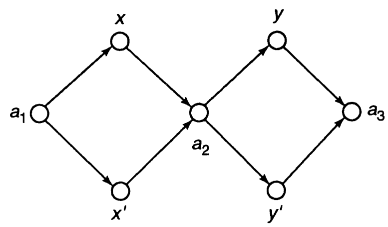

Figure IV.7: Graph G_2 for Lipton's algorithm (with two variables, x and y). [source: Lipton (1995)]

B.2 Lipton: SAT

This lecture is based on Richard J. Lipton (1995), "DNA solution of hard computational problems," *Science* **268**: 542–5.

B.2.a Review of SAT problem

¶1. **Boolean satisfiability:** The first problem proved to be NP-complete.

¶2. Use conjunctive normal form with n variables and m clauses.

B.2.b Data representation

¶1. **Solutions:** Solutions are n-bit binary strings.

¶2. These are thought of as paths through a particular graph G_n (see Fig. IV.7).
For vertices $a_k, x_k, x'_k, k = 1, \ldots, n$, and a_{n+1},
there are edges from a_k to x_k and x'_k,
and from x_k and x'_k to a_{k+1}.

¶3. Binary strings are represented by paths from a_1 to a_{n+1}.
A path through x_k encodes the assignment $x_k = 1$ and through x'_k encodes $x_k = 0$.

¶4. The DNA encoding is essentially the same as in Adleman's algorithm.

B.2.c Algorithm

¶1. Suppose we have an instance (formula) to be solved:
$I = C_1 \wedge C_2 \wedge \cdots \wedge C_m$.

¶2. **Step 1 (initialization):** Create a test tube of all possible n-bit binary strings, encoded as above.
Call this test tube T_0.

¶3. **Step 2 (clause satisfaction):** For each clause C_k, $k = 1, \ldots, m$:
Extract from T_{k-1} only those strings that satisfy C_k, and put them in T_k.
The goal is that for every string $\forall x \in T_k \forall 1 \leq j \leq k : C_j(x) = 1$.
This is done as follows.

¶4. **Extract operation:** Let $E(T, i, a)$ be the operation that extracts from test tube T all (or most) of the strings whose ith bit is a.

¶5. For $k = 0, \ldots, m - 1$:
Precondition: The strings in T_k satisfy clauses $C_1, \ldots, C_k$.
Let $\ell = |C_k|$, and suppose C_{k+1} has the form $v_1 \vee \cdots \vee v_\ell$, where the v_i are literals (plain or complemented variables).
Let $\overline{T}_k^0 = T_k$.
Do the following for literals $i = 1, \ldots, \ell$.

¶6. **Positive literal:** Suppose $v_i = x_j$ (some positive literal).
Let $T_k^i = E(\overline{T}_k^{i-1}, j, 1)$.
These are the paths that satisfy this positive literal.

¶7. **Negative literal:** Suppose $v_i = x'_j$ (some negative literal).
Let $T_k^i = E(\overline{T}_k^{i-1}, j, 0)$.
These are the paths that satisfy this negative literal.

¶8. In either case, T_k^i are the strings that satisfy literal i.
Let $\overline{T}_k^i = E(\overline{T}_k^{i-1}, j, \neg a)$ be the remaining strings (which do not satisfy this literal).
Continue until all literals are processed.

¶9. Combine $T_k^1, \ldots, T_k^\ell$ into T_{k+1}.
Postcondition: The strings in T_{k+1} satisfy clauses $C_1, \ldots, C_{k+1}$.

¶10. **Step 3 (detection):** At this point, the strings in T_m satisfy $C_1, \ldots, C_m$, so do a *detect* operation to see if there are any strings left.
If there are, the formula is satisfiable; if not, not.

¶11. **Performance:** If the number of literals is fixed (as in 3SAT), then performance is linear in m.

¶12. **Errors:** The main problem is the effect of errors. But imperfections in extraction are not fatal, so long as there are enough copies of the desired sequence.