CMSC 858F: Algorithmic Lower Bounds
Fall 2014
3-SAT and NP-Hardness

Instructor: Mohammad T. Hajiaghayi
Scribe: Philip Dasler

September 23, 2014

The most important NP-Complete (logic) problem family!

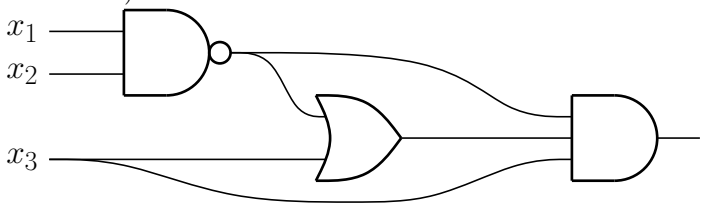
1 SAT = Satisfiability

The Satisfiability problem (introduced by Cook in 1971 [2] and again by Levin in 1973 [11][1]), asks the following question: given a boolean formula F (AND, NOT, and OR) over n variables $x_1, \ldots, x_n$, does there exist a set of assignments for each variable x_i such that F is TRUE.

This problem is well studied and is a well known example of an NP-Complete problem.

2 Circuit SAT

The formula to be satisfied can also be expressed as a circuit of logic gates (allows re-use).

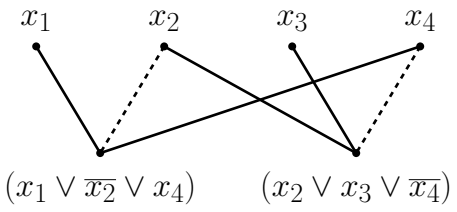


[1] This citation is to a survey in English which translates the original Russian paper. You can find the Russian version here if you're curious.

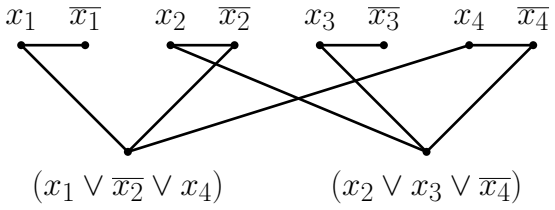
3 CNF SAT [2]

CNF = Conjunctive Normal Form

- The formula consists of a conjunction (logical AND) of clauses
- Each clause is a disjunction (logical OR) of literals
- Each literal $\in \{x_i, \overline{x_i}\}$
- A CNF formula can also be represented as a bipartite graph. One set contains the variables, the other set contains the clauses, and edges are drawn from variables to the clauses they are in. In this case, there are two types of edges, representing positive and negative literals, respectively.



- The first set can contain all of the literals rather than the variables. If so, literals of the same variable (the positive and the negative) must have an edge between them (making this technically not bipartite).



Note that if the bipartite graph between clauses and *variables* is planar; the problem is called planar CNF (type 1).

If the bipartite graph between clauses and *literals* plus all edges $(x_i, \overline{x_i})$ forms a planar graph, the problem is called planar CNF (type 2).

Both versions are NP-Complete by a reduction from 3-SAT (by uncrossing the edge-crosses).

Very useful to prove NP-Hardness of problems on planar graphs and geometric planar graphs.

3.1 3-SAT [2]

Each clause is an OR of exactly 3 variables, i.e., the degree of each clause is 3.

3.1.1 3-SAT-5 [5]

Each variable occurs in ≤ 5 clauses.

3.1.2 Monotone 3-SAT [7]

Each clause consists of either 3 positive literals or 3 negative literals. There is no mixing of "signs".

4 Polynomial-Time Variants of SAT

: All of the above variants of SAT are NP-Complete, but beware of the polynomial-time variants below!

4.1 2-SAT

- Each clause is the logical OR of two literals
- The satisfiability of a formula in this form can be determined in polynomial time
- First, notice that each clause is of the form $(x \vee y)$, but this is equivalent to $(\overline{x} \implies y)$ or $(\overline{y} \implies x)$
- This causes implication chains where we can guess a value for x, forcing assignments for the rest of the values.
- To solve in polynomial time, first try TRUE as an assignment for x and if the formula is satisfied, you're work is done. If not, try FALSE. If the formula is still not satisfied, then it must not be satisfiable.

P	Q	P $\implies$ Q
T	T	T
T	F	F
F	T	T
F	F	T

BUT ...

4.1.1 MAX 2-SAT [6]

A simple modification to the 2-SAT problem will make it NP-Complete. The MAX 2-SAT problem asks for a satisfying assignment of the variables that maximizes the number of TRUE clauses. As always, we can turn an optimization problem into a decision problem by assigning a bounding variable (e.g., can we make more than K clauses TRUE?).

4.2 Horn SAT

A Horn SAT formula is one in which each clause has ≤ 1 positive literals. For example, a Horn Clause may take the form $\overline{x} \vee \overline{y} \vee \overline{z} \vee w$. However, thanks to DeMorgan's Law, this is equivalent to $\neg(x \wedge y \wedge z) \vee w$. This, in turn, is equivalent to $(x \wedge y \wedge z) \implies w$, leading to an implication change, making this a generalization of 2-SAT and thus solvable in polynomial time.

4.3 Dual Horn SAT

In Dual Horn SAT, each clause has ≤ 1 negative literal. This can be reduced to Horn SAT by negating everything (both the literals and the final variable assignment), making it solvable in polynomial time.

5 DNF

DNF = Disjunctive Normal Form

- The formula consists of a disjunction (logical OR) of clauses
- Each clause is a conjunction (logical AND) of literals
- Each literal $\in \{x_i, \overline{x_i}\}$
- To satisfy a formula it is sufficient to satisfy a single clause, which is done as long as there are no contradictions, i.e., a clause cannot include both the positive and negative literals of a variable. Thus, it is essentially trivial.
- Attempting to make the formula FALSE instead is, essentially, the dual of a CNF SAT problem.

6 Alternative Clauses for 3-SAT

6.1 1-in-3-SAT [10]

Exactly 1 of 3 literals in a clause is TRUE.

6.1.1 "Monotone" 1-in-3-SAT

Similar to 1-in-3-SAT, though no negations are allowed, i.e., all literals are positive. This formulation was omitted by Schaefer.

BUT...

6.1.2 "Monotone" Not Exactly 1-in-3-SAT [10]

- Again, all literals are positive.
- "Not Exactly" means that only 0, 2, or 3 variables in a clause may be TRUE
- I.e., $x_i \implies (x_j \vee x_k) \rightarrow$ Dual Horn SAT
- Additionally, x_1 must be assigned the value TRUE (otherwise you could just make everything FALSE)
- Polynomial

6.2 NAE 3-SAT [10]

- NAE = Not All Equal
- The 3 literals in a clause may not all be assigned the same value
- I.e., FFF and TTT are forbidden assignments in a clause, whereas only FFF is forbidden in 3-SAT
- this leads to a nice symmetry between TRUE and FALSE

6.2.1 "Monotone" NAE 3-SAT

- No negations, all literals are positive.
- Also omitted by Schaefer.
- Also NP-Complete

Of all of the problems listed above, the most important ones to remember are: 3-SAT, 1-in-3-SAT, and NAE 3-Sat.

7 Shaefer's Dichotomy Theorem [10] (A Universal Theorem)

- A formula = AND of clauses
- general clause = relation on variables (with implicit truth tables, similar to a type in C++ or having an oracle).
  - assume in CNF
    $\implies$ AND of subclauses
- SAT is polynomial if either:

- setting all variables to TRUE or all variables to FALSE satisfies all relations,
- OR - subclauses are all Horn or all Dual Horn,
- OR - relations are all 2-CNF (subclause sizes ≤ 2, i.e., 2-SAT case),
- OR - every relation can be expressed as a system of linear equations over $\mathbb{Z}_2$.

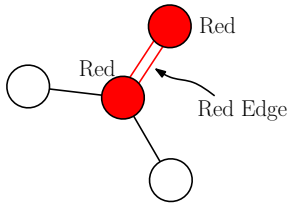
$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0 \text{or} 1$$

where $\oplus$ = XOR. This can be solved via Gaussian elimination.

- If none of the above are true, then SAT is NP-Complete!

8 2-Colorable Perfect Matching [10]

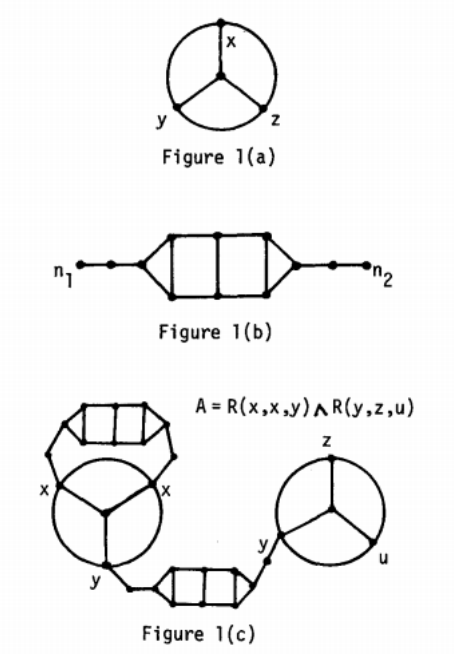
- given a planar 3-regular graph (i.e., a graph in which the degree of each vertex is 3).
- 2-color the vertices such that every vertex has exactly 1 same-colored neighbor. special case of 2-in-4-SAT



(planarity and 3-regular are left as an exercise)

Theorem 1 *There is a reduction from Monotone NAE 3-Sat to 2-Colorable Perfect Matching [10].*

Proof: The proof is based on two key gadgets, as illustrated in the image on the right, taken directly from the paper by Schaefer [10]. Figure 1(a) is a gadget used to represents the individual clauses in a 3-SAT formula. A vertex is created for each literal, which are all linked with an edge. Additionally, an intermediate vertex is added in the center to which each variable is linked. In this way, one cannot set each variable to the same color.
The second gadget, illustrated in Figure 1(b), is a copy gadget. In this case, the color of n_1 must be equal to n_2.
Finally, figure 1(c) illustrates the reduction of A to a 2-Colorable Perfect Matching problem. ■



Some fun games can be proved to be NP-Complete via reduction from 3-SAT, e.g. Push-1 [8].

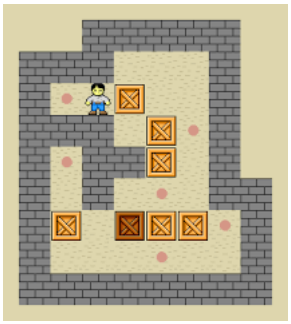


Image of a Push-1 (Sokoban) game taken from Wikipedia - Sokoban.

You may see Erik's class for high-level ideas. The hardness of other games can proven as well, such as:

- Super Mario Bros.
- Legend of Zelda
- Metroid
- Donkey Kong Country
- Pokemon

9 Cryptarithms/Alphametics [9]

- given a formula $x + y = z$ with each number written in base b (which should be some $f(n)$ to be interesting) and encoded with "letters" by an *unknown* bijection between $\{0, 1, \ldots, b-1\}$ and letters
- goal: feasable? / recover bijection
- strongly NP-Complete [3].

Cryptarithm rules:

- each letter represents a unique digit
- often numbers must not start with zero
- often the solution is unique

Example:

$$\begin{array}{r} 9\,5\,6\,7 \\ +\,1\,0\,8\,5 \\ \hline 1\,0\,6\,5\,2 \end{array}$$

Can also be represented as:

$$\begin{array}{r} a\,b\,c\,d \\ +\,e\,f\,g\,b \\ \hline e\,f\,c\,b\,h \end{array}$$

Or:

$$\begin{array}{r} S\,E\,N\,D \\ +\,M\,O\,R\,E \\ \hline M\,O\,N\,E\,Y \end{array}$$

9.1 Reduction from 3-SAT

To reduce from 3-SAT, a long equation is created that represents the clauses, variables, and the boolean nature of the SAT problem. First, the rightmost three columns are set thusly:

$$\begin{array}{r} \mathbb{0}\,p\,\mathbb{0} \\ \mathbb{0}\,p\,\mathbb{0} \\ \hline \mathbb{1}\,q\,\mathbb{0} \end{array}$$

Here the letters $\mathbb{0}$ and $\mathbb{1}$ are forced to be 0 and 1 for any mod.

Next, for each pair of literals v_i and $\overline{v_i}$, the portion below is added to the left side of the formula:

$$\begin{array}{r} \mathbb{0}\ d_i\ \mathbb{0}\ \mathbb{1}\ y_i\ \mathbb{0}\ c_i\,y_i\ \mathbb{0}\ b_i\,y_i\ \mathbb{0}\ a_i\,\mathbb{0} \\ \mathbb{0}\ e_i\ \mathbb{0}\ d_i\,y_i\ \mathbb{0}\ c_i\,y_i\ \mathbb{0}\ b_i\,y_i\ \mathbb{0}\ a_i\,\mathbb{0} \\ \hline \overline{v_i}\ \mathbb{0}\ e_i\,z_i\ \mathbb{0}\ d_i\,z_i\ \mathbb{0}\ v_i\,z_i\ \mathbb{0}\ b_i\,a_i\,\mathbb{0} \end{array}$$

Thus, for the variable gadget:

$$\begin{aligned}
b_i &= 2a_i \\
v_i &= 2b_i + C \text{ (where } C = carry(y_i + y_i) \in \{0,1\}) \\
&= 4a_i + C \equiv C (mod 4) \\
d_i &= 2c_i + C \\
e_i &= d_i + 1 + C \\
&= 2c_i + 1 + 2C \\
\overline{v_i} &= d_i + e_i \\
&= 4c_i + 1 + 3C \\
&\equiv 3C + 1 \equiv 1 - C (mod 4)
\end{aligned}$$

Next, for each clause add:

$$\begin{array}{c}
\mathbb{0}\, u_{ab}\, \mathbb{0}\ v_a\ \mathbb{0}\ \mathbb{1}\ r_i\, \mathbb{0}\, g_i\, w_i\, \mathbb{0}\, f_i\, \mathbb{0} \\
\mathbb{0}\ v_c\ \mathbb{0}\ v_b\ \mathbb{0}\, h_i\, r_i\, \mathbb{0}\, g_i\, w_i\, \mathbb{0}\, f_i\, \mathbb{0} \\
\hline
\mathbb{0}\ t_i\ \mathbb{0}\, u_{ab}\, \mathbb{0}\, t_i\, s_i\, \mathbb{0}\, h_i\, x_i\, \mathbb{0}\, g_i\, \mathbb{0}
\end{array}$$

Thus, for the clause gadget:

$$\begin{aligned}
g_i &= 2f_i \\
h_i &= 2g_i + \{0,1\} \\
&= 4f_i + \{0,1\} \\
t_i &= h_i + 1 + \{0,1\} \\
&= 4f_i + 1 + \{0,1,2\} \\
&= 4f_i + \{1,2,3\} \\
v_a + v_b + v_c &= t_i \equiv \{1,2,3\} (mod 4)
\end{aligned}$$

The reduction is good for NP-Completeness for any mod multiple of 4, but we still need a solution for the puzzle from a satisfying solution, e.g. uniqueness issues.

9.2 Simplified Reduction From 1-in-3-SAT

- Variable gadget: just v_i, no $\overline{v_i}$ (monotone)
- Clause gadget:

$$\begin{aligned}
g_i &= 2f_i \\
h_i &= 2g_i \\
&= 4f_i \\
t_i &= h_i + 1 \\
&= 4f_i + 1 \\
v_a + v_b + v_c &= t_i \\
&= 4f_i + 1 \equiv 1 (mod 4)
\end{aligned}$$

9.3 3-SAT Solvable $\implies$ Cryptorithm Solvable

- distinguish $a_i, b_i, c_i, d_i, \ldots$ by value mod 128 (one class for each variable)
- e.g., below are possible choices for each variable:

$$\begin{aligned} v_i &\equiv 8 (\mathrm{mod} 128) \text{ if TRUE} \\ &\equiv 9 (\mathrm{mod} 128) \text{ if FALSE} \\ a_i &\equiv \{2, 34, 66, 98\} (\mathrm{mod} 128) \\ b_i &\equiv \{4, 68\} \end{aligned}$$

- set $\lfloor \frac{v_i}{128} \rfloor$ and $\lfloor \frac{\overline{v_i}}{128} \rfloor \in [0, (2n)^3]$
  - such that we have distinct sums of triples for all clauses
  - It was proven that for any k there is a set of k numbers all between $1 \ldots k^3$ such that their sum in triples are distinct (we can use powers of 2 but the base would be in $O(4^n)$ and not good for strong NP-Hardness)[1].
- easy proof of polynomial range (based on fusion trees):
  - if $< i$ set by induction, v_i must avoid $v_j + v_k - v_l - v_m - v_p \ < (2n)^5$ choices
    $\implies (2n)^5$ suffices

  $\implies$ strongly NP-hard
- The final result would be in base $(2n)^3 3.128 = 3072n^3$ (see [3] and its revision [4] for all of the details).

A Notes

It is unknown whether or not the graph isomorphism problem is NP-Complete. Because of this, researchers have defined a new class GI, the set of problems which are polynomial-time reducible to the graph isomorphism problem. If the graph isomorphism problem is solvable in polynomial time, then GI would be equal to P.

References

[1] SC Bose and S Chowla. Report inst. theory of numbers, 1959.

[2] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[3] D Eppstein. On the np-completeness of cryptarithms. *ACM SIGACT News*, 18(3):38–40, 1987.

[4] David Eppstein. On the np-completeness of cryptarithms. *Computer Science Department, Columbia University, URL: http://www. ics. uci. edu/˜ eppstein/pubs/Epp-SN-87. pdf*, pages 06–8, 2000.

[5] Uriel Feige. A threshold of ln n for approximating set cover. *JOURNAL OF THE ACM*, 45:314–318, 1998.

[6] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, February 1976.

[7] E Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, June 1978.

[8] Michael Hoffmann. Motion planning amidst movable square blocks: Push-* is np-hard, 2000.

[9] Joseph S Madachy. *Madachy's Mathematical Recreations*. Dover Publications, 1979.

[10] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM.

[11] B.A Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, 6(4):384–400, October 1984.