AN INTRODUCTION TO NP-COMPLETE PROBLEMS

JAMES ZHOU

Abstract. We introduce the concept of NP-complete decision problems and the question of P versus NP through a series of working definitions and examples. We explore the concept of NP-completeness further by proving that several problems are NP-complete through a series of reductions. Finally, we examine search and optimization problems related to these NP-complete decision problems.

Contents

1. Introduction

The time complexity of computational problems is of major interest in both theoretical and practical applications. In particular, many problems of interest such as satisfiability and traveling salesman reside in a large class known as NP.

Despite their importance, many of these problems do not have known efficient algorithms. A longstanding question is whether or not there exist efficient, polynomial algorithms for these problems. Problems with such efficient algorithms are said to be a class P, and the question is more succinctly stated as whether P equals NP or not.

We aim to introduce this question to the reader. In the first section, we describe definitions of P and NP problems as well as introduce a selection of notable problems taken from Dasgupta's *Algorithms* [4]. In the second section, we go through a process of reductions to further elaborate on these problems akin to pioneering work by Richard Karp [5]. Finally, in the last section we examine search and optimization, problems related to those in NP despite not technically belonging to that class. While our approach lacks concepts used in formal treatments of the subject, it should still capture the general ideas.

2. NP-Complete Problems

Definition 2.1. A problem is a set of instances, each of which has a set of solutions.

An algorithm that solves a problem inputs an instance of the problem and outputs either a solution or that there are none.

Example 2.2 (Euler Path)**.** Given a graph G, find a path in G containing every edge exactly once. Such a path is known as an Euler path.

In this example, an instance would be a graph, and a solution would be a path meeting the requirements given in the problem.

Definition 2.3. A decision problem is a problem such that the solution to any instance must be yes or no.

Example 2.4. Given a graph G, determine the existence of a path in G containing every edge exactly once.

This is a decision version of Euler path. The instance is the same, but the solutions have been changed to yes or no.

The original problem of finding the Euler path is the search version of the problem. The solutions to search problems for a particular instance are objects meeting the requirements of the problem for that instance. Search and decision are closely related, seeing as any search problem can be reframed as a decision problem of existence, and we will return to their relationship in Section 4.

Definition 2.5. A problem is in the class NP if it is a decision problem such that there exists an algorithm that can verify any proposed solution to the corresponding search problem in polynomial time.

NP stands for non-deterministic polynomial.

Theorem 2.6. *Euler path is in NP.*

Proof. Consider any path in a graph G. Simply check every edge; the path is a solution if and only if every edge is in the path.

This procedure is $O(m)$, where m is the number of edges, so such an algorithm is polynomial. $\square$

Most proofs that a problem is in NP are similar basic checking procedures, so we will neglect them from here on.

Definition 2.7. A problem is in the class P if it is a decision problem such that there exists an algorithm that can determine a solution to the corresponding search problem in polynomial time.

P stands for polynomial.

Theorem 2.8. *Euler path is in P.*

The desired algorithm relies on the fact that a graph contains a Euler path if and only if exactly zero or two vertices of the graph have odd degree. We will not provide a full proof here, since it does not contribute to our understanding of the relationship between P and NP.

Note that every problem in P is also in NP. However, as stated before the question of whether P = NP or P $\neq$ NP is unknown.

Despite this, there remain ways of approaching problems in NP and other related problems by examining their relative complexity.

Definition 2.9. Let A and B be two problems. We say that A reduces to B if, given an algorithm solving B in polynomial time, there exists an algorithm that solves A in polynomial time.

Definition 2.10. If any problem in NP reduces to a problem A, then A is NP-hard.

In terms of this definition, we can say that if there exists a polynomial time algorithm for an NP-hard problem, then P = NP.

The converse is not necessarily true, in that an NP-hard problem does not necessarily reduce to any problem in NP.

Definition 2.11. A problem is NP-complete if it is both NP and NP-hard.

Thus, NP-complete problems reduce to each other, and can be considered in certain ways to be equivalent. Moreover, if P = NP, then P = NP = NP-complete.

Here follows an incomplete list of several seemingly unrelated problems known to be NP-complete. Proof of NP-completeness will follow in the Section 3.

Definition 2.12 (Boolean Satisfiability or SAT)**.** Given a Boolean formula, determine the existence of an assignment of variables as true or false that satisfies the formula.

Often, we express SAT in conjunctive normal form, where we have variables and negations of variables, called literals, organized in clauses. Literals within clauses are joined by OR operators; clauses are joined by AND operators. A clause cannot contain both a variable and its negation.

Example 2.13 (SAT)**.**

$$(x \vee y \vee z)(x \vee \neg y \vee \neg z)(\neg x \vee y \vee z)(\neg x \vee \neg z).$$

Clauses are indicated by parentheses. AND operators are omitted for brevity. In this instance, the solution is yes since $\neg x, \neg y, z$ satisfies the instance.

Definition 2.14 (3SAT)**.** Given a Boolean formula in conjunctive normal form such that each clause has at most three literals, determine the existence of an assignment of variables as true or false that satisfies the formula. A special case of SAT.

Definition 2.15 (Independent Set)**.** Given a graph G and budget b, determine the existence of an independent set of b vertices of G.

Definition 2.16 (Three-Dimensional Matching or 3D Matching)**.** Given three disjoint sets X, Y, and Z such that $|X| = |Y| = |Z| = n$ and a set of valid triples $T \subseteq X \times Y \times Z$, determine the existence of a perfect matching. A perfect matching is a set U of n triples in T such that every $x \in X$, $y \in Y$, and $z \in Z$ is part of some triple in U.

This problem has a general version where the cardinalities of X, Y, and Z are not equal, but this special case suffices for our purposes. A later result will show the NP-completeness of the general version.

Definition 2.17 (Set Cover)**.** Given a set S, subsets $E_1, \ldots, E_n$, and budget b, determine the existence of a cover $\{E_i \mid i \in \lambda\}$ of S, where $\lambda \subseteq \{1, \ldots, n\}$ and $|\lambda| = b$.

Definition 2.18 (Vertex Cover)**.** Given a graph G with n vertices and budget b, determine the existence of a set of b vertices covering all edges of G. A special case of set cover. The set S is the set of all edges of G; a subset E_i of S, for $1 \leq i \leq n$, is the set of all edges incident on vertex i, and the budget is b.

Definition 2.19 (Hamiltonian Cycle)**.** Given a graph G, determine the existence of a cycle containing all vertices in G. Such a cycle is known as a Rudrata or Hamiltonian cycle.

This problem has other variants that reduce to each other, such as one asking for a Hamiltonian path or one involving directed graphs.

Definition 2.20 (Traveling Salesman or TSP)**.** Given a complete weighted graph G and budget b, determine the existence of a Hamiltonian cycle in G such that the sum of edge weights in the cycle is less than b.

Definition 2.21 (Integer Linear Programming or ILP)**.** Given a system of linear equations and inequalities, determine the existence of an assignment of variables such that all of the equations and inequalities are satisfied.

Definition 2.22 (Zero-One Equations or ZOE)**.** Given an $m \times n$ matrix $\mathbf{A}$ of 0's and 1's, find a vector $\mathbf{x}$ of 0's and 1's such that $\mathbf{xA} = \mathbf{1}$. A special case of ILP. The variables correspond to the elements in $\mathbf{x}$ and coefficients of the equations correspond to the rows in A.

We will work with the ILP form of ZOE. For both problems, we omit variables that are not part of any equation.

3. Reductions

We will prove that these problems are NP-complete by starting with a prototypical NP-complete problem and reducing to the others as thus:

(1) SAT (Prototypical NP-Complete Problem)
  - (I) 3SAT
    - (A) Independent Set
      - (i) Vertex Cover
        - (a) Set Cover
    - (B) 3D Matching
      - (i) ZOE
        - (a) ILP
        - (b) Hamiltonian Cycle
          - (1) TSP

Problems reduce to those nested underneath them.

To prove that A reduces to B, we do three things:

(1) Find a transformation for an instance of A to an instance of B that takes polynomial time.
(2) Verify that if there exists a solution satisfying this instance of B, then there exists a solution satisfying the corresponding instance of A.
(3) Verify that if there does not exist a solution satisfying the instance of B, then there does not exist a solution satisfying the corresponding instance of A. Equivalently, verify that if there exists a solution satisfying the instance of A, then there exists a solution satisfying the corresponding instance of B.

Thus, given a polynomial time algorithm for B, we obtain a polynomial algorithm for A by applying the algorithm for B to the transformation of an instance of A to one of B.

Theorem 3.1 (Cook-Levin Theorem)**.** *SAT is NP-complete* [3].

SAT is our prototypical problem, and we will provide a brief explanation rather than a formal proof of its NP-completeness.

The underlying property of all problems in NP is that they have polynomial verification algorithms. Such algorithms can be expressed as polynomial size Boolean circuits, which form the basis of computers. The problem of finding satisfying assignments to Boolean circuits is called Circuit SAT, and it is a generalization of SAT (and an alternatively used prototypical problem). Thus, the solutions to any problem in NP are in bijective correspondence with the solution assignments to Circuit SAT, so any problem in NP reduces to Circuit SAT and in turn reduces to SAT.

Theorem 3.2. *SAT reduces to 3SAT.*

Proof. Consider any instance θ of SAT. Let

$$(a_1 \vee a_2 \vee \cdots \vee a_k)$$

be an arbitrary clause α of k literals in θ, where each a_i is either a variable or the negation of one.

Introduce new variables $x_1, \dots, x_{k-3}$. Replace α with

$$(a_1 \vee a_2 \vee x_1)(\neg x_1 \vee a_3 \vee x_2)(\neg x_2 \vee a_4 \vee x_3) \dots (\neg x_{k-3} \vee a_{k-1} \vee a_k),$$

which we denote as β. Repeat with new variables for each clause in θ. Let the resultant instance of 3SAT be ϕ.

The replacement procedure is $O(mn)$, where n is the number of variables and m is the number of clauses, so it is polynomial.

Claim: There exists an assignment satisfying θ if and only if there exists an assignment satisfying ϕ.

Proof: Let there exist an assignment satisfying θ and thus all α. Thus, at least one a_l is true. Then, set x_i true for $i \leq l-2$ and x_i false for all $i \geq l-1$. This satisfies all β and thus ϕ.

Let there exist an assignment satisfying ϕ and thus all β. Consider the first clause in β. Since β is satisfied, a_1 or a_2 is true and α is satisfied or x_1 is true. Now consider the second clause. If x_1 is true, then a_3 is true or x_2 is true. Continue to the final clause. If all x_i are true, including x_{k-3}, then a_{k-1} or a_k is true. Thus, the same assignment satisfying all β satisfies all α and thus θ. □ □

This reduction is an example of one from a more general problem to a special case since 3SAT is a special case of SAT.

However, such reductions do not necessarily exist. For example, if we restrict SAT to clauses of at most one literal, calling it 1SAT, then 1SAT is clearly P since an algorithm can simply assign variables values for each clause until a solution is reached or becomes impossible. By Theorem 3.1, if SAT reduces to 1SAT, then P = NP, which is unknown.

Theorem 3.3. *3SAT reduces to independent set.*

Proof. Consider any instance θ of 3SAT.

We construct a graph G such that

(1) For each clause in θ, we construct a clique with one vertex for each literal in the clause. A clique is a set of vertices such that every two vertices in the set are adjacent; in other words, it is a complete subgraph.
(2) For each pair of vertices in the graph, if the corresponding literals are negations of each other, we draw an edge between them.

Set budget m, where m is the number of clauses in θ. Let the resultant instance of independent set be ϕ.

The replacement procedure is $O(m^2)$ since each clause has at most 3 literals, making constructing the edges between negations take at most $\binom{3m}{2}$ steps, so it is polynomial.

Claim: There exists an assignment satisfying θ if and only if there exists an independent set G satisfying ϕ.

Proof: Let there exist an assignment satisfying θ. Thus, every clause has at least one true literal. Select the vertices corresponding to one true literal for each clause. The set is independent since vertices are adjacent if and only if their literals are in the same clause, which we chose them not to be, or are negations of each other, which is not a valid assignment.

Let there exist an independent set of m vertices of G. Since G has m cliques, there is exactly one vertex from each clique in the set, corresponding to literals satisfying each clause. Since vertices corresponding to negations of each other are adjacent, the assignment of variables corresponding to the vertices is valid, satisfying θ. □ □

Theorem 3.4. *Independent set reduces to vertex cover.*

Proof. Consider any instance θ of independent set. θ consists of graph G and budget b. Let n be the number of vertices of G.

Let ϕ be the instance of vertex cover consisting of graph G and budget $n - b$.

The replacement procedure is constant and thus polynomial time.

Claim: There exists an independent set satisfying θ if and only if there exists a vertex cover satisfying ϕ.

Proof: Let there exist an independent set of b vertices of G. There are no edges between the b vertices of the independent set, so the complement consisting of the other $n - b$ vertices is incident to all the edges of G.

Let there exist a set of $n - b$ vertices covering all edges of G. All edges of G are incident to some vertex in this set, so the complement consisting of the other n vertices is an independent set. □ □

Lemma 3.5. *3SAT reduces to a special case of 3SAT where each variable appears no more than three times (and no literal appears more than twice).*

Proof. Consider any instance θ of 3SAT. Let a be an arbitrary variable in θ, appearing in k different clauses.

Introduce new variables $x_1, \ldots, x_k$. Replace the ith appearance of a with x_i so that no x_i appears more than once. Denote these clauses α. Introduce new clauses

$$(\neg x_1 \vee x_2)(\neg x_2 \vee x_3) \ldots (\neg x_k \vee x_1).$$

Denote these clauses β. Repeat this construction for each variable in the instance of θ. Let the resultant instance of restricted 3SAT be ϕ.

The replacement procedure is at most $O(mn)$, where n is the number of variables and m is the number of clauses, so it is polynomial.

Claim: There exists an assignment satisfying θ if and only if there exists an assignment satisfying ϕ.

Proof: Let there exist an assignment satisfying θ. For each variable a, assign the corresponding x_i the same truth value as a. Since the x_i replace a in ϕ, this satisfies all α. Since the x_i have the same truth value as a and thus as each other, this satisfies β, satisfying ϕ.

Let there exist an assignment satisfying ϕ. For any two variables y and z, the clause $(\neg y \vee z)$ is equivalent to $y \implies z$. Thus, since β is satisfied, all x_i for $1 \leq i \leq k$ have the same truth value. These variables satisfy α, so assign the truth value of the corresponding a to be the same as the x_i, satisfying θ. □ □

Theorem 3.6. *3SAT reduces to 3D Matching.*

Proof. By Lemma 3.5, it suffices to show that the special case of 3SAT reduces to 3D Matching.

Consider any instance θ of the special case of 3SAT described in Lemma 3.5. Let n be the number of variables in θ and m be the number of clauses in θ.

Construct X, Y, Z, and T as thus:

(1) For each variable a_i where $1 \leq i \leq n$, introduce $x_{vi1}, x_{vi2} \in X$, $y_{vi1}, y_{vi2} \in Y$, and $z_{vi1}, z_{vi2}, z_{vi3}, z_{vi4} \in Z$ such that:
$(x_{vi1}, y_{vi1}, z_{vi1}), (x_{vi2}, y_{vi1}, z_{vi2}), (x_{vi2}, y_{vi2}, z_{vi3}), (x_{vi1}, y_{vi2}, z_{vi4}) \in T$. We will hence refer to these as variable triples.

(2) For each clause b_{ci} where $1 \leq i \leq m$ introduce $x_{ci} \in X$ and $y_{ci} \in Y$ such that:
For every variable a_j that is part of b_{ci}, let $(x_{ci}, y_{ci}, z_{vj2}), (x_{ci}, y_{ci}, z_{vj4}) \in T$ if b_{ci} requires a_j to be true and $(x_{ci}, y_{ci}, z_{vj1}), (x_{ci}, y_{ci}, z_{vj3}) \in T$ if b_{ci} requires a_j to be false. We will hence refer to these as clausal triples.
(3) Introduce $x_{ui} \in X$ and $y_{ui} \in Y$ for all $1 \leq i \leq 2n - m$ such that:
All triples containing both x_{ui} and y_{ui} for $1 \leq i \leq 2n - m$ are in T. We will hence refer to these as universal triples. No literal appears more than twice, so $2n - m \geq 0$.

Let the resultant instance of 3D Matching be ϕ. Note that $|X| = |Y| = |Z| = 4n$.

The replacement procedure is $O(n)$, so it is polynomial.

Claim: There exists an assignment satisfying θ if and only if there exists a perfect matching satisfying ϕ.

Proof: Let there exist an assignment satisfying θ. We can now select triples from T to attain a perfect matching. We have to make sure we have $4n$ triples and no triples share a variable.

(1) For every variable a_i where $1 \leq i \leq n$, select the variable triples $(x_{vi1}, y_{vi1}, z_{vi1}), (x_{vi2}, y_{vi2}, z_{vi3})$ if a_i is true and select the variable triples $(x_{vi2}, y_{vi1}, z_{vi2}), (x_{vi1}, y_{vi2}, z_{vi4})$ if a_i is false. There is no overlap, and we have $2n$ triples.
(2) Since all clauses are satisfied, each clause has at least one literal satisfying it. Consider exactly one such literal for each clause. Let the list of these literals be λ where $|\lambda| = m$. For each appearance of every literal in λ, corresponding to a variable a_j, select a clausal triple containing either z_{vj2} or z_{vj4} if the literal is true and select a clausal triple containing either z_{vj1} or z_{vj3} if the literal is false. No literal appears more than twice, so we can do this without overlap, and we have a total of $2n + m$ triples.
(3) Finally, for the remaining triples, select $2n - m$ universal triples, one for each $1 \leq i \leq 2n - m$, so they have no overlap. We now have $4n$ triples.

This matching is thus a perfect matching satisfying ϕ.

Let there exist a perfect matching satisfying ϕ.

(1) Since $x_{vi1}, x_{vi2}, y_v, y_{vi2}$ are part of no other triples, either $(x_{vi1}, y_{vi1}, z_{vi1}), (x_{vi2}, y_{vi2}, z_{vi3})$ or $(x_{vi2}, y_{vi1}, z_{vi2}), (x_{vi1}, y_{vi2}, z_{vi4})$ are triples in the matching. Assign the variables a_i for $1 \leq i \leq n$ as thus:
   (a) If $(x_{vi1}, y_{vi1}, z_{vi1}), (x_{vi2}, y_{vi2}, z_{vi3})$ are part of the matching, then let a_i be true.
   (b) If $(x_{vi2}, y_{vi1}, z_{vi2}), (x_{vi1}, y_{vi2}, z_{vi4})$ are part of the matching, then let a_i be false.

   No other variable triple can be in the matching due to overlap. We have considered $2n$ of the triples in the matching.
(2) There are a maximum of $2n - m$ non-overlapping universal triples, so at least m triples must be clausal. Since the clausal triples do not overlap, they must correspond to literals in different clauses, so we must have one for each clause for exactly m triples. For all $1 \leq i \leq n$:
   (a) If $(x_{vi1}, y_{vi1}, z_{vi1}), (x_{vi2}, y_{vi2}, z_{vi3})$ are part of the matching, then the only clausal triples that do not overlap are those containing z_{vi2} or z_{vi4}, which correspond to the variable a_i being true in the corresponding clause.
   (b) If $(x_{vi1}, y_{vi1}, z_{vi2}), (x_{vi2}, y_{vi2}, z_{vi4})$ are part of the matching, then the only clausal triples that do not overlap are those containing z_{vi1} or z_{vi3}, which correspond to the variable a_i being false in the corresponding clause.

In both cases, our assignment satisfies the corresponding clause, so all clauses are satisfied. We have considered $2n + m$ triples in the matching.

(3) Universal triples make up the remaining $2n-m$ triples in the perfect matching to bring our total to $4n$ but have no further bearing on the assignment of variables.

Our assignment thus satisfies θ. □ □

Theorem 3.7. *3D Matching reduces to ZOE.*

Proof. Consider any instance θ of 3D Matching. θ consists of the sets X, Y, and Z of size n and the set of valid triples T of size k.

For each triple in T, introduce a variable x_i where $1 \leq i \leq k$. For any $a \in X$ or Y or Z, let $\sum_{j\in\lambda_a} x_j = 1$, where λ_a is the set of all triples containing a. Let the resultant instance of ZOE be ϕ.

The replacements procedure is $O(k)$, with k bounded by n^3, so it is polynomial.

Claim: There exist n disjoint triples satisfying θ if and only if there exists a variable assignment of 0's and 1's satisfying ϕ.

Proof: Let there exist n disjoint triples satisfying θ. Assign every variable x_i corresponding to these disjoint triples to 1 and assign every other variable to 0. Since only disjoint triples are 1, $\sum_{j\in\lambda_a} x_j = 1$ for any $a \in X$ or Y or Z, so this assignment satisfies ϕ.

Let there exist a variable assignment of 0's and 1's satisfying ϕ. Since $\sum_{j\in\lambda_a} x_j = 1$ for any $a \in X$ or Y or Z, all variables with values of 1 correspond to disjoint triples, and there are no more than n of such variables. Since there exists a λ_a for each $a \in X$ or Y or Z, there are at least n and thus exactly n variables with values of 1 corresponding to a perfect matching. □ □

Lemma 3.8. *ZOE reduces to a generalization of Hamiltonian cycle where, given a set $C \subseteq E \times E$ of edge pairs of the graph, exactly one edge of each pair in C is part of the cycle. This is the paired edge condition. This generalization is called Hamiltonian cycle with paired edges. We allow multigraphs.*

Proof. Consider any instance θ of ZOE. θ consists of m equations in n variables, and every equation is a summation of the variables equaling 1.

Construct a graph $G = (V, E)$ where V is the set of vertices with $|V| = m + n$ and E is the set of edges such that G is a cycle of a series of parallel edges, with no other edges. That is, any vertex $v_i \in V$ where $1 \leq i \leq m + n$ is adjacent only to vertices v_{i+1} and v_{i-1}, taking $v_0 = v_{m+n}$ and $v_1 = v_{m+n+1}$, and the adjacency occurs through multiple, parallel edges. Construct the parallel edges as follows:

(1) For every variable a_j where $1 \leq j \leq n$, a distinct pair of adjacent vertices has 2 parallel edges, where one edge corresponds to $a_j = 1$ and the other corresponds to $a_j = 0$.
(2) For every equation, a distinct pair of adjacent vertices has k parallel edges, where k is the number of variables in the equation, and each edge corresponds to a different variable in the equation.

Construct the set $C \subseteq E \times E$ by adding for every appearance of a variable a in an equation the pair (e, e'), where e is the edge corresponding to the appearance of that variable in the equation and e' is the edge corresponding to $a = 0$.

The replacement procedure is $O(m + n)$, so it is polynomial.

Claim: There exists a variable assignment of 0's and 1's satisfying θ if and only if there exists a Hamiltonian cycle satisfying ϕ.

Proof: Let there exist a variable assignment of 0's and 1's satisfying θ. Consider the Hamiltonian cycle in G passing through:

(1) The edge corresponding to $a = 0$ or the edge corresponding to $a = 1$ for every variable a, depending on the value of a in the assignment. This takes care of parallel edges coming from each variable.
(2) The edges corresponding to the appearance of a variable in an equation for every variable with a value of 1. This takes care of parallel edges coming from each equation.

This cycle exists since every variable has a value of 0 or 1 and every equation is satisfied by having exactly one variable within it equal 1. The cycle meets the paired edges condition since one edge in the pair corresponds to a variable equaling 0 and the other corresponds to the same variable equaling 1 to satisfy an equation, and the equations are satisfied by having exactly one or the other for each variable in each equation.

Let there exist a Hamiltonian cycle meeting the paired edges condition.

(1) For the parallel edges coming from each variable, assign the variable the value corresponding to the edge in the cycle.
(2) For the parallel edges coming from each equation, variables corresponding to the edge in the cycle must have a value of 1 and variables corresponding to the edge not in the cycle must have a value of 0 to meet the paired edges condition.

Additionally, this implies that no variable has a value of both 1 and 0. Every equation is satisfied since the edge traveling through the parallel edges coming from each equation has exactly one variable equaling 1. □

Theorem 3.9. *ZOE reduces to Hamiltonian cycle.*

Proof. By Lemma 3.8, it suffices to show that Hamiltonian cycle with paired edges allowing multigraphs reduces to Hamiltonian cycle. While our stated version of Hamiltonian cycle disallows multigraphs, these versions reduce to each other since one may eliminate excess parallel edges.

Consider any instance θ of Hamiltonian cycle with paired edges. θ consists of the graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, and the set of paired edges C. Let k be the number of paired edges in C.

For every $(e_j, e'_j) = (\{v_{j1}, v_{j3}\}, \{v_{j2}, v_{j4}\}) \in C$, where $1 \le j \le k$, construct from G as thus:

(1) If e_j and e'_j have not yet been modified, eliminate e_j and e'_j. Then introduce vertices $w_1, w_2, \ldots, w_{12}$ such that the following paths exist and encompass the only edges involving all w_i for $1 \le i \le 12$:
$\alpha = \{v_{j1}, w_1, w_2, w_3, w_6, w_5, w_4, w_7, w_8, w_9, w_{12}, w_{11}, w_{10}, v_{j3}\}$ and
$\beta = \{v_{j3}, w_3, w_2, w_1, w_4, w_5, w_6, w_9, w_8, w_7, w_{10}, w_{11}, w_{12}, v_{j4}\}$.

$$\begin{tikzcd}
v_{j1} \arrow[r, dash] & w_1 \arrow[r, dash] \arrow[d, dash] & w_4 \arrow[r, dash] \arrow[d, dash] & w_7 \arrow[r, dash] \arrow[d, dash] & w_{10} \arrow[r, dash] \arrow[d, dash] & v_{j3} \\
 & w_2 \arrow[r, dash] \arrow[d, dash] & w_5 \arrow[r, dash] \arrow[d, dash] & w_8 \arrow[r, dash] \arrow[d, dash] & w_{11} \arrow[d, dash] & \\
v_{j2} \arrow[r, dash] & w_3 \arrow[r, dash] & w_6 \arrow[r, dash] & w_9 \arrow[r, dash] & w_{12} \arrow[r, dash] & v_{j4}
\end{tikzcd}$$

Essentially replace e_j and e'_j with this construction. The construction is known as a gadget since it is a part of a problem instance designed to mimic the behavior of the conditions of another problem. Specifically, it is a part of the normal Hamiltonian cycle problem designed to mimic the condition of paired edges. Note that any Hamiltonian cycle involving the gadget must contain exactly one of two possible paths through the gadget, α or β, just as exactly one edge must be chosen from an edge pair. Keep the definitions of α and β in mind.

(2) If e_j has already been modified, make no additional constructions.
(3) If e'_j has already been modified, make no additional constructions.

Let the graph be H, and let the resultant instance of Hamiltonian cycle be ϕ.

The replacement procedure is $O(k)$, so it is polynomial.

Claim: There exists a Hamiltonian cycle satisfying θ if and only if there exists a Hamiltonian cycle satisfying ϕ.

Proof: Let there exist a Hamiltonian cycle in G meeting the matched pairs condition. Construct a Hamiltonian cycle for H as thus:

(1) For edges in the cycle that are part of both G and H, retain the edge.
(2) For edges e_j in the cycle that are only part of G but not H, e_j must have been eliminated during the process of constructing H. Thus, there exists an edge pair $(e_j, e'_j) \in C$, where $e_j = \{v_{j1}, v_{j3}\}$, from which one of the gadgets was constructed. Choose α from the gadget.

This construction is a cycle since both cases retain the start and end vertices of the sequence of paths that union to the Hamiltonian cycle in G. This cycle is Hamiltonian since all vertices originally part of G as well as the new ones introduced to H are incorporated.

Let there exist a Hamiltonian cycle in H. Construct a Hamiltonian cycle for G as thus:

(1) Edges in the cycle that are part of H but not part of G must be in exactly one of the gadgets constructed. Specifically, supposing the particular gadget was constructed from the edge pair $(e_j, e'_j) \in C$, then either α or β in the gadget are in the cycle (the edges are in one of these paths). Thus:
    (a) If α is part of the cycle, choose e_j.
    (b) If β is part of the cycle, choose e'_j.
(2) Retain all other edges in the cycle, which are part of both G and H.

This construction is a cycle for the same reasons as the other direction in that start and end vertices are preserved. This cycle is Hamiltonian since all vertices remain incorporated other than the vertices present in H but not in G. We meet the paired edges condition since, for edge pairs that led to the construction of a gadget, exactly one of α and β must be part of the cycle in H and thus exactly one of the corresponding edges in the edge pair $(e_j, e'_j) \in C$ (α with e_j and β with e'_j) must be part of the cycle in G. Other edge pairs must share an edge with one of these initial pairs and are satisfied the same way. If e_j is the shared edge and α is the cycle in H, then e_j is in the cycle in G; if β is in the cycle in H, then the other edge in the pair is in both the cycle in G and the cycle in H). □

Theorem 3.10. *Hamiltonian cycle reduces to TSP.*

Proof. Consider any instance θ of Hamiltonian cycle. θ consists of graph G. Let n be the number of vertices of G.

Construct a weighted graph H by assigning all edges of G a weight of 1 and then adding edges of weight 2 to create a complete graph.

The replacement procedure is $O(m)$, where m is the number of edges in H, so it is polynomial.

Claim: There exists a Hamiltonian cycle satisfying θ if and only if there exists a Hamiltonian cycle satisfying ϕ.

Proof: Let there be a Hamiltonian cycle in G. Taking the corresponding Hamiltonian cycle in H consisting of edges of weight 1, the sum of edge weights equals n, so the cycle is within budget.

Let there be a Hamiltonian cycle in H with edge weights that sum to less than or equal to n. Since H is a complete graph with n vertices and minimum edge weight 1, the Hamiltonian cycle must only have edges of weight 1. Edges of weight 1 have corresponding edges in G, so there exists a Hamiltonian cycle in G. $\square$

Theorem 3.11. *If a problem A is a special case of a problem B, then A reduces to B.*

Proof. Consider an algorithm that solves B in polynomial time. Since A is a special case of B, the same algorithm also solves A in polynomial time. $\square$

Corollary 3.12. *Vertex cover reduces to set cover.*

Corollary 3.13. *ZOE reduces to ILP.*

4. Related Problems: Search and Optimization

We defined the notion of NP in Section 2 for decision problems. However, in our examples, this definition restricted the problems to those concerning the existence of some object meeting the problem's requirements. Thus, we turn to search problems, whose solutions are the actual objects of interest.

Recall that Euler Path in Example 2.2 is initially given as a search problem. As another example:

Example 4.1. Consider the Hamiltonian cycle problem. Given a graph G,

(1) Decision... determine the existence of a Hamiltonian cycle in G.
(2) Search... find a Hamiltonian cycle in G.

It is clear that decision reduces to search since the existence of a solution fulfilling search implies yes to decision, and nonexistence implies no. It turns out that search reduces to decision in at least some cases.

Theorem 4.2. *The search version of Hamiltonian cycle reduces to the decision version of Hamiltonian cycle.*

Proof. Consider any instance θ of Hamiltonian cycle. θ consists of graph $G = (V, E)$, where V is the set of vertices and E is the set of edges with $|E| = m$.

Let there exist an algorithm that solves the decision version of Hamiltonian cycle in polynomial time. Consider any edge $e \in E$. Call the decision algorithm on the graph $G' = (V, E - \{e\})$.

(1) If the output is that there does not exist a Hamiltonian cycle in G', keep e.
(2) If the output is that there exists a Hamiltonian cycle in G', remove e.

Repeat the process for each resultant graph after an edge is considered until every edge in the original G has been considered.

Thus, an edge is removed only if there exists a Hamiltonian cycle not including that edge, so the final output is either the desired Hamiltonian cycle if it exists or the original graph G if it does not exist.

We call the decision algorithm $O(m)$ times, so the overall complexity is polynomial. $\square$

While the proof in this case exploited properties of graphs pertaining to this specific problem, search actually reduces to decision for all NP-complete problems [1]. This is because reductions for NP-completeness can be created for both decision and search versions of problems; indeed, all of the reductions in Section 3 apply to both versions. Thus, using the NP-complete Hamiltonian Cycle example problem we have already established, we can reduce other NP-complete problems from search to decision.

A corollary is that if there exist a class of problems in NP such that search does not reduce to decision, then P $\neq$ NP since such problems would not be NP-complete and P = NP implies P = NP = NP-complete. Consequently, such problems have not been shown to exist except under certain complexity assumptions [2].

Another type of problem of interest is optimization. For problems involving a budget, we may be more interested in the minimum or maximum assignment, not just any assignment within a given budget.

Like search and decision, reduction in one direction is trivial. A polynomial algorithm solving the optimization problem solves the corresponding search problem. The reverse is not true for as broad a class as all NP-complete problems, though it is true in some instances.

Definition 4.3 (Traveling Salesman Optimization or TSP-OPT)**.** Given a weighted complete graph, find a Hamiltonian cycle with the minimum edge sum.

We consider the special case where all the edge weights are non-negative integers.

Theorem 4.4. *TSP-OPT reduces to the search version of TSP.*

Proof. Consider an arbitrary weighted graph G. Let the number of vertices in G be n.

Consider the sum of all edge weights s. The edge sum of any cycle is bounded between 0 and s.

Perform binary search on the set $\{0, 1, \ldots, s\}$. Thus, we call the search algorithm $O(\log s)$ times.

Consider the largest edge weight w. s is bounded by $\binom{n}{2}w$, so the overall complexity is polynomial. $\square$

Clearly, we cannot create such a reduction for all NP-complete problems since many lack meaningful optimization versions, such as Hamiltonian cycle.

Note that both search problems and optimization problems are NP-hard rather than NP-complete since they are not decision problems and thus not in NP. Additionally, while a proposed solution to a search problem is easily verified against an instance, the same cannot easily be done for optimization problems.

Acknowledgements. I would like to thank my mentor Shiva Chidambaram for helping me choose this topic and advising me throughout the process of writing the paper. I would also like to thank Peter May for directing the REU program.

References

[1] Mihir Bellare. Decision versus Search lecture notes. January 3 2010.

[2] Mihir Bellare and Shafi Goldwasser. The Complexity of Decision versus Search. *SIAM J. Comput* 23(1), 97–119, February 1994.

[3] Stephen Cook. The complexity of theorem-proving procedures. *Proceedings of the third annual symposium on Theory of computing*, 151-158, ACM, 1971.

[4] Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani. *Algorithms*, McGraw-Hill, 2008.

[5] Richard Karp. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, 85-103, Springer, 1972.