Independent Set Problem

Input: a graph G and a lower bound k.

Output: "yes" iff there are at least k *independent* nodes of G; i.e., nodes with no edges interconnecting.

Reduction from: 3SAT.

- Clearly, this problem is in $\mathcal{NP}$; just guess k nodes and check that they have no edges among them.

The Reduction

Take a 3-SAT instance such as $(x+y+z)(\bar{x}+\bar{z}+w)$.

- Create node $[i,j]$ for the jth literal in the ith clause.
  - ✥ i ranges from 1 to the number of clauses — certainly $O(n)$, where $n =$ the input length.
  - ✥ $j = 1, 2,$ or 3.
- Edges among the three nodes with a common i prevent more than one of them being chosen in an independent set.
- Edges between nodes for any literal and its complement.
  - ✥ In our little example: $[1,1]$ and $[2,1]$ are connected (x and $\bar{x}$); $[1,3]$ and $[2,2]$ are also connected (z and $\bar{z}$).
- Pick $k =$ number of clauses.

Proof the Reduction is Correct

- First, suppose we have a satisfying truth assignment for the variables.
  - ✥ Pick one true literal from each clause (there could be more, but not fewer).
  - ✥ The nodes corresponding to these literals form an independent set of size k.
  - ✥ Why? The only edges among them would connect nodes for different clauses, and these would have to go between a literal and its complement, both of which could not have been selected.

- Now, suppose we have an independent set of size k.
  - This set cannot have more than one node from any one clause.
  - This set cannot choose nodes corresponding to a literal and its complement.
  - Thus, it tells us a truth assignment for enough of the variables that every clause is made true.

Coping With Complexity

When faced with an NP-complete problem, there are three things we can do:

1. *Approximate.* For example, do we need an absolutely maximum-size independent set?
   - Perhaps a greedy heuristic (grab any node we see as long as it has no edges connected it to those we've selected already) will get an independent set that is big enough?
2. *Restrict.* Do we really need to solve the problem in all its generality? Or could a special case that has a polynomial algorithm serve our needs?
   - Example, while 3SAT is NP-complete, the 2SAT problem (clauses of 2 literals only) has a subtle, linear-time algorithm.
3. *Tough It Out.* Sometimes we are only interested in problem instances that are small enough that the exponential growth doesn't overwhelm our resources.
   - Query optimization algorithms are like that: everything is NP-complete, but database queries tend to be very small.
   - Traveling Salesman is an unusual NP-complete problem because it is in fact very easy to solve even 1000-city problems. Thus, it is used by many snake-oil salesmen to demonstrate that their favorite algorithmic methodology "beats" NP-completeness (e.g., Hopgood with neural nets, Adelman with DNA algorithms).

Out Beyond $\mathcal{NP}$

There is no end to the number of complexity classes that can be invented by mathematically inclined academics desirous of gaining tenure. Some of these are actually interesting.

Co-NP

A language/problem is in Co-NP if its complement is in $\mathcal{NP}$.

- If $\mathcal{P} = \mathcal{NP}$, then Co-NP $= \mathcal{NP}$.
  - Why? because the complement of a problem in $\mathcal{P}$ is surely in $\mathcal{P}$, since we can just complement the answer in one more step.
- However, if $\mathcal{P} \neq \mathcal{NP}$, as we assume, then Co-NP $\neq \mathcal{NP}$ is likely, although not certain.
- Apparent example: The complement of SAT (i.e., all Boolean expressions that are not satisfiable, plus the "garbage" that is not a well-formed expression) appears not to be in $\mathcal{NP}$.
  - While we can guess a satisfying truth assignment and check that we guessed right in polynomial time, there is no way to "guess why there is no such assignment."
  - Note that the nonsatisfiable expressions are the negations of the *tautologies* (expressions that are always true), so tautology testing is another example of a Co-NP problem that appears not to be in $\mathcal{NP}$.

PSPACE

A TM that uses no more than $p(n)$ space on input of length n, for some polynomial p, is said to be in PSPACE.

- You might think that it matters whether the TM is deterministic or nondeterministic, but it doesn't! See below.
- A PSPACE TM can take exponential time before accepting.
- However, if it takes more than $k^{p(n)}$ moves, where k = sum of the number of states and tape symbols, then it has repeated an ID and so has a shorter sequence of moves leading to acceptance if it accepts at all.

Example

The tautology problem is in PSPACE.

- Use linear space to enumerate all possible truth assignments, one at a time (i.e., run a counter in binary).
- Check each assignment, say "no" if you find one that doesn't make the expression true, and say "yes" if you reach the end.

PSPACE-complete Problems

While $\mathcal{P} \subseteq \mathcal{NP} \subseteq$ PSPACE is obvious (remember that PSPACE includes nondeterministic TM's), it is not even known whether $\mathcal{P} =$ PSPACE.

- Say a problem L is *PSPACE-complete* if every problem in PSPACE polynomial-*time* reduces to L.
  - Thus, if L is in $\mathcal{P}$, then $\mathcal{P} =$ PSPACE; if L is in $\mathcal{NP}$, then $\mathcal{NP} =$ PSPACE.

Example

QBF (*Quantified Boolean Formulas*) is a PSPACE-complete problem.

- Example of a QBF: $(\forall x)(\exists y)(x\bar{y} + \bar{x}y)$.
  - This instance of QBF has answer "yes" (true), because we can pick y to be the complement of x.

Savitch's Theorem: Equivalence of Deterministic and Nondeterministic PSPACE

Key ideas:

1. If a PSPACE NTM accepts, it does so within $k^{p(n)}$ steps.
2. A simulating DTM uses a recursive algorithm to answer questions of the form: "does ID $\alpha \overset{*}{\vdash}$ ID β in at most 2^i steps?"

- *Basis*: $i = 0$. Check if $\alpha = \beta$ or $\alpha \vdash \beta$.
- *Induction*: For each possible γ [ID of length at most $p(n)$], recursively check if $\alpha \overset{*}{\vdash} \gamma$ in at most 2^{i-1} moves and $\gamma \overset{*}{\vdash} \beta$ in at most 2^{i-1} moves.
  - Return "yes" if any such γ found; return "no" if not.
  - You need only one "stack frame" of length $p(n)$ to generate and store each possible γ (use a counter in base k).

- Clincher: We can limit the stack to $p(n)\log_2 k$ recursive calls, taking a total of $p^2(n)\log_2 k$ space, a polynomial if $p(n)$ is.
  - Why? That is enough to answer the question "does $\alpha \overset{*}{\vdash} \beta$ in at most $2^{p(n)\log_2 k} = k^{p(n)}$ moves?"
  - Let α be the initial ID, and (using a counter) β be any of the possible accepting ID's of length $p(n)$.
  - Remember, if acceptance occurs, $k^{p(n)}$ moves is enough.