A Guide to Reductions

Jason D. Hartline

First version: February 22, 2017;
This version: April 30, 2017.

A reduction is an algorithm for solving one problem using an algorithm for another problem. These notes describe how to construct reductions and prove them correct.

A reduction is an algorithm for problem Y that makes use of a black box (the existence of an algorithm) for problem X. Reductions can be used positively: if algorithms exist for X, then a reduction from Y to X shows that algorithms exist for Y. This approach solves a new problem Y by "reduction **to** X." Reductions can be used negatively: if problem Y is believed to be hard, then a reduction from Y to X shows that problem X is at least as hard as Y. This approach shows that a new problem X is hard by "reduction **from** Y." The former is a proof of tractability, the latter is a proof of intractability.

While, in general, an algorithm that gives a reduction from Y to X can make many calls to the black box that solves X, many reductions have a much simpler and more direct form. We will call these *one-call* reductions.[1] A one-call reduction is a polynomial time algorithm that constructs an instance x^y of X from an instance y of Y so that their optimal values are equal, i.e., $\text{OPT}(x^y) = \text{OPT}(y)$. The main challenge of obtaining such a construction is figuring out how to use the constraints of problem Y to simulate the constraints of problem X so that the solutions have the same value.

To prove a reduction is correct, it must be shown that for the constructed x^y from y that $\text{OPT}(x^y) = \text{OPT}(y)$. Such a proof, intuitively, must show that the optimal value of y is high if and only if the optimal value of x is high. Usually such a proof will be proved in two parts. The first part will show that $\text{OPT}(y) \geq \text{OPT}(x^y)$ and the second part will show that $\text{OPT}(x^y) \geq \text{OPT}(y)$. A straightforward way to prove that $\text{OPT}(y) \geq \text{OPT}(x^y)$ is to construct, from the solution to $\text{OPT}(x^y)$, a solution to y that has value at least $\text{OPT}(x^y)$, and vice versa for $\text{OPT}(x^y) \geq \text{OPT}(y)$.

From the above discussion we can view a reduction and its proof of correctness as giving three algorithms:

[1] One-call reductions are also called *Karp reductions* (after Richard Karp, one of the pioneers of the theory of NP-completeness) and *strong reductions*. The more general reduction where the algorithm for Y may make many calls to the algorithm that solves X are known as *Turing reductions* after Alan Turing, one of the pioneers of the theory of computation.

- The *forward instance construction* is an algorithm that takes an instance y of problem Y and produces an instance x^y of problem X such that the optimal solutions of both instances to have the same objective value. This construction must have polynomial runtime.

- The *backward solution construction* is an algorithm takes an optimal solution $\text{OPT}(x^y)$ for instance x^y of problem X and produces a solution for instance y of problem Y with value at least $\text{OPT}(x^y)$. This construction must have polynomial runtime if the solution to the original problem Y is desired.

- The *forward solution construction* is an algorithm that takes an optimal solution $\text{OPT}(y)$ for instance y of problem Y and produces a solution for instance x^y of problem X with value at least $\text{OPT}(y)$. The runtime of this construction is not important.

The following theorem formally proves that the existence of the backward and forward solution construction are sufficient to imply the correctness of the reduction given by forward instance construction.

Theorem 1. *The reduction from problem Y to problem X given by a forward instance construction (and then running a black box for X on the instance produced) is correct if there there exists reverse and forward solution constructions that are correct.*

Proof. By the correctness of the forward instance construction, the value of the solution to x^y that is constructed from a solution to y is at least $\text{OPT}(y)$. Thus, $\text{OPT}(x^y) \geq \text{OPT}(y)$. By the correctness of the backward solution construction, the value of the solution to y that is constructed from a solution to x^y is at least $\text{OPT}(x^y)$. Thus, $\text{OPT}(y) \geq \text{OPT}(x^y)$. Combining these observations, $\text{OPT}(x^y) = \text{OPT}(y)$. □

Notice that the discussion above has assumed that just the value of the optimal solution to instance y of problem Y is to be computed. If the black box for problem X computes its optimal solution (and not only the solution's value), then the optimal solution to y can be computed as well. Simply run the backwards solution construction on the optimal solution computed by the black box for X. Its result is the optimal solution to y.

Notice that every instance y of problem Y has a corresponding instance x^y of problem X per the construction. The opposite is not generally true. There will generally be instances x of X that do not have corresponding instances of problem Y. Consequently, while the forward solution construction must be able to be applied to any solution to instance y of Y, the backward certificate construction does not need to have any prespecified behavior on solutions to general instances x of X that are not generated by the forward instance construction on an instance y of Y.

Rubrics are given with the reduction in Section 2.

1 Reductions for Optimization Problems

Reductions can be used to show that a family of problems Y is tractable by *reducing to* a family of problems X which is known to be tractable. A canonical example of such a reduction is the reduction from bipartite matching to integral network flow.

- The *network flow* problem (V, E, c, s, t) has as input a directed graph (V, E), edge capacities $c : E \to \mathbb{R}$, source vertex s, sink vertex t. A flow $f : E \to \mathbb{R}$ is feasible if it satisfies (a) *capacity constraints*, i.e., $0 \leq f(e) \leq c(e)$, and (b) *conservation constraints*, i.e., for all $v \neq s, t$, the flow into v equals the flow out of v, i.e., $\sum_{\{u:(u,v)\in E\}} f(u, v) = \sum_{\{u:(v,u)\in E\}} f(v, u)$. The value of a flow is the total flow from s to t, denoted $|f|$, which is equal to the flow out of s, i.e., $\sum_{\{u:(s,u)\in E\}} f(s, u)$. The desired output is a feasible flow with the maximum value.

  The integral network flow problem additionally assumes that the capacities on each edge be an integer and requires that the flow on each edge be an integer.

- The *bipartite matching* problem (A, B, E) has as input a bipartite graph (A, B, E). The vertices in a bipartite graph are $A \cup B$ and the edges are between vertices $u \in A$ and $v \in B$. A matching $M \subset E$ is a set of edges such that each vertex $A \cup B$ has at most one incident edge. The desired output is a matching with the maximum cardinality.

We will solve the bipartite matching problem by reduction to network flow. This reduction is motivated by the following theorem which shows that network flow is tractable.

Theorem 2. *There is an algorithm that computes the maximum flow f in a network flow instance (V, E, c, s, t) in polynomial time. If capacities c are integral, then the computed flow f is integral.*

Part I: Forward Instance Construction

To show that the bipartite matching problem is tractable, we will *reduce to* the integral network flow problem. Specifically, we will exhibit an algorithm to solve the bipartite matching problem using a black box that solves the integral network flow problem. A one-call reduction, as outlined above, converts any instance $x = (A, B, E)$ of the bipartite perfect matching to an instance $y^x = (V^x, E^x, c^x, s^x, t^x)$ of network flow problem such that optimal flows in the flow problem y^x can be related to optimal matchings in the matching problem x.

The main task in coming up with such a construction is mapping the constraints of one problem onto constraints of the other problem. The constraints on matchings $M \subset E$ in a bipartite matching instance (A, B, E) are:

- Each vertex v in A and B has at most one incident edge in M.

The constraints on the flow f^x of a flow in the network flow problem $(V^x, E^x, c^x, s^x, t^x)$ are:

- capacity constraints at each vertex $v \neq s^x, t^x$, and
- conservation constraints on each edge $e \in E^x$.

The basic idea of this reduction is to create a new vertex s^x as the source for the network flow, connect this vertex to each vertex u in A with an edge with capacity 1, and put capacity 1 on all outgoing edges from u in A to v in B. The edge from s^x to $u \in A$ allows at most one unit of flow via u. Conservation of flow, then, requires that at most one unit of flow leaves each u in A for B, integrality of the flow requires that all of this flow leaves v on a single edge. The edges between A and B that have flow on them can be selected in a matching. To be formally proved below, the maximum flow will correspond to the maximum matching.

Forward instance construction: from bipartite matching (A, B, E) to network flow problem $(V^x, E^x, c^x, s^x, t^x)$.

1. Set graph (V^x, E^x) with
   - vertices $V^x = A \cup B \cup \{s^x, t^x\}$,
   - edges $E^x = E \cup \{(s^x, u) : u \in A\} \cup \{(v, t^x) : v \in B\}$
2. Set capacities $c^x(e) = 1$ for all $e \in E^x$.

Runtime Analysis: $O(n + m)$ where $n = |A \cup B|$ and $m = |E|$.

Correctness of this construction is proven via the two algorithms given in the subsequent parts.

Part II: Backward Solution Construction

The backward solution construction, for the reduction from bipartite matching to the integral network flow problem, constructs from any flow f^x a matching M with cardinality equal to the value of the flow.

Backward certificate construction: from network flow f^x for integral network flow instance $y^x = (V^x, E^x, c^x, s^x, t^x)$ to matching M for bipartite matching instance $x = (A, B, E)$.

1. Set $M = \{(u, v) : u \in A, v \in B, f^x(u, v) = 1\}$.

Runtime analysis: $O(m)$ where $m = |E|$.

Proof of correctness:

Claim 1. *If f^x is a flow in $(V^x, E^x, c^x, s^x, t^x)$ then M from the backward solution construction is matching in (A, B, E) with cardinality $|M|$ equal to the value of the flow f^x.*

Proof. The flow f^x is integral and the capacities on edges out of s^x are unit, thus each $v \in A$ receives either 0 or 1 unit of flow from s^x. Let A' be the set of vertices in A that receive 1 unit of flow. The cardinality of A' is the total flow out of s^x which, by definition, is the value $|f^x|$ of the flow f^x. Conservation and integrality of flow implies that the unit of flow received at any vertex $u \in A'$ leaves for a single vertex $v \in B$. Since the capacity of the (v, t^x) edge is 1, conservation and capacity of flow implies that the only flow into this vertex v is from u. By definition, this edge (u, v) is selected in M. As u sends and v receives only a single unit of flow each has only one edge in M and, thus, M is a matching. The cardinality of M is equal to the cardinality of A' and the value of the flow f^x. □

Part III: Forward Solution Construction

The forward solution construction, for the reduction from bipartite matching to integral network flow constructs, from any matching M an integral flow f^x with value $|M|$.

Forward solution construction: from matching M for bipartite matching instance (A, B, E) to flow f^x in integral network flow problem (G^x, c^x, s^x, t^x).

1. For $e = (u, v)$ with $u \in A$ and $v \in B$, set $f^x(e) = \begin{cases} 1 & \text{if } e \in M, \\ 0 & \text{otherwise.} \end{cases}$
2. For $e = (s^x, u)$ with $u \in A$, set $f^x(e)$ to the flow out of u from Step 1.
3. For $e = (v, t)$ with $v \in B$, set $f^x(e)$ to the flow in to v from Step 1.

To prove this construction correct, we must show that the construction applied to any solution to bipartite matching gives a solution to the corresponding network flow problem with at least the value of the matching.

Proof of correctness:

Claim 2. *If M is a matching in $x = (A, B, E)$ then f^x from the forward solution construction is a flow with value at least $|M|$ in $y^x = (V^x, E^x, c^x, s^x, t^x)$.*

Proof. We begin by arguing that f^x is a feasible flow, i.e., it satisfies capacity and conservation of flow:

1. The capacity constraints on edges (u, v) for $u \in A$ and $v \in B$ are satisfied by the flow from Step 1.
2. Since M is a matching, each vertex $u \in A$ has outgoing flow at most 1. Thus, from Step 2, the incoming flow to u from from s^x on edge (s^x, u) is at most 1 and satisfies the capacity constraint of edge (s^x, u).
3. Analogously the capacity constraints on edges from $v \in B$ to t^x are satisfied.
4. Steps 2 and 3 guarantee conservation of flow at each vertex $u \in A$ and $v \in B$, respectively.

We now argue that the value of f^x, i.e., the total flow out of s^x, is $|M|$. The total flow from s^x into vertices in A, by conservation of flow, is equal to the total flow from vertices in A to vertices in B which, by Step 1, is equal to $|M|$. □

2 Reductions for Decision Problems

A decision problem is one where the desired answer is either yes or no. For example, determining whether or not there is a perfect matching in a bipartite graph is a decision problem. (Recall, a perfect matching is one where every vertex is matched.) Optimization problems can also be recast as decision problems. For example, determining whether there is a flow in a network flow graph with at least a given target value θ is a decision problem. A yes instance is one where the there is a flow with value at least the target, a no instance is one where there is no such flow. For many decision problems, yes instances have corresponding solutions that certify that they are yes instances. For example, a perfect matching M in a bipartite graph certifies that the graph is a yes instance for the perfect matching problem; a flow with value greater than target θ certifies that the flow graph admits such a flow. Notice that there is not typically such a solution that certifies a no instance.

- The *network flow decision problem* (V, E, c, s, t, k) is the network flow problem with an additional parameter θ; it asks whether there exists a flow with value at least θ.
- The *bipartite perfect matching problem* (A, B, E) asks whether there exists a matching in bipartite graph (A, B, E) in which all vertices are matched.

The exposition of the previous section referred to the value of a solution. For example, in bipartite matching the value of a solution (i.e., a matching) is

the cardinality of the matching, in network flow the value of a solution (i.e., a flow) is the value of the flow. For decision problems the value of a solution is 1 if it satisfies the criteria of yes instances and 0 if it does not. For decision problems the forward instance construction must satisfy the property that y is a yes instance of Y if and only if the constructed instance x^y is a yes instance of X.

The "solution" to a yes instance of a decision problem can be viewed as a certificate that proves that it is a yes instance. For example, if you want to prove that a bipartite graph has a perfect matching, an straightforward way to do that is to exhibit the matching; if you want to prove that a flow graph admits a flow with value at least target θ, simply give the flow. On the other hand, for no instances there is no solution to give. Thus, there is a natural asymmetry between yes and no instances of decision problems. When considering decision problems, we will refer to any object by which we can easily verify that an instance is a yes instance as a *certificate*. When we are considering decision problems we will refer more generally to the "solution constructions" discussed previously as *certificate constructions*.

For forward and backward certificate constructions, recall that we are looking for a transformation of the optimal solution of an instance of one problem to a solution to an instance of the other problem that has at least the value of the optimal solution to the original problem instance. If the original problem instance is a no instance then no constraint is imposed by this requirement. Thus, for decision problems, it is sufficient to check that certificates to yes instances of one problem are transformed to certificates that prove that the corresponding instance of the other problem is also a yes instance.

If the goal of the problem Y is simply to determine whether it is a yes or no instance (and the solution or certificate that proves that the instance is a yes instance is not needed) then only the forward instance construction need be polynomial time. Moreover, neither the forward nor backward certificate constructions need be polynomial time algorithms.

Part I: Forward Instance Construction

To show that the the bipartite perfect matching problem is tractable, we will *reduce to* the integral network flow decision problem. Specifically, we will exhibit an algorithm to solve the bipartite matching problem using a black box that solves the integral network flow problem. A one-call reduction, as outlined above, converts any instance $x = (A, B, E)$ of the bipartite perfect matching problem to an instance $y^x = (V^x, E^x, c^x, s^x, t^x, \theta^x)$ of integral network flow decision problem such that (A, B, E) is a yes instance if and only if $(V^x, E^x, c^x, s^x, t^x, \theta^x)$ is a yes instance.

As in the reduction from the bipartite matching (optimization) problem to the integral flow (optimization) problem, the main task in coming up with such a construction is mapping the constraints of yes instances of one problem onto constraints of yes instances of the other problem. Relative to that reduction, we have the added constraint that the matching should be perfect (i.e., all vertices

are matched), and the added constraint that the flow is at least the target value. We will employ the same reduction and additionally transform the constraint that the matching is perfect to the constraint that the flow has value at least $|A| = |B|$.

Forward instance construction: from bipartite perfect matching $x = (A, B, E)$ to network flow decision problem $y^x = (V^x, E^x, c^x, s^x, t^x, \theta^x)$.

1. Set target flow value to $\theta^x = |A| = |B|$
2. Set graph (V^x, E^x) with
   - vertices $V^x = A \cup B \cup \{s^x, t^x\}$,
   - edges $E^x = E \cup \{(s^x, u) : u \in A\} \cup \{(v, t^x) : v \in B\}$
3. Set capacities $c^x(e) = 1$ for all $e \in E^x$.

Runtime Analysis: $O(n + m)$ where $n = |A| = |B|$ and $m = |E|$.

To prove this construction correct, we must show that a bipartite perfect matching instance is a yes instance if and only if the constructed network flow decision problem instance is a yes instance. The proof of this if and only if is given by the subsequent parts.

Rubric I. a. *The direction of the reduction is correct. To show Y is tractable, reduce from Y to tractable problem X. To show X is intractable, reduce from intractable problem Y to X.*

Rubric I. b. *The forward instance construction gives an instance x^y of problem X from any instance y of problem Y.*

Rubric I. c. *The runtime of the forward instance construction is correctly analyzed.*

Part II: Backward Certificate Construction

The backward certificate construction, for the reduction from perfect bipartite matching to the network flow decision problem, constructs, from any integral flow f^x with value θ^x, which certifies that the network flow instance is a yes instance, a perfect matching M which certifies that the bipartite matching instance is a yes instance.

Backward certificate construction: from network flow f^x for network flow instance $(V^x, E^x, c^x, s^x, t^x, \theta^x)$ to matching M for bipartite matching instance (A, B, E).

1. Set $M = \{(u, v) : u \in A, v \in B, f^x(u, v) = 1\}$.

To prove this construction correct, we must show that for any bipartite perfect matching instance the construction applied to the solution that certifies a yes instance of the corresponding network flow decision problem gives a solution that certifies that the original bipartite perfect matching instance is a yes instance.

Proof of correctness:

Claim 3. *If f^x is a flow with value at least θ^x in $(V^x, E^x, c^x, s^x, t^x, \theta^x)$ then M from the backward certificate construction is a perfect matching in (A, B, E).*

Proof. We begin by arguing that M is a matching, i.e., each vertex $u \in A$ and $v \in B$ has at most one incident edge in M.

1. The capacity constraint on edge (s^x, u) implies at most one unit of flow enters u. The conservation constraint at vertex u then implies that at most one unit of flow leaves u. Vertex u has an incident edge in M by the reverse instance construction if the edge has one unit of flow on it. Thus, u has at most one incident edge in M.

2. Analogously $v \in B$ has at most one incident edge in M.

We now argue that M is a perfect matching. Since f^x has value $\theta^x = |A| = |B|$, each edge out of s^x has exactly one unit of flow, conservation implies that each vertex $u \in A$ has exactly one unit of flow out, the integrality of the flow implies that one edge out of u has one unit of flow, and this edge is included in the matching. Consequently every vertex in $u \in A$ is matched, likewise for $v \in B$, and M is a perfect matching. □

Rubric II. a. *The backward certificate construction is well defined and gives a certificate for y from any yes-instance certificate for x^y.*

Rubric II. b. *The proof of correctness of the backward certificate construction shows how the constraints of instance x^y imply the satisfaction of constraints of instance y.*

Rubric II. c. *The proof of correctness of the backward certificate construction is correct.*

Part III: Forward Certificate Construction

A perfect matching M in bipartite graph (A, B, E) certifies that a bipartite matching instance (A, B, E) is a yes instance. A integral flow f^x with value at least θ^x certifies that the integral network flow decision problem instance $(G^x, c^x, s^x, t^x, \theta^x)$ is a yes instance. For the reduction from the perfect bipartite

matching problem to the network flow decision problem, the forward certificate construction constructs such a flow f^x from such a perfect matching M.

Forward certificate construction: from matching M for bipartite matching instance (A, B, E) to flow f^x in network flow decision problem $(G^x, c^x, s^x, t^x, \theta^x)$.

1. For $e = (u, v)$ with $u \in A$ and $v \in B$, set $f^x(e) = \begin{cases} 1 & \text{if } e \in M, \\ 0 & \text{otherwise.} \end{cases}$
2. For $e = (s^x, u)$ with $u \in A$, set $f^x(e)$ to the flow out of u from Step 1.
3. For $e = (v, t^x)$ with $v \in B$, set $f^x(e)$ to the flow in to v from Step 1.

To prove this construction correct, we must show that the construction applied to the solution that certifies a yes instance of bipartite perfect matching gives a solution that certifies that the corresponding network flow decision problem is also a yes instance.

Proof of correctness:

Claim 4. *If M is a perfect matching in (A, B, E) then f^x from the forward certificate construction is a flow with value at least θ^x in $(V^x, E^x, c^x, s^x, t^x, \theta^x)$.*

Proof. We begin by arguing that f^x is a feasible flow, i.e., it satisfies capacity and conservation of flow:

1. The capacity constraints on edges (u, v) for $u \in A$ and $v \in B$ are satisfied by the flow from Step 1.
2. Since M is a matching, each vertex $u \in A$ has outgoing flow at most 1. Thus, from Step 2, the incoming flow to u from from s^x on edge (s^x, u) is at most 1 and satisfies the capacity constraint of edge (s^x, u).
3. Analogously the capacity constraints on edges from $v \in B$ to t^x are satisfied.
4. Steps 2 and 3 guarantee conservation of flow at each vertex $u \in A$ and $v \in B$, respectively.

We now argue that the value of f^x is at least θ^x (in fact, that it is exactly θ^x). Since M is a perfect matching, $|M| = |A| = |B| = \theta^x$. The total flow from s^x into vertices in A, by conservation of flow, is equal to the total flow from vertices in A to vertices in B which is equal to $|M|$ by Step 1. Thus, the flow from s^x, which is the value of the flow, is θ^x as desired. □

Rubric III. a. *The forward certificate construction is well defined and gives a certificate for y from any yes-instance certificate for x^y.*

Rubric III. b. *The proof of correctness of the forward certificate construction shows how the constraints of instance y imply the satisfaction of constraints of instance x^y.*

Rubric III. c. *The proof of correctness of the forward certificate construction is correct.*

3 Reductions to Show Intractability

NP hardness is the main approach for demonstrating computational intractability. If a problem is NP hard then the existence of a polynomial time algorithm for the problem would imply the existence of polynomial time algorithms that solve all other NP problems. It is widely believed that these algorithms do not exist, thus NP hardness implies the wide belief that there does not exist a polynomial time algorithm for the given problem.

Demonstrating intractability directly, i.e., proving that there do not exist polynomial time algorithms that solve a given problem, is difficult. Showing such a non-existence is difficult because it requires quantifying over all algorithms and the space of algorithms is complex. In contrast showing that a problem is tractable requires just demonstrating the existence of an algorithm. While the field of computational complexity has had little success in showing that polynomial time algorithms do not exist, the field of algorithms has had great success identifying polynomial time algorithms. An NP-hardness reduction is an algorithm; thus the approach of NP-hardness reductions replaces the hard task of showing non-existence of algorithms with the easier task of showing the existence of an algorithm.

Suppose we wish to show that a given problem X is hard. The approach of NP-hardness reductions is to identify an NP-hard problem Y and show that the existence of a polynomial time algorithm for X implies the existence of a polynomial time algorithm for Y. This existence is shown by reduction. Specifically, an NP-hardness reduction is an algorithm for problem Y that employs an algorithm for problem X as a black box. This method for showing that a problem is hard is essentially a proof by contradiction. Assume for contradiction that there is a polynomial time algorithm for problem X, then the reduction, an algorithm for solving Y in polynomial time that uses the algorithm that solves X in polynomial time, would imply a polynomial time algorithm for Y. Since Y is NP hard, we do not believe there is a polynomial time algorithm for it, thus, we must also not believe the assumption that there is a polynomial time algorithm for X.

NP is the class of decision problems for which yes instances have short certificates. Here are several classic NP problems:

- The *traveling salesperson problem* (V, E, c, k) has as input a complete graph (V, E), edge weights $c : E \to \mathbb{R}$, and target tour length θ. A

yes instance is one where there exists a tour of the vertices, i.e., a path that visits each vertex exactly once and finishes at the vertex it starts with, such that the total cost of the edges in the tour is at most θ. A no instance has no such tour.

- The *independent set problem* (V, E) has as input a graph (V, E) and target independent set size θ. A yes instance is one where there exists a subset of vertices S such that no pair of vertices in the subset is connected by an edge, i.e., $\forall u, v \in S$, $(u, v) \notin E$, and the cardinality of the subset S is at least the target θ. A no instance has no such subset.

- The *satisfiability problem* has as input a n variable boolean formula f. This formula is assumed to be in *conjunctive normal form*: it is the conjunction (i.e., and) of m clauses, each clause is the disjunction (i.e., or) of three literals, and each literal is a variable or its negation. Denoting the jth literal of the ith clause by ℓ_{ij} the formula is then

$$f(\mathbf{x}) = \bigwedge_{i=1}^{m} (\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$$

  where each literal ℓ_{ij} is either x_k or $\neg x_k$ (i.e., the boolean negation of x_k). A yes instance of the satisfiability problem has an assignment of boolean values true and false to the variables $\mathbf{x} = (x_1, \ldots, x_n)$ such that the formula $f(\mathbf{x})$ evaluates true. A formula in conjunctive normal form evaluates to true on an assignment of variables if and only if there is at least one true literal per clause.

In many NP problems the short certificate from which yes instances can be easily verified is explicit in the definition of the problem. In the traveling salesperson problem the certificate is the tour. Given the tour it is easy to check that (a) it is a tour and (b) the total cost of the edges in the tour is at most the target C. Similarly, in the independent set problem the certificate is the set S, and in the satisfiability problem the certificate is the satisfying assignment $\mathbf{x}$ of boolean values to the variables.

The independent set problem will be a running example. Assume for the purpose of the discussion that it is unknown whether the independent set problem is NP hard, but known that the satisfiability problem is NP hard. We will prove that independent set is NP hard by reduction from the satisfiability problem.

Part I: Forward Instance Construction

To show that the independent set problem is NP hard, we will *reduce from* the satisfiability problem. Specifically, we will exhibit an algorithm to solve the satisfiability problem using a black box that solves the independent set problem. The direct approach, as outlined above, is to convert any instance f of the satisfiability problem to an instance (V, E, k) of the independent set problem such that f is a yes instance if any only if (V, E, k) is a yes instance.

The main task in coming up with such a construction is mapping the constraints of yes instances of one problem onto constraints of yes instances of the other problem. The constraints on a certificate $\mathbf{x}$ of yes a instance of the satisfiability problem are:

- Each of the m clauses has at least one true literal.
- All positive occurrences of any variable as a literal have the same boolean value and all negative occurrences of the variable have the opposite boolean value (to the positive occurrences).

The constraints on a certificate S of a yes instance of the independent set problem are:

- There are at least θ vertices in S.
- For any pair of vertices connected by an edge, at most one is in S.

At a high-level the construction of an independent set instance from a satisfiability instance will equate the true literal in each clause with the vertices in the independent set. In the construction, there will be a satisfying assignment of f if and only if there is an independent set of size at least m, the number of clauses. Consider the independent set instance given by target set size $\theta^f = m$ and the graph (V^f, E^f) with vertices V^f and edges E^f defined as follows. There are $3\,m$ vertices V^f that correspond to the literals in the satisfiability formula f, i.e., for each literal ℓ_{ij} there is a vertex v_{ij}. The edges are $E^f = E^f_{\text{clause}} \cup E^f_{\text{variable}}$ as follows:

- *Clause edges*, E^f_{clause}, connect all vertices that correspond to literals in the same clause have edges between them. As there are three literals per clause, these edges alone give a graph that is a collection of m triangles.

  Notice that at most one vertex from each triangle can be in any independent set. Thus, an independent set of size at least $k = m$ in the graph $(V^f, E^f_{\text{clause}})$ selects exactly one vertex in each of the m triangles that correspond to clauses, as the edges in each triangle prevent selecting multiple vertices in the same triangle. The size of any such independent set is exactly $k = m$.

- *Variable edges*, E^f_{variable}, connect vertices that correspond to literals that cannot both by true, specifically, that correspond to a variable and its negation. Let V_{x_k} and $V_{\neg x_k}$ correspond to the subset of vertices that correspond to x_k and $\neg x_k$ literals, respectively. Thus, variable edges connect each $u \in V_{x_k}$ to each $u \in V_{\neg x_k}$ for each variable x_k.

Forward instance construction: from satisfiability instance f to independent set instance (V^f, E^f, θ^f).

1. Set target independent set size $\theta^f = m$.

2. Set graph (V^f, E^f) with
   - vertices $V^f = \{v_{ij} : i \in \{1, \ldots, m\} \vee j \in \{1, 2, 3\}\}$,
   - clause edges $E^f_{\text{clause}} = \bigcup_i \{(v_{i1}, v_{i2}), (v_{i2}, v_{i3}), (v_{i3}, v_{i1})\}$,
   - variable edges $E^f_{\text{variable}} = \{(u, v) : k \in \{1, \ldots, n\} \wedge u \in V_{x_k} \wedge v \in V_{\neg x_k}\}$ (with V_{x_k} and $V_{\neg x_k}$ defined above), and
   - edges $E^f = E^f_{\text{clause}} \cup E^f_{\text{variable}}$.

Runtime Analysis: $O(n + m)$ where n is number of variables and m is number of clauses in formula f.

To prove this construction correct, we must show that a satisfiability instance f is a yes instance if and only if the constructed independent set instance (V^f, E^f, θ^f) is a yes instance. The proof of this if and only if is given by the existence of the two algorithms given below.

Part II: Backward Certificate Construction

The backward certificate construction, for the independent set reduction from satisfiability, constructs, from any yes instance certificate S for the independent set instance (V^f, E^f, θ^f), a yes instance certificate $\mathbf{x}$ for the satisfiability instance f.

Backward certificate construction: from independent set S^f for independent set instance (V^f, E^f, θ^f) to assignment $\mathbf{x}$ for satisfiability instance f.
For each variable x_k ($k \in \{1, \ldots, n\}$):

1. Set x_k to true if literal ℓ_{ij} is x_k (not negated) and the corresponding vertex v_{ij} is in S^f for some ij.
2. Set x_k to false otherwise.

In the proof of correctness, recall that vertices in the graph (V^f, E^f) correspond to literals in the formula f.

Proof of correctness:

Claim 5. *If S^f is a yes instance certificate for (V^f, E^f, θ^f) then the constructed* $\mathbf{x}$ *is a yes instance certificate for satisfiability problem f.*

Proof. The assignment $\mathbf{x}$ is a valid assignment as each variable x_k is considered once and set to either true or false. Assume that S^f certifies that (V^f, E^f, θ^f) is a yes instance.

1. Because of the clause edges, S^f can contain at most one vertex per clause. Because there are m clauses and S^f has cardinality at least $\theta^k = m$, there must be exactly one vertex per clause in S^f.

2. Consider any clause j for which the selected vertex $v_{ij} \in S^f$ corresponds to a literal that is not negated, i.e., ℓ_{ij} is x_k for some k; by construction this clause is true because this literal is set to true in $\mathbf{x}$.

3. Consider any clause j for which the selected vertex $v_{ij} \in S^f$ corresponds to a literal that is negated i.e., ℓ_{ij} is $\neg x_k$ for some k. By the construction of the variable edges of the independent set graph (V^f, E^f) there are edges between v_{ij} and all vertices V_{x_k} that correspond to literals where x_k appears positively. Since $v_{ij} \in S^f$ and S^f is an independent set, there can be no vertex in V_{x_k} that is also in S^f, thus, the construction sets x_k to false and the literal $\neg x_k$ and the clause itself is true.

In conclusion, under the assignment $\mathbf{x}$ all clauses are true and, thus, $\mathbf{x}$ is a certificate that demonstrates that f is a yes instance. □

Part III: Forward Certificate Construction

The forward certificate construction, for the independent set reduction from satisfiability, constructs, from any yes instance certificate $\mathbf{x} = (x_1, \ldots, x_n)$ for the satisfiability instance f, a yes instance certificate S for the constructed instance of independent set (V^f, E^f, θ^f).

Forward certificate construction: from assignment $\mathbf{x}$ for satisfiability instance f to independent set S for independent set instance (V^f, E^f, θ^f).

1. Initialize $S = \emptyset$.

2. For each clause i, add the vertex v_{ij} to S that corresponds to the first (smallest j) true literal ℓ_{ij} (according to $\mathbf{x}$) in the clause.

Proof of correctness:

Claim 6. *If* $\mathbf{x}$ *is a yes instance certificate for satisfiability problem* f *then the constructed* S^f *is a yes instance certificate for independent set problem* (V^f, E^f, θ^f).

Proof. We argue that S^f has the correct cardinality, and that neither clause edges nor variable edges are violated.

1. Since there is one true literal in each clause, the selected set S^f has cardinality $m = \theta^f$.

2. Since only one vertex is selected corresponding to each clause, the clause edges are satisfied. I.e., for any pair of vertices $u, v \in S^f$, edge (u, v) is not in E^f_{clause}.

3. Since we have chosen vertices corresponding to a satisfying assignment, the variable edges are satisfied. Specifically, any pair of vertices $u, v \in S^f$ correspond to literals that are not a variable x_i and its negation $\neg x_i$ and the edge (u, v) is not in E^f_{variable}.

In conclusion, the constructed set S^f is independent and has cardinality θ^f, i.e., S^f is a certificate that demonstrates that (V^f, E^f, θ^f) is a yes instance. □

4 Reductions: Summary of Rubric

Rubric I. a. *The direction of the reduction is correct. To show Y is tractable, reduce from Y to tractable problem X. To show X is intractable, reduce from intractable problem Y to X.*

Rubric I. b. *The forward instance construction gives an instance x^y of problem X from any instance y of problem Y.*

Rubric I. c. *The runtime of the forward instance construction is correctly analyzed.*

Rubric II. a. *The backward certificate construction is well defined and gives a certificate for y from any yes-instance certificate for x^y.*

Rubric II. b. *The proof of correctness of the backward certificate construction shows how the constraints of instance x^y imply the satisfaction of constraints of instance y.*

Rubric II. c. *The proof of correctness of the backward certificate construction is correct.*

Rubric III. a. *The forward certificate construction is well defined and gives a certificate for y from any yes-instance certificate for x^y.*

Rubric III. b. *The proof of correctness of the forward certificate construction shows how the constraints of instance y imply the satisfaction of constraints of instance x^y.*

Rubric III. c. *The proof of correctness of the forward certificate construction is correct.*