NP-Complete Problems

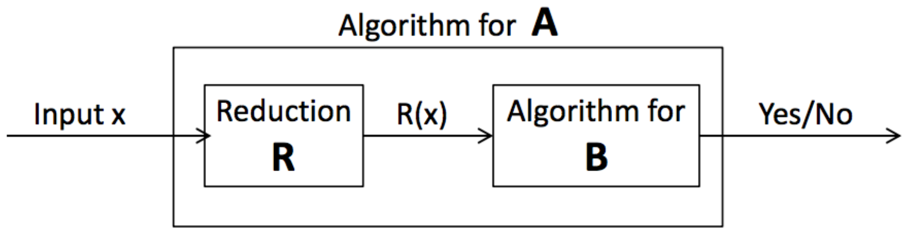
1 Definitions

P: The set of all decision problems D for which there exists a polynomial time algorithm A such that $A(x) = D(x)$. We consider a polynomial time algorithm to be "efficient".

NP: The set of all decision problems D for which there exists a polynomial time verification algorithm V such that for all inputs x, $D(x) = 1$ if and only if there exists a polynomial-length "certificate" y such that $V(x, y) = True$.

In order to compare the "hardness" of solving different problems, we use reductions! The idea is that if I have two problems A and B, if I can show that I can solve A by using a black box that solves B, then I can understand the difficulty of solving A in terms of the difficulty of solving B plus the work required to transform a solution of B to the solution of A.

"Karp"-Reduction: Let $A : X \to \{0, 1\}$ and $B : Y \to \{0, 1\}$ be decision problems. A is *polytime reducible* to B, or "A reduces to B" ($A \underline{\propto} B$), if there exists a function $R : X \to Y$ that transforms inputs to A into inputs to B such that $A(x) = B(R(x))$. The following picture shows how this reduction leads to an algorithm for A which simply builds upon the algorithm for B.



Solving A is no harder than solving B. In other words, if solving B is "easy" (i.e. $B \in P$), then solving A is easy ($A \in P$). Equivalently, if A is "hard", then B is "hard". Given an algorithm for B, we can easily construct an algorithm for A.

NP-Hard: A decision problem D is NP-Hard if all problems Q in NP are polynomial time reducible to it ($Q \underline{\propto} D$ for all $Q \in NP$). That is, given an efficient algorithm which solves an NP-Hard problem D, we can construct an efficient algorithm for *any* problem in NP.

NP-Complete: A decision problem D is NP-Complete if it is in NP and is NP-Hard.

2 Reducing Hamiltonian Cycle to Hamiltonian Path

Hamiltonian Cycle: Given a directed graph $G = (V, E)$, is there a cycle that visits every vertex exactly once?

Hamiltonian Path Given a directed graph $G = (V, E)$, is there a path that visits every vertex exactly once?

Given that Hamiltonian Cycle is NP-Complete, we prove that Hamiltonian Path is NP-Complete.

1. **Show Hamiltonian Path is $\in$ NP**

   To prove this, we need to prove that there exists a verifier $\mathcal{V}(x, y)$. Let $x = G$ be a "yes" input. Let y be a path P that satisfies the condition.

   We can verify that the path traverses every vertex exactly once, then check the path to ensure that every edge in the path is an edge in the graph. Naively, it takes O(n^2) to check that every vertex is traversed exactly once and O(nm) to check that every edge in the path is in the graph.

2. **Show Hamiltonian Path is $\in$ NP-Hard.**

   We prove this by giving a Karp-reduction of Hamiltonian Cycle to Hamiltonian Path.

   (a) Given an instance of Hamiltonian Cycle G, choose an arbitrary node v and split it into two nodes v' and v''. All directed edges into v now have v' as an endpoint, and all edges leaving v leave v'' instead. We call this new graph G'. The transformation takes at most O(E).

   (b) If there is a Hamiltonian Cycle in G, there is a Hamiltonian Path on G'. We can use the edges of the cycle on G as the path on G', but our path must begin on v'' and end on v'.

   (c) If there is a Hamiltonian Path on on G', there is a Hamiltonian Cycle on G. The path must begin at v'' and end at v', since there are no edges into v'' or out of v'. Thus we can use the path on G' as a cycle on G once v' and v'' are remerged.

3. This proves Hamiltonian Cycle reduces to Hamiltonian Path in polynomial time, which means that Hamiltonian Path is at least as hard as Hamiltonian Cycle, so Hamiltonian Path is NP-Complete.

3 Reducing 3 Dimensional Matching to 4 Partition

3DM: Given 3 disjoint sets X, Y, Z, which each contain n elements, and triples T $\subset X \times Y \times Z$, is there a a subset $S \subset T$ such that each element $a \in X \cup Y \cup Z$ is in exactly one $s \in S$.

4-Partition: Given n integers $\{a_1, a_2, \ldots, a_n\}$ is there a partition into $\frac{n}{4}$ subsets of 4 elements each with the same sum t = $\sum A/\frac{n}{4}$. Each integer $a_i \in (\frac{t}{5}, \frac{t}{3})$ (which ensures that each subset has exactly 4 elements).

Given that 3DM is NP-Complete, we prove that 4-Partition is NP-Complete.

1. **Show 4-Partition $\in$ NP** We can verify a potential solution by checking that the elements in each partition sum to t, and that no element is used more than once. This takes at most O(n^2) naively.

2. **Show 4-Partition $\in$ NP**

   We prove this by giving a Karp-reduction of 3DM to 4-Partition.

   (a) Given an input X, Y, Z and triples T to 3DM, we create numbers in base r, where r = $100 * \sum(X \cup Y \cup Z)$. We note that $N[x_i]$ denotes the number of times an element x_i appears in a triple in T.

   Then we create the ”actual” numbers as part of our set of n integers.

   i. For every element $x_i \in X$: $10r^4 + ir^3 + 1$
   ii. For every element $y_j \in Y$: $10r^4 + jr^2 + 2$
   iii. For every element $z_k \in Z$: $10r^4 + kr + 4$

   We also add the following ”dummy” numbers to our set of n integers.

   i. For every element $x_i \in X$: $N[x_i] - 1$ copies of the number $11r^4 + ir^3 + 1$
   ii. For every element $y_j \in Y$: $N[y_j] - 1$ copies of the number $11r^4 + jr^2 + 2$
   iii. For every element $z_k \in z$: $N[z_k] - 1$ copies of the number $8r^4 + kr + 4$

   Finally, we add a ”triple” number for each triple in T.

   i. For every triple (x_i, y_j, z_k): $10r^4 - ir^3 - jr^2 - kr + 8$

   We set our target sum t = $40r^4 + 15$, which can be achieved by adding a ”triple element” $10r^4 - ir^3 - jr^2 - kr + 8$ with three ”actual elements” or a ”triple element” with three ”dummy elements”. Additionally, we have chosen r sufficiently large that no other combination of ”actual”, ”dummy”, or ”triple” elements will meet the target sum.

   (b) If $S \subset T$ is a solution to 3DM, we can construct a 4 Partition solution. For every triple $(x_i, y_j, z_k) = s \in S$, we create a 4 element set out of the corresponding ”triple element” and the three ”actual elements” corresponding to x_i, y_j, z_k which will sum to $40r^4 + 15$. The remaining matchings will correspond to partitions with a ”triple” element and three ”dummy” elements corresponding to x_i, y_j, z_k which will still sum to $40r^4 + 15$. Thus we can form sets of 4 elements, and we have a 4 Partition.

(c) Suppose we are given a solution to 4 Partition. Then consider any 4 element set in this 4 Partition. By considering the the sum of the element sizes modulo $r, r^2, r^3, r^4, andr^5$ we show that this set contains one element corresponding to each of the members in a triple, and that all three elements are either ”actual” elements or ”dummy” elements. If the triple contains only ”actual elements”, it is part of the 3DM solution $S \subset T$, otherwise it is not.

If B equals the sum of the four elements, we know that B mod r = 15. This is possible only if the 4 elements correspond to a triple element, and one element each from X, Y, and Z.

The sum B mod r^2 = 0r + 15, which is possible only if the triple element and the dummy or actual element for z_k match, ensuring that we use a triple element and the corresponding z_k element in our set.

The sum B mod r^3 = 0r^2 + 15, which is possible only if the triple element and the dummy or actual element for y_j match, ensuring that we use a triple element and the corresponding y_j element in our set.

The sum B mod r^4 = 0r^3 + 15, which is possible only if the triple element and the dummy or actual element for x_i match, ensuring that we use a triple element and a x_i element in our set.

The sum B mod r^5 = 40r^4 + 15, which is possible only if the triple element and only three dummy elments or three actual elements were used.

Thus, we can guarantee that if we have a 4 Partition, each set in the partition corresponds to a 3DM matching in S or an unused matching.

3. This proves 3DM reduces to 4-Partition in polynomial time, which means that 4-Partition is at least as hard as 3DM, so 4-Partition is NP-hard.

4 Reducing Clique to Independent Set

Clique: Given graph $G = (V, E)$ and integer k, is there a set of vertices $C \subseteq V$ with $|C| = k$ that form a complete graph?

Independent Set: Given graph $G = (V, E)$ and integer k, is there a set of vertices $I \subseteq V$ with $|I| = k$ such that for any $u, v \in I$, $(u, v) \notin E$?

Given that Clique is NP-Complete, we prove that Independent Set is NP-complete.

1. **Show Independent Set $\in$ NP**

   To prove this, we need to prove there exists a verifier $\mathcal{V}(x, y)$. Let $x = (G, k)$ be a "yes" input. Let y be I that satisfies the condition.

   It takes $O(|I|)$ to check that $|I| = k$. It takes $O(|I|^2)$ to check that for every $u, v \in V'$, $(u, v) \notin E$. This checks in polynomial time that the certificate y proves that x is a valid input. Therefore, Independent Set is in NP.

2. **Show Independent Set $\in$ NP-hard**

   We prove this by giving a Karp-reduction of Clique to Independent Set.

   (a) Given an input $x = (G, k)$ to Clique, create input G' which has the same vertices, but has edge (u, v) if and only if $(u, v) \notin E$. This takes $O(|E|)$ time, so this reduction takes polynomial time.

   (b) If I is a set of vertices that form a k-Independent Set for G', then $C = I$ is a k-Clique for G because for $u, v \in I$, Independent Set says that $(u, v) \notin E'$, but this implies that $(u, v) \in E$ for the Clique problem due to the method of construction. This shows that there are edges between every pair of nodes in C. In addition $|C| = k$, and so C is a k-clique.

   (c) If C is a set of vertices that form a k-Clique in G, then $I = C$ is a k-Independent set for G'. This is because $u, v \in C$ implies that $(u, v) \in E$ for Clique, and this implies that $(u, v) \notin E'$ for Independent Set. Since $|I| = |C| = k$, this shows that there are k elements in the construction G' that are not adjacent to each other.

3. This proves Clique reduces to Independent Set in polynomial time, which means that Independent Set is at least as hard as Clique, so k-Independent Set is NP-hard.

5 Reducing Vertex Cover to Set Cover

Vertex Cover: Given graph $G = (V, E)$ and integer k, does there exist $Y \subseteq V$ such that $|Y| = k$ and for each $(u, v) \in E$, either $u \in Y$ or $v \in Y$ (or both)?

Set Cover: Given a set S of n elements $\{1, 2, \ldots, n\}$ and m sets $S_1, \ldots, S_m$ where $S_i \subseteq S$, does there exist a set of k sets $S_{i_1}, \ldots, S_{i_k}$ such that $S_{i_1} \cup \cdots \cup S_{i_k} = S$?

Given that Vertex Cover is NP-complete, we prove that Set Cover is NP-Complete.

1. **Show Set Cover $\in$ NP:** To prove this, we need to prove there exists a verifier $\mathcal{V}(x, y)$. Let $x = S, S_1, \ldots, S_m$ be a "yes" input. Let y be $S_{i_1}, \ldots, S_{i_k}$ that satisfies the condition.

   In $O(k)$ time, we can figure whether or not we have exactly k sets. In $O(kn)$ time we can determine whether or not all all elements in S is in the union. This proves we have a polynomial time verifier, which means that Set Cover is in NP.

2. **Show Set Cover $\in$ NP-Hard:** To prove this, we reduce Vertex Cover to Set Cover.

   (a) For an input $x = (G, k)$ to Vertex Cover, we make $R(x) = S, S_1, \ldots, S_m$. Let S be the set of all edges $e_j \in E$. For each $v_i \in V$, create set S_i. This set contains the set of edges e_j that touch v_i. This new input is polynomial because $|S| = |E|$ and each set S_i has size at most $|E|$ and there are $|V|$ sets.

   (b) If there is a k-Vertex Cover, there is a k-Set Cover. If the vertex $v_i \in V'$ is part of the vertex cover, then S_i is part of the set cover. Since every edge $e_j \in E$ is incident to some vertex $v_i \in V'$, this means that every element $e_j \in S$ is covered the set S_i.

   (c) If there is a k-Set Cover, there is a k-Vertex Cover. If S_i is in the set cover, choose v_i to be in the vertex cover. Every element e_j is contained in some set S_i. By construction, this means every edge e_j is incident to the vertex v_i that got chosen. Since there are k sets, there will be k vertices chosen for the vertex cover.

6 Prove Max2SAT is NP-Complete: Reducing from Clique

Clique(G, k)**:** Given graph $G = (V, E)$ and integer k, is there a set of vertices $U \subseteq V$ with $|U| \geq k$ that form a complete graph (k-clique)?

Max2SAT(C, X, k)**:** Given a CNF formula consisting of clauses $C = \{c_1, c_2, \ldots c_n\}$ and literals $X = \{x_1, x_2, \ldots x_k\}$ such that each clause involves exactly *two* literals, does there exist an assignment of TRUE/FALSE to the literals such that at least k clauses are satisfied?

6.1 Show Max2SAT $\in$ NP

To prove this, we need to prove there exists a polytime verifier. Given any "yes" input (CNF formula), the "witness/certificate" is an assignment of TRUE/FAlSE to the literals that satisfies at least k clauses. A polytime verifier can simple evaluate each clause of the CNF formula given the assignment, and verify that at least k of them are satisfied.

6.2 Show Max2SAT $\in$ NP-Hard:

1. To prove this, we reduce Clique to Max2SAT. Given an input (G, k) to Clique, we will create an input (C, X, k') to Max2SAT such that "yes" instances of Clique map to "yes" instances of Max2SAT, and "no" instances of Clique map to "no" instances of Max2SAT.

   (a) In designing the CNF formula, we use literals $x_1, x_2, \ldots x_n$ to represent the n vertices in the graph. We want to design clauses that act as constraints to enforce that $x_1 =$ TRUE corresponds to vertex 1 being chosen in the clique.

   (b) Recall that a set of vertices U is a clique if for all $i, j \in U$, $(i, j) \in E$. Equivalently (by the contrapositive), U is a clique if for all $i, j \in V$ such that $(i, j) \notin E$, either $i \notin U$ or $j \notin U$. We will use these constraints to design the clauses in our CNF formula.

   (c) Therefore, for every "non-edge" $(i, j) \notin E$, we have a clause $(\neg x_i \vee \neg x_j)$. This means that for every non-edge, at least one of the endpoints must not be in the clique.

   (d) However, these clauses could also be satisfied by setting all literals x_i to FALSE. In order to encourage choosing cliques with more vertices, for every vertex i we introduce the clauses $(x_i \vee z) \wedge (x_i \wedge \neg z)$, where z is a new literal. In order to satisfy these two clauses, x_i must be true.

   (e) Choose $k' =$ number of non-edges $+ |V| + k$.

   Therefore, the input to Max2SAT is defined by

   - literals $X = V \cup \{z\}$,
   - clauses $C = \{(\neg x_i \vee \neg x_j)$ for all non-edges $(i, j)\} \cup \{(x_i \vee z) \wedge (x_i \wedge \neg z)$ for all vertices $i\}$,
   - and $k' = |\text{non-edges}| + |V| + k$.

2. Show that if Clique is "yes", then Max2SAT is "yes". Given a clique $U \subseteq V$ in graph G such that $|U| \geq k$, we can use an assigment such that for all $i \in U$, x_i is TRUE, for all $i \notin U$, x_i is FALSE, and z is TRUE. This is an assigment that satisfies at least k' clauses in our CNF formula. First, for all non-edges (i, j), the clause $(\neg x_i \vee \neg x_j)$ is satisfied because U is a clique. For all $i \in U$, both $(x_i \vee z)$ and $(x_i \wedge \neg z)$ are satisfied. For all $\imath \notin U$, $(x_i \vee z)$ is satisfied while $(x_i \wedge \neg z)$ is not satisfied. Therefore, the number of satisfied clauses is $|\text{non-edges}| + 2|U| + |V \setminus U| = |\text{non-edges}| + |V| + |U| \geq k'$.

3. Show that if Max2SAT is "yes, then Clique is "yes". Given an assignment for the literals X such that at least k' clauses are satisfied, we will show that we can find a k-clique in the original graph. If for all $(i, j) \notin E$, $(\neg x_i \vee \neg x_j)$ is satisfied, then the literals that are set to TRUE form a clique in graph G that has size $\geq k$ (by construction). However, if the current assignment does not correspond to a clique, then we will show that we can modify the assigment to find a clique of size at least k. For every clause $(\neg x_i \vee \neg x_j)$ that is not satisfied, we change x_i to FALSE (doesn't matter which one we pick), thus satisfying this clause. This will cause one of $(x_i \vee z)$ or $(x_i \wedge \neg z)$ to become not satisfied. In addition, x_i may also appear in other "non-edges" clause. Therefore, the net change is that the number of clauses satisfied can only increase or stay the same by this modification. We continue to choose unsatisfied "non-edge" clauses and modifying the assignment in this way until all the "non-edge" clauses are satisfied. Note that the number of unsatisfied "non-edge" clauses is monotonically decreasing in each modification. Therefore, we obtain an assignment which corresponds to a clique (since all "non-edge" clauses are satisfied). In addition the clique has size at least k because the number of satisfied clauses is still at least k', since every modification does not decrease the number of satisfied clauses.