## The Satisfiability Problem

- Definition A formula $F \in Form(\mathcal{L}^p)$ is called *satisfiable* if there exists a value assignment $v$ that makes the formula true, i.e., such that $v(F) = 1$.
- *The Satisfiability (SAT) Problem*:

  INPUT: A set $U$ of propositional variables and a formula $F$ (dependent on the variables in $U$) in conjunctive normal form,

  $$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

  where each clause $C_i$ is of the form $p_1 \vee p_2 \ldots \vee p_k$ and $p_j$ is a propositional variable from $U$, or its negation.

  OUTPUT: YES if a value assignment $v$ that satisfies the formula $F$ exists, and NO otherwise.
- If $U$ has $n$ propositional variables, the current best method for solving the SAT problem essentially tries all $2^n$ possible value assignments.
- The Satisfiability Problem is NP-complete.

## Examples

- If $U = \{x, y\}$ and

  $$F = (x \vee y) \wedge (\neg x \vee \neg y)$$

  then the answer to the problem is YES.

  A satisfying value assignment is $v(x) = 0$ and $v(y) = 1$.

  Another satisfying value assignment is $v'(x) = 1$ and $v'(y) = 0$.
- If $U = \{x, y\}$ and

  $$F_1 = (x \vee y) \wedge (x \vee \neg y) \wedge \neg x$$

  the answer is NO because no satisfying value assignment exists, i.e. the formula $F_1$ is not satisfiable.

## Value assignments and graphs

The initial test tube of our DNA program corresponds to the following graph $G_n$ that encodes all the possible value assignments as $n$-bit numbers as follows:

- The graph $G_n$ has nodes

  $$a_1, x_1, x_1', a_2, x_2, x_2', \ldots a_n, x_n, x_n', a_{n+1}$$
- There are directed edges:

  – from $a_k$ to $x_k$ and $x_k'$ and

  – from both $x_k$ and $x_k'$ to $a_{k+1}$

This graph is constructed so that all paths that start at $a_1$ and end at $a_{n+1}$ encode an $n$-bit binary number (a value assignment).

At each stage, a path has exactly two choices:

– if it takes the vertex with an unprimed label, it will encode a 1,
– if it takes the vertex with a primed label it will encode a 0.



## Example

In the graph below, the path $a_1 x' a_2 y a_3$ encodes the binary number 01, which corresponds to the value assignment $v(x) = 0$, $v(y) = 1$.

## Encoding the graph as a test tube

Following Adleman's method, the graph is encoded into a test tube of DNA as follows.

- Each vertex of the graph is assigned a random string of length $l$ from $\{A, C, G, T\}$. The length $l = 20$ used by Adleman should suffice.
- This "name" of the vertex has two parts: the first half is denoted by $p_i$, and the second half by $q_i$. Thus, $p_i q_i$ is the name associated with the $i$th vertex.
- Then, a test tube is filled with the following kinds of DNA strands:

  1) For each vertex, put many copies of a $5' \to 3'$ DNA sequence of the form $p_i q_i$ into the test tube.

  2) For each edge from $i$ to $j$, place many copies of a $3' \to 5'$ DNA sequence of the form $\bar{q}_i \bar{p}_j$ ($\bar{x}$ denotes the sequence that is the Watson-Crick complement to $x$).

  3) Add a $3' \to 5'$ sequence of length $l/2$ complementary to the first half of the initial vertex to the test tube. Similarly, add a $3' \to 5'$ sequence complementary to the last half of the final vertex, to the test tube, i.e. add $\bar{p}_1$ and $\bar{q}_n$.

- The key is the following: Every legal path in $G_n$ corresponds to a correctly matched sequence of vertices and edges.
- In order to see this, consider any path in the graph. It naturally consists of a sequence that alternates "vertex, edge, vertex, edge,..."
- Suppose that $v \longrightarrow u$ is an edge. Then, a path that passes through $v$ and then $u$ fits together like "bricks":

- The top $5' \to 3'$ part consists of a series of "vertices".
- The bottom $3' \to 5'$ part consists of a series of edges.
- Note, the vertex $v$ is encoded $p_v q_v$ while the edge is $\bar{q}_v \bar{p}_u$.
- Thus, the end of the vertex and the beginning of the edge can anneal since they are Watson-Crick complements.
- In the same way, the end of the edge and the beginning of the next vertex can also anneal.

- Because the sequences are chosen randomly,, if $l$ is large enough, then there is a high probability that no inadvertent paths will form.
- Thus, after annealing, the test tube will contain DNA encoding all of the paths through the graph; that is, it will encode all $n$-bit sequences, i.e. it will encode all the possible value assignments for the $n$ propositional variables.



## Bio-operations

- The DNA program for SAT uses the bio-operations *Extract* and *Detect*.
- $E(t, i, a)$ denotes all of the sequences in test tube $t$ for which the $i$th bit is equal to $a$, $a \in \{0, 1\}$.
- To implement $E(t, i, a)$ we perform an extract operation that retains from the test tube $t$ all the sequences that contain the name of $x_i$ if $a = 1$, and the name of $x_i'$ if $a = 0$.

## DNA program for SAT: Example

$$F = (x \vee y) \wedge (\neg x \vee \neg y)$$

The computation will consists of constructing a series of tubes.

*Step 1.* The first tube $t_0$ is just the test tube that contains all 2-bit sequences (all possible value assignments for the 2 variables $x$ and $y$).

*Step 2.*

- Let $t_1$ be the test tube that corresponds to $E(t_0, 1, 1)$.
- Let the remainder be $t'_1$ and let $t_2 = E(t'_1, 2, 1)$.
- Pour $t_1$ and $t_2$ together to form $t_3$.
- *Note:* $t_1$ contains all those assignments that make $x = 1$ and therefore make the first clause true.

  From the remaining assignments, there are still some that make the first clause true, namely those that make $y = 1$. This corresponds to $E(t'_1, 2, 1) = t_2$.

  Thus, $t_3 = t_1 \cup t_2$ contains *all* the value assignments that make the first clause true, either because they make $x$ true (these are contained in $t_1$) or because they make $y$ true (these are contained in $t_2$).

## DNA program for SAT contd.

*Step 3.*

- Let $t_4$ be the test tube that corresponds to $E(t_3, 1, 0)$.
- Let the remainder be $t'_4$.
- Let $t_5$ be $E(t'_4, 2, 0)$.
- Pour together $t_4$ and $t_5$ to form $t_6$.
- *Note.* From the value assignments that satisfy the first clause (contained in $t_3$), we have to retain only those that *also* satisfy the second clause.

  To do this, extract first the value assignments that make $\neg x$ true and put them into $t_4$.

  From the value assignments for which $\neg x$ is false (contained in $t'_4$), keep only those that make $\neg y$ true (in $E(t'_4, 2, 0) = t_5$)

  The union of these (contained in $t_6$) contains all value assignments that, besides making the first clause true, make also the second clause true.

*Step 4.*
Check to see if there is any DNA in the last test tube $t_6$. If YES, then the answer to our problem is YES, otherwise it is NO.



| test tube | assignments present |
|---|---|
| $t_0$ | 00, 01, 10, 11 |
| $t_1$ | 10, 11 |
| $t'_1$ | 00, 01 |
| $t_2$ | 01 |
| $t_3$ | 01, 10, 11 |
| $t_4$ | 01 |
| $t'_4$ | 10, 11 |
| $t_5$ | 10 |
| $t_6$ | 01, 10 |

Note that $t_3$ consists of all those sequences (value assignments) that satisfy the first clause: 01, 10, 11.

In the same way, $t_6$ consists of those sequences *from* $t_3$ that satisfy the second clause: 01, 10.

The latter are exactly the correct answers to the SAT problem.

## The general case

Any SAT problem with $n$ variables and $m$ clauses can be solved with at most order $m$ extract steps and one detect step. (By order $m$ we mean that the number of steps is linear in $m$.)

Let $C_1, C_2, \ldots C_m$ be the clauses in the formula.

We will construct a series of test tubes $t_0, \ldots t_m$ so that $t_k$ is the set of $n$-bit numbers corresponding to the value assignments $v$ such that

$$v(C_1) = v(C_2) = \ldots = v(C_k) = 1.$$

For $t_0$ use $t_{all}$ of all possible $n$-bit numbers.

Let $t_k$ be constructed; we will show how to construct $t_{k+1}$.

Let $C_{k+1}$ be the clause

$$p_1 \vee p_2 \vee \ldots \vee p_l$$

where each $p_i$ is a literal (either a propositional variable or its negation).
For each literal $x_i$, operate as follows:

- If $p_i$ is equal to $x_j$, then form $E(t_k, j, 1)$;
- If $p_i$ is equal to $\neg x_j$ then form $E(t_k, j, 0)$.
  (Note that, as in the example, we perform each extraction and then use the remainder for the next one)
- Put the tubes together like in the example to form $t_{k+1}$.

In the end, do one detect operation on $t_m$ to decide whether or not the clauses are satisfiable.

## Comments

- In the above we assumed that the operations are perfect, i.e. that the operations are performed withouth error.
- The assumption that *Extract* gets all of the sequences is not needed. If the original test tube has many copies of the desired sequence, then we only need some reasonable probability that it is correctly extracted to make everything work propely.