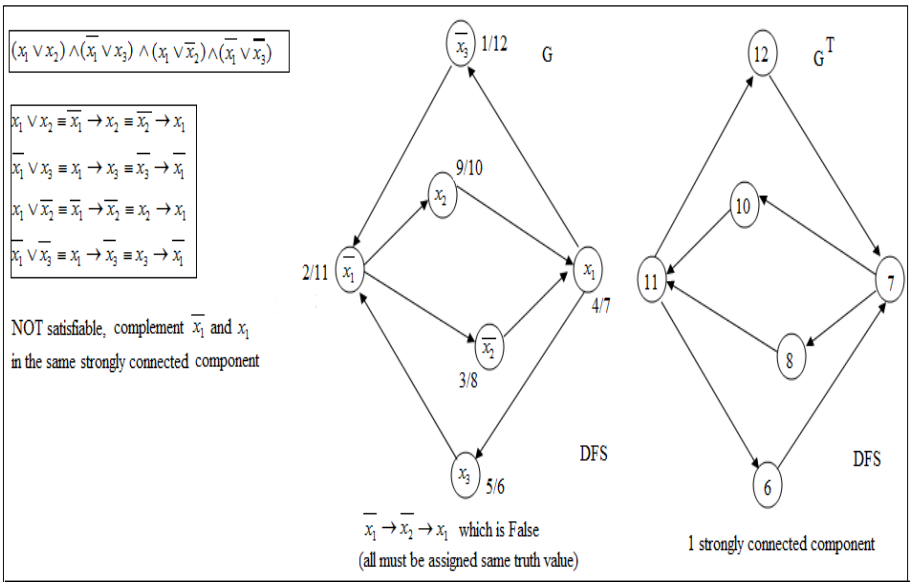
- $2-CNF$ can be represented in the following form:
  - $F = \underset{i=1\ldots m}{\wedge} C_i$.
  - Each clause C_i is a 2-literal caluse and hence of the form $(y \vee z) \equiv (\bar{y} \rightarrow z) \equiv (\bar{z} \rightarrow y)$.
  - $y, z \equiv x_i$ or $\bar{x_i}$, with $i, j \in 1 \ldots n$.
  - F contains total m clauses formed using total n literals.
  - If there is a one literal clause, it must be assigned to true, trivially.
- Construct a directed graph $G(V, E)$, with
  - $v_i \in V$ corresponding to the literal x_i, $\forall i = 1 \ldots n$.
  - $\bar{v_i} \in V$ corresponding to the negated literal $\bar{x_i}$, $\forall i = 1 \ldots n$
  - $|V| = 2n$.
  - $(v_i, v_j) \in E \Leftrightarrow x_i \rightarrow x_j \in F$.
  - e.g., $C_k = \bar{x_i} \vee x_j \equiv x_i \rightarrow x_j \equiv \bar{x_j} \rightarrow \bar{x_i} \in F \Leftrightarrow (v_i, v_j),\ (\bar{v_j}, \bar{v_i}) \in E$.
- The $2-CNF$ formula F is satisfiable iff no pair of complimentary literals are in the same strongly connected component of G. This is illustrated in the following figure.

Proof

$\Rightarrow$

- If F is satisfiable, let's assume to the contrary: let's assume that there is a literal x_i for which x_i and $\bar{x_i}$ are in the same strongly connected component.
  By definition, there is path from u to v and v to u, for every vertex u, v


$(x_1 \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_3)$
$x_1 \vee x_2 \equiv \overline{x_1} \to x_2 \equiv \overline{x_2} \to x_1$
$\overline{x_2} \vee x_3 \equiv x_2 \to x_3 \equiv \overline{x_3} \to \overline{x_2}$
$\overline{x_1} \vee x_3 \equiv x_1 \to x_3 \equiv \overline{x_3} \to \overline{x_1}$
satisfiable , no complement
in the same strongly connected
component
1/12
G
2/7
8/11
3/6
9/10
4/5
DFS
6 strongly connected components


$(x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3})$
$x_1 \vee x_2 \equiv \overline{x_1} \to x_2 \equiv \overline{x_2} \to x_1$
$\overline{x_1} \vee x_3 \equiv x_1 \to x_3 \equiv \overline{x_3} \to \overline{x_1}$
$x_1 \vee \overline{x_2} \equiv \overline{x_1} \to \overline{x_2} \equiv x_2 \to x_1$
$\overline{x_1} \vee \overline{x_3} \equiv x_1 \to \overline{x_3} \equiv x_3 \to \overline{x_1}$
NOT satisfiable, complement $\overline{x_1}$ and x_1
in the same strongly connected component
1/12
G
9/10
2/11
4/7
3/8
5/6
DFS
$\overline{x_1} \to \overline{x_2} \to x_1$ which is False
(all must be assigned same truth value)
12
G^T
10
11
7
8
6
DFS
1 strongly connected component

in the strongly connected component ⇒ in any truth assignment, the corresponding literals must have the same value (since a path is a chain of implications).
⇒ all (literals corresponding to) vertices in the same strongly connected component must have the same assignment. This cannot be the case for a literal and its complement. Hence, if a literal and complement are in the same component the formula is not satisfiable, because then $x_i \to \bar{x_i}$ and vice versa, which is a contradiction.

$\forall x_i$, do the following:

- Collapse each strongly connected component into a single vertex.
- This results in a DAG.
- Run topological sort (using DFS) on this DAG.
- If x_i appears in its true from in one component and in its negated form in another, if the component of x_i appears topologically before that of $\bar{x_i}$, then we set x_i to false.
- Alternatively, if the components appear in the opposite order, we set x_i to true.

$\Leftarrow$

- The truth assignment described above is a satisfying assignment.

  Again, let's prove this by contradiction, as above. Suppose it is not a satisfying assignment, i.e., $\exists$ some clause $(x \vee y)$ that is false
  ⇒ both x and y are set to false. This would have happened exactly when the component containing $\bar{x}$ appears after the component containing x (and similarly the component containing $\bar{x}$ appears after the component containing x).

- Now, from the clause $(x \vee y)$, we had the directed edges $(\bar{x} \to y)$ and $(\bar{y} \to x)$ ⇒ the component containing $\bar{x}$ is before or the same as the component containing y, and the component containing $\bar{y}$ is before or the same as the component containing x.
  But if at the same time the component containing $\bar{x}$ must be later than the component containing x, this would imply that the component containing y must be later than the component containing $\bar{y}$; a contradiction.

$0-1\ IP \in NPC$

1) $0-1\ IP \in NP$

We have to show the 'Yes' certificate is polynomial-time verifiable. The certificate (guess) will be the n unknowns $x = [x_1, x_2, \ldots, x_n]^T$. To verify the

correctness of such a certificate, we need to verify the inequality $Ax \leq b$, the constraint that the solution must satisfy. But thisinvolves just a matrix multiplication (of an $n \times n$ matrix with $n \times 1$ vector with runtime $\theta(n^2)$) followed by n comparisions, hence total runtime $\theta(n^2)$, hence in NP (polynomial time verifiable).

2) $0-1\ IP$ is NP-hard

We prove that $3-SAT \leq_m^p 0-1\ IP$.

Reduction(F)

- In the $3-SAT$, suppose there are n boolean variables $x_1, x_2, \ldots, x_n$ ($x_i \in 0, 1$) and m clauses in the CNF formula.
- Transform each clause $C_j = (x_i \vee x_j \vee x_k)$ in 3-SAT to a linear constraint $x_i + x_j + x_k \geq 1$.
- If a variable appears negated, as $\bar{x_i}$, change it to $1-x_i$ in the corresponding linear constraint.
- To be uniform, change the constraints into standard ($\leq$) form.
- Following the above method, every clause will be changed to to a linear constraint on x.
- There are totally m constraints on n variables x_i.
- Let A be the coefficient matrix of every constraints, and b be the right hand side vector, then A is $m \times n$ and b is $n \times 1$ and $A.x \leq b$. So the 3-SAT problem is transformed into the 0-1 integer programming problem.
- Since there are m clauses, the transformation can be done in polynomial time $O(m)$.
- $3-SAT \leq_m^p 0-1\ IP$

ϕ is satisfiable $\Leftrightarrow \exists$ an n-vector x that solves $Ax \leq b,\ x \in \{0,1\}^n$

Proof ($\Rightarrow$)

- If 3-SAT has a satisfying truth assignment, at least one literal in each clause evaluates to be true. For each literal that evaluates to be true in any given clause, the corresponding term in the corresponding inequality evaluates to 1 (If $x_i = 1$ in the clause, so it is in the inequality; if $\bar{x_i} = 1$ in the clause, $1 - x_i = 1$ in the inequality).

- For each literal that evaluates to be false in any given clause, the corresponding term in the corresponding inequality evaluates to 0 (If $x_i = 0$ in the clause, so it is in the inequality; if $\bar{x_i} = 0$ in the clause, $1 - x_i = 0$ in the inequality).
- $x_i + x_j + x_k \geq 1$ always holds and so does every inequality constraint.
- Hence, there is an n-vector x of $\{0, 1\}$ so that $A.x \leq b$.

Proof ($\Leftarrow$)

- if $Ax \leq b$ has a solution where $x \in \{0, 1\}^n$, use this value to set the corresponding variables in the clauses.
- Since the inequalities are satisfied, $x_i + x_j + x_k \geq 1$ always holds, there is at least one 3-literal (x_i or $1 - x_i$) evaluates to 1, and the corresponding x_i or $\bar{x_i}$ evaluates to 1.
- Hence, each clause is satisfied and 3-SAT has a satisfying truth assignment.

Thus, we have proved that F is a polynomial-time reduction and it follows that 0-1-IP is NP-complete.