Karp Demystified (for publication)

Josie O'Harrow, Pranshul Lakhanpal, Jonathan Schreiber, Theresa Migler

August 2020

1 Introduction

The class of known NP Complete problems has expanded greatly since its introduction in the early 1970's. This has given us a structure on the NP problems, providing formal verification that the NP Complete problem are related either by the challenge they present to solve or by their intractability. The first such case is P = NP, and the second represents P ⊂ NP, a result that follows from Cook's "The Complexity of Theorem-Proving Procedures" [3].

From "The Complexity of Theorem-Proving Procedures", we gain an extraordinary result concerning Satisfiability: any problem which may be decided by a non-deterministic turning machine in polynomial time has a polynomial reduction via a deterministic Turing machine to the Satisfiability problem [3]. Cook shows that for any arbitrary NP problem M, we may bound the reduction from M to SAT in a way that is related the complexity of solving M on a nondeterministic machine. By construction of the class NP M must have polynomial complexity.

Generally speaking, reductions represent a way to quickly convert between an instance of one problem into another in a way that preserves acceptance or rejection of input. We define quickly as being polynomial. While an efficient programmer would be weary of adding unnecessarily nested code, we generalize quickness to mean there is a fixed number of nested algorithms- that we have a run time that is polynomial in our input. Reductions give us a way to formalize our intuition that certain problems are at least as hard as other problems, and provide some sense of ordering among decision problems from their transitivity. We may use that idea to define complexity classes, although we acknowledge that other constructions for complexity, such as restricting to linear in input reductions, may be valuable. [4].

Using reductions, Karp constructed the NP-Complete class as a set of problems known to be reducible from Satisfiability. This has extraordinary consequences: from Cook, any problem in NP is polynomial reducible to SAT [3]. In "Reducibility Among Combinatorial Problems", Karp presents 20 additional problems which, through the transitivity of polynomial reductions, may fill the role of SAT in the Cook thesis.

The expansion of the NP-Complete class has provided new canvas for researchers working on $P = NP$, formalized difficulty of algorithms for software engineers, and increased the trust in systems that assume $P \neq NP$.

In our work, we consider only the original 21 NP-Complete problems, and further reductions among these problems. We seek to independently show that all 21 problems have the same difficulty by constructing a cycle of reductions. Finally, we present disambiguation for the original reductions presented by Karp.

2 Related Work

2.1 Reducibility among Combinatorial Problems by Richard M. Karp

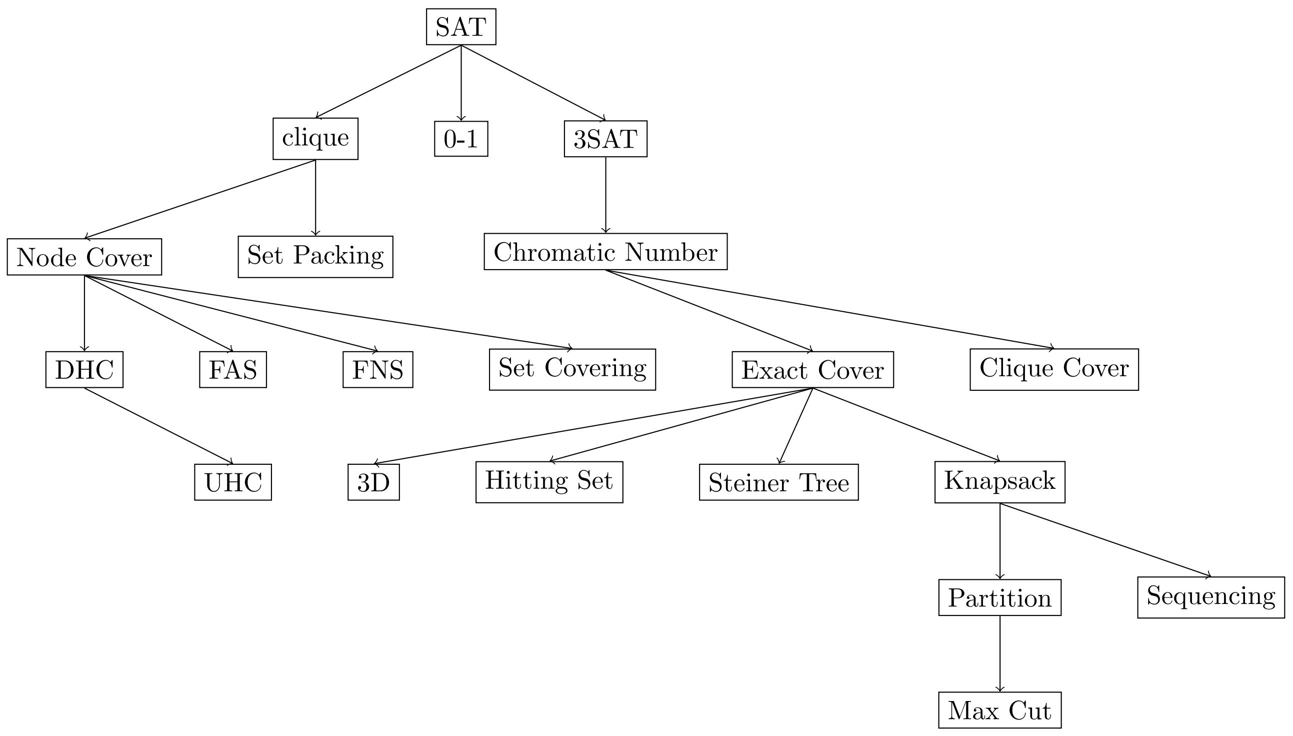


Figure 1: Karp's Tree

Our Summer research project was based on the paper ”Reducibility among Combinatorial Problems” published by Richard Karp. In his paper he shows a group of 21 problems to be NP-complete, by creating a tree of reductions which starts from Satisfiability. We outline the major sections of Karp's paper below.

Karp introduces a general model of computation, deterministic algorithms, but quickly refines his model to be terminating recognition algorithms. That is, algorithms which have finite computations and result in either “ACCEPT” or “REJECT”. Karp introduces string recognition algorithms as recognition algorithms with the domain $\{0, 1\}^*$. Classes of string recognition algorithms are a significant backbone for complexity theory, and introduce the idea of “equivalent” models of computation. Karp defines the class P to be those languages recognizable by single tape Turing Machines in polynomial time, and notes that this class is equivalent to the class recognized by multihead or multitape Turing Machines in polynomial time. The famous P vs. NP problem asks whether P is additionally equivalent to the class NP, problems which are recognizable in polynomial time on a nondeterministic Turing Machine. In a later section, Karp provides some surprising members of the class P: 2SAT, min. Spanning tree, shortest path, min. Cut, arc cover, arc deletion, 2d matching, sequencing with deadlines, solvability of linear equations. The simplicity of these problems and not their NP Complete relatives is shocking!

To define reducibility, Karp uses the notion of polynomial functions from Σ^* to Σ^*. The class of such functions is Π, and a language L is defined to be reducible to M exactly when there is an element in Π which transfers input in L to input in M and preserves acceptance and rejection. Of course, Karp assumes that languages are defined exactly over Σ^* here, but that requirement is later relaxed.

The class NP is defined in two distinct ways: The first uses I have absolutely no idea what existential quantification means*existential quantification*. The second is: NP problems are those recognizable by nondeterministic Turing Machines in polynomial time. Finally Karp introduces Cook's result on the class of NP problems, although notably the version of satisfiability used in Karp's work is CNF tautologies, equivalent to DNF tautologies as Cook pointed out via quick work with De Morgan's Laws.

The complete class is defined to be the set of languages which are both in NP and at least as hard as satisfiability. Because of Cook's result, this is equivalent to the class of languages in NP that are at least as hard as any other problem in NP. Although originally stated in terms of functions on $\Sigma^* \to \Sigma^*$, Karp extends the idea of reducibility to encompass functions between arbitrary countable domains. In order to retain the polynomial nature of the reduction, Karp requires that the function $F : D \to D'$: i) Preserves string acceptance ii) Is equivalent to a polynomial function when F composed with the encoding functions for the domain and the inverse encoding function for the codomain is defined. In the reductions provided, Karp does not give encoding functions or functions in Π. Instead, Karp uses his equivalent characterization to provide reductions between domains and codomains for each problem, and allows the reader to check axioms (i) and (ii). In our work, we have further verified axiom (i) for the reader. Karp provides a tree of reductions (figure –) to show that each problem in the tree is reducible from SAT, and consequently may play the role of SAT in Cook's theorem. We have that the problems listed are equivalent, as defined via reducibility, since each problem also reduces to SAT from Cook.

After listing his 20 NP Complete problems, Karp proceeds to list and prove that a certain class of problems are at least as hard as every NP Complete problem. Now these problems are considered "NP Hard". Karp introduces "Equivalence of Regular Expressions" (1), "Equivalence of Nondeterministic Finite Automata" (2), and "Context-Sensitive Recognition" (3). He then shows that 3SAT reduces to (1) which reduces to (2), and in addition that any NP problem will reduce to (3).

Karp concludes his paper by listing three problems suspected to be NP Complete. These problems were Graph Isomorphism, Nonprimes, and Linear Inequalities. We now know that Nonprimes is not NP Complete [8], although thankfully we still struggle to find the factors of such Nonprimes efficiently.

2.2 Adjacent Works

2.2.1 Cook reducibility is faster than Karp reducibility in NP by Luc Longpre and Paul Young

[7] Longpre and Young's paper compared Cook reducibility to Karp reducibility, to determine which is more efficient in proving problems are NP. This research paper investigates specific subsets of NP to show that some classes of NP-Complete set have Cook reductions which are must faster than Karp reductions. In relation to our research paper, this paper served as an additional resource to help understand the main differences between Cook and Karp reductions. Cook reductions rely on Turing Machines operating in polynomial time to reduce a problem to another one, while Karp reductions utilize a polynomial time function that converts an instance of one problem to an instance of another problem.

2.2.2 On the Structure of Combinatorial Problems and Structure Preserving Reductions by Giorgio Ausiello, Allessandro D'Atri, and Marco Protasi

[1] Ausiello, D'Atri, and Protasi studied the structure of optimization problems and created classes of problems based on these structures. Through this, they were able to create a partial ordering of problems and provide guidelines for creating reductions which preserve the structure of combinatorial problems. They also provided a list of reductions which all reduce easily and elegantly because the structure of the problem is preserved. The reductions in this list include Clique $\propto$ Set Packing, Clique $\propto$ Node-Cover, Node-Cover $\propto$ Set-Covering I, Node-Cover $\propto$ Feedback Node Set, Set Covering I $\propto$ Hitting Set, and Chromatic Number $\propto$ Exact Clique-Cover. This was especially useful in our research when coming up with original reductions, and understanding why some problems reduced to each other easily while others are extremely challenging and not intuitive.

2.2.3 Different adiabatic quantum optimization algorithms for the NP-complete exact cover problem by Vicky Choi

[2] Choi investigated an optimization algorithm for the Exact Cover problem. She found that although it was claimed by Altshuler that adiabatic quantum optimization fails for random instances of exact cover, this claim is incorrect because flexible parameters were overlooked. Thus, Choi concludes that the complexity of adiabatic quantum algorithms for random instances of exact cover is open. While there was nothing from this research that directly contributed to my research, it was interesting and important to learn more about Exact Cover, as several of the original reductions I came up with involved the Exact Cover Problem.

2.2.4 Improved Results on Geometric Hitting Set Problems by Nabil H. Mustafa and Saurabh Ray

[10] Mustafa and Ray worked with the Geometric Hitting Set Problem to find an optimization algorithm. The optimization algorithm they came up with was a simple local-search algorithm that iterated over local improvements. Similarly to the last related work on Exact Cover, this paper helped improve understanding of the Hitting Set problem, as many of the original reductions I came up with involved Hitting Set.

2.2.5 Linearly-Growing Reductions of Karp's 21 NP-Complete Problems

In this text, the authors present a kernel of Karp's original 21 problems which every other problem is reducible to in not only polynomial, but linear time on the inputs. The kernel contains 0-1 Integer Programming, which shows to be versatile for linear reductions on Karp's NP Complete problems. Also included are Feedback Node Set, Undirected Hamiltonian Cycle, Chromatic Number, Clique Cover, and Job Sequencing. The authors suggest that linear reductions provide a useful way to define complexity classes. The authors also introduce slack and surplus variables as a way to handle the inequality present in $0-1$, an idea that was inspirational for our reduction from Satisfiability to Knapsack as we considered how to allow for clauses to be satisfied by more than one literal that they contain.

2.2.6 The Complexity of Optimization Problems by Mark W. Krentel

[6] Krentel deeply investigated several NP-Complete Problems to compute their optimal value. He was able to create clases within the class of NP-Complete Problems through this investigation, and quantified different levels of NP-completeness within the set of NP-complete problems. He found that Traveling Salesperson is strictly harder than Clique, and Clique is strictly harder than Bin Packing. This related to our research as it helped to understand that different problems are of different difficulties, which helped to understand why some reductions were trivial and others were extremely complex. It was also extremely interesting to understand how there are different classes within the class of NP-Complete Problems, based on the structure and optimal value of the problems.

2.2.7 The NP-Completeness of Steiner Tree and Dominating Set for Chordal BiPartite Graphs by Haiko Muller and Andreas Brandstadt

[9] Muller and Brandstadt showed that Steiner Tree, Dominating Set, and Connected Dominating Set are NP-Complete for chordal bipartite graphs. The NP-Completeness of Steiner Tree was particularly useful to our research, as it was one of the problems we were working with in attempting to construct a simple cycle. In this paper, it was shown that Steiner Tree is NP-Complete by reducing Vertex Cover to Steiner Tree. This reduction was valuable to our research as it added a potential edge in our cycle, and showed how problems can be reduced to Steiner Tree.

2.2.8 Primes is in P

[8] The set PRIMES is the language consisting of all prime numbers. Manindra Agrawal, Neeraj Kayal, and Nitin Saxena present an algorithm for testing membership in PRIMES in polynomial time on a single tape, deterministic Turing Machine. They verify their result using number theoretic results, both original and from existing literature. This paper has major significance to the original Karp paper, because Karp listed PRIMES as a problem of unknown NP-Complete status; their paper shows that PRIMES is not NP Complete unless P = NP, as it is a member of P. This paper has other significance on the field of computing as a whole; traditionally fast primality tests used number theory heuristics such as Fermat's Little Theorem, which holds for all primes with any base, but also for some composite-base pairs. For primality testing, work has been done to improve the accuracy of the result through additional layers of testing or use of multiple methods, but until "PRIMES is in P" there was no known polynomial algorithm that guaranteed correctness. This

holds major consequences on areas of computing based on primality testing, including cryptography, and gives us a surprising result for a problem previously considered hard.

2.3 The Complexity of Theorem-Proving Procedures

[3] Cook's 1971 paper established the now famous result that for any problem L in NP, $L \leq_P$ SATISFIABILITY. We provide his result and a close rendition of the proof supplied below. This proof needs to be worked on. I am hoping to construct all of the clauses explicitly, like I did for the first four already. In addition, I think it makes sense to justify and modify the machine before the construction of literals and clauses, because at the very end it throws me off reading through it.

Theorem 2.1. *For any problem $L \in NP$, $L \leq_P$ Satisfiability in Disjunctive Normal Form (DNF-SAT).*

Proof. Proof. First note that Satisfiability in Conjunctive Normal Form (CNF-SAT) and DNF-SAT are equivalent under the relation of p-reducibility, since an instance of one may be converted to an instance of the other using De Morgan's Laws. We proceed to show that NP problems are reducible to CNF-SAT. Let L be a problem in NP. Since L is in NP, $\exists Q \in \pi$ so that a non-deterministic machine M decides L within Q(n) time. Let w be an input for M. We seek to construct a set of clauses from w which are satisfiable if and only if w is accepted by M.

Because Q(n) bounds our computation on a nondeterministic machine, we seek a computation of length $\leq Q(|w|)$ in both time and number of squares. We will employ this fact in our construction.

Let $\{\sigma_1, \sigma_2, \cdots, \sigma_l\}$ be the tape alphabet for M, and $\{q_1, q_2, \cdots, q_s\}$ be the states. Let $\{x_i\}$ be defined as follows:

$$\begin{aligned} \{x_i\} =& \{P^i_{s,t} \mid 1 \leq i \leq l \wedge 1 \leq s,t \leq Q(|w|)\} \\ & \cup \{Q^i_t \mid 1 \leq i \leq s \wedge 1 \leq t \leq Q(|w|)\} \\ & \cup \{S_{s,t} \mid 1 \leq s,t \leq Q(|w|)\} \end{aligned}$$

We construct the following clauses for CNF-SAT:

$$C_1 = \bigwedge_{1 \leq t \leq Q(|w|)} \left(\left(\bigvee_{1 \leq i \leq Q(|w|)} S_{i,t} \right) \wedge \left(\bigwedge_{1 \leq i < j \leq Q(|w|)} (= \neg S_{i,t} \vee \neg S_{j,t} \right) \right)$$

Here C_1 guarantees that for each time t, exactly one square is scanned. We are gauranteed existence from the first sub-clause in each D_t, and uniqueness from the second. We proceed:

$$C_2 = \bigwedge_{1 \leq s,t \leq Q(|w|)} \left(\left(\bigvee_{1 \leq i \leq l} P^i_{s,t} \right) \wedge \left(\bigwedge_{1 \leq i < j \leq l} \neg P^i_{s,t} \vee \neg P^j_{s,t} \right) \right)$$

C_2 gaurantees that for each square s and time t, there is exactly one symbol.

$$C_3 = \bigwedge_{1 \leq t \leq Q(|w|)} \left(\left(\bigvee_{1 \leq i \leq s} Q^i_t \right) \wedge \left(\bigwedge_{1 \leq i < j \leq s} \neg Q^i_t \vee \neg Q^j_t \right) \right)$$

C_3 gives us that for each time t, there is exactly one state.

$$C_4 = Q_1^{\cdot} \wedge S_{1,1} \wedge P_{1,1}^{i_1} \wedge \cdots \wedge |w|, 1^{i_{|w|}} \wedge P_{|w|,1}^{1} \wedge \cdots \wedge P_{Q(|w|),1}^{q}$$
$$\text{Given that w } = \sigma_{i_1} \cdots \sigma_{i_{|w|}} \text{ and } \sigma_1 \text{ is the blank symbol}$$

C_4 verifies that the computation contains the correct initial conditions.

C_5, C_6, and C_7 are constructed according to the transition function for the machine. C_5 is responsible for ensuring that at each time transition, the symbols update appropriately. C_6 ensures that the states update according to δ, and C_7 ensures that square (head position) is updated appropriately.

C_8 verifies that an accepting state is reached by the machine at some point. Cook's final specification is that we modify the machine to continue computing after the accept state so that $C_1 \wedge C_2 \wedge \cdots \wedge C_8$ is satisfied for an accepting computation. The reader is left to verify the complexity. □

In addition to his strong result, Cook discusses limitations on complexity theory and suggests models to formalize complexity classes for theorem proving procedures. Cook further defines degree of difficulty from the idea of polynomial reducibility. The consequences of Cook's paper extends far beyond his result on SAT reducibility; Cook inspired theoretical computer scientists for generations to come, including Karp who references Cook's paper as a source of inspiration, and introduced new formalisms to investigate complexity. Finally, Cook suggests that future work center around showing that tautologies is not in P, which of course has been given considerable attention and remains an open question.

2.4 Notation

$\mathbb{N}$	Natural numbers $\{1, 2, 3, ...\}$
$\mathbb{Z}$	Integers {..., -2, -1, 0, 1, 2, ...}
$\leq_P$	Reduces via a single-tape Turing Machine in polynomial time
G = (V, E)	A graph with vertices V and edges E
P	The class of problems decided in polynomial time on a single tape Turing Machine
NP	The class of problems decided in polynomial time on a single tape nondeterministic Turing Machine.

Should we add more notation? I didn't put a lot in the actual table The reductions we present in this paper are polynomial reductions between NP complete problems. We formalize the idea of a polynomial reduction through the notion of a single tape Turing machine.

From the construction of NP, we know any problem presented here may be decided by some nondeterministic Turing machine in polynomial time. That is, if we were to create a tree of all possible solutions to check for validity, the longest branch would be polynomial on the input. Because some Turing Machine decides each problem, we make the simplifying assumption that the input alphabet for any NP Complete problem presented here is $\mathbb{Z}_2$, and all inputs are a finite string over $\mathbb{Z}_2$. Note an alternative choice would be to justify the use of arbitrary domains as Karp did in [5].

Here I am not sure if this is what Sipser/ Cook do, but it is definitely different than Karp's convention. Karp uses arbitrary finite domains, and his lemma to justify this. I am instead trying to say something of the form, if it is fast on an arbitrary domain then there is a fast, equivalent machine that recognized that domain encoded into $\mathbb{Z}_2$. I think it is nuanced from what Karp did but I am not sure. I would like to keep this section because it was fun to write but I am not sure if this is how we want to do it.

Theorem 2.2. *NP problems are decided in polynomial time by a nondeterministic Turing machine over the alphabet $\mathbb{Z}_2$.*

Proof. Let A be an NP problem decided by nondeterministic machine M in time O(p(n)) with alphabet β. Note that β is finite, by the definition of Turing machines. We may uniquely index every element in β using $\mathbb{Z}_2$ with strings over $\mathbb{Z}_2$ of length $log_2(|\beta|)$. Assume that the indexing is done via f. If you'd like to include the complexity of computing f, note that it has at most $|\beta|$ items to compute, and thus will be linear in the length of the original alphabet. Now we construct G, a nondeterministic Turing machine that decides A over $\mathbb{Z}_2$:

On input I = $(w \in \mathbb{Z}_2^*)$
Run M on f(w)

Note that f(w) yields an output of length $|w| * log_2(|\beta|)$. This is polynomial in our original input, and since the composition of polynomial functions is polynomial we have verified the existence of G. □

From here on, we are justified making the assumption that each problem uses the library $\mathbb{Z}_2$, and we assert that Σ^* represents the set of all finite strings over $\mathbb{Z}_2$. The language of each NP problem is the set of strings over Σ^* that are accepted. For each problem, the accepting criteria is explicitly given so that only those encodings satisfying the accept clause will be accepted by the machine. Now we are prepared to define reductions as they will be used throughout this paper.

Definition 2.1. Reduction [5] Given two languages, M and G, a reduction from M to G is given by a function $f : \Sigma^* \to \Sigma^*$ which is computable in polynomial time by a one-tape turing machine and satisfies:
i) w $\in M \iff f(w) \in G$.

That is, there is a polynomial function that maps input strings for M to input strings for G and preserves language inclusion.

Following the conventions established by Karp, we will treat each decision problem as a language recognition problem. For each language, the INPUT is the possible input string, and SOLUTION describes arbitrarily all of the members of the language. We feel justified in using arbitrary alphabets for our problems because we have shown that all NP problems have an equivalent NP machine which uses as an alphabet $\mathbb{Z}_2$.

Definition 2.2. L $\leq_P$ M For our purposes, we say that a problem L $\leq_P$ M exactly when there is a reduction from L to M. In this case, we say that M is "at least as hard" as L. Note the opposite does not necessarily hold; if $L \leq_P M$, we do not necessarily have that $M \leq_P L$. In fact, Cook points out that the relation defined by R = $\{(L, M) \mid L, M \in NP \wedge L \leq_P M \wedge M \leq_P L\}$ is in equivalence relation, but $\leq_P$ is behaves closer to a partial ordering.

When discussing reductions, we care only about the time to convert between instances, however implicit in this model is the idea that if $L \leq_P$ M and we are able to efficiently solve M, then we can quickly solve L. Cook uses the notion of an “Oracle” which instantly decides the problem being reduced to. We can also consider this from the context of polynomial solvability. If $L \leq_P M$ and M has a polynomial time solving algorithm, then L must as well.

For a reduction from L to M, we consider the function $f : \Sigma^* \to \Sigma^*$ which converts an instance of L to an instance of M, and a function h. When given a solution for M, h maps to a solution for L. Similarly, when there is no solution for M, h outputs no solution. Thus, we adopt the following convention to diagram a reduction from L to M:

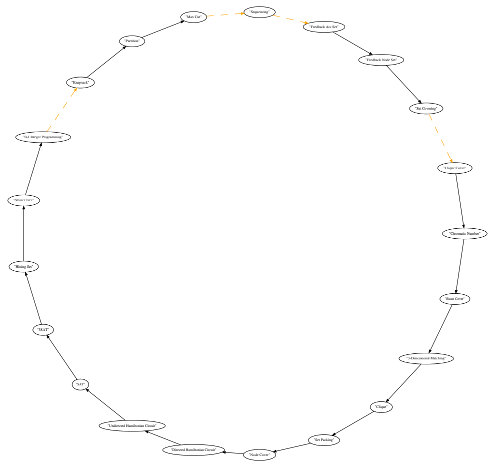
$$\overset{\text{Algorithm for L}}{\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for M}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}}$$

3 The Cycle

Figure of the cycle. And some words describing it.

I am not sure what we are doing here- are we still trying to find a simple cycle? Are we going to do something else tricky instead? Or just end with the new reductions we added?

- here is one cycle I spent some time on, somewhat fruitlessly.



4 Our Contributions to the Cycle

Theorem 4.1. *The following table displays NP problems (left) which are reducible to NP problems (right) via a single tape turing machine in polynomial time:*

3D Matching	$\leq_P$	Clique
Undirected Hamiltonian Circuit	$\leq_P$	Directed Hamiltonian Circuit
Feedback Arc Set	$\leq_P$	Set Covering
Set Packing	$\leq_P$	Node Cover
Exact Cover	$\leq_P$	Undirected Hamiltonian Circuit
Hitting Set	$\leq_P$	Steiner Tree
Satisfiability	$\leq_P$	Steiner Tree
Knapsack	$\leq_P$	0-1 Integer Programming
Clique Cover	$\leq_P$	Chromatic Number
Feedback Arc Set	$\leq_P$	Feedback Node Set
Set Covering	$\leq_P$	Steiner Tree
Hitting Set	$\leq_P$	Exact Cover
Hitting Set	$\leq_P$	Clique
Set Packing	$\leq_P$	Clique
Exact Cover	$\leq_P$	Clique
Feedback Node Set	$\leq_P$	Set Covering
Needs finished. XOR gatesMax Cut	$\leq_P$	Steiner Tree
Exact Cover	$\leq_P$	Satisfiability
Hitting Set	$\leq_P$	Satisfiability
Needs checked - Satisfiability	$\leq_P$	Knapsack
INAVLID - Satisfiability	$\leq_P$	Feedback Node Set
Needs checked- Undirected Hamiltonian Circuit	$\leq_P$	Hitting Set

4.1 3D Matching $\leq_P$ Clique

Algorithm for 3D Matching

$$\boxed{\text{I} \mapsto \text{f(I)} \to \boxed{\text{Algorithm for Clique}} \; \frac{\to x \mapsto h(x)}{\to N.S. \mapsto N.S.}}$$

3-Dimensional Matching
Input: $U \subseteq T \times T \times T$ where T is a finite set
Property: there is a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of W agree in any coordinate

Clique
Input: A graph G and a positive integer k
Property: G has a clique of size k

Details WLOG assume that U is enumerated by $\{1, ..., n\}$. Let V = $\{1, ..., n\}$ and E = $\{\{i, j\} \mid i, j$ do not overlap in a coordinate.$\}$ Let k = $|T|$.
Given a solution for clique, use the exact same elements in your solution for 3D matching.

Theorem 4.2. *There is a solution for our instance of clique $\iff$ there is a solution for the given instance of 3D matching.*

Proof. First note that as there are $\dfrac{n(n-1)}{2}$ pairings to iterate over when forming E, and for each pairing at most 3 check steps, the complexity of our reduction function is polynomial. We now proceed to prove the theorem.
$\Rightarrow$ Assume there is a solution for 3D matching. In that case, the k elements in U that do not overlap in a coordinate will all be connected to each other in our graph, and thus form a solution for clique.
$\Leftarrow$ Now suppose that there is a solution for our instance of clique. In that case, $|T|$ of the elements in U do not overlap in a coordinate, so they are a solution for 3D matching. $\square$

4.2 Undirected Hamiltonian Circuit $\leq_P$ Directed Hamiltonian Circuit

Algorithm for Undirected Hamiltonian Circuit

$$\boxed{\text{I} \mapsto \text{f(I)} \to \boxed{\text{Algorithm for Directed Hamiltonian Circuit}} \; \frac{\to x \mapsto h(x)}{\to N.S. \mapsto N.S.}}$$

Undirected Hamiltonian Circuit
Input: A graph G
Property: G has a cycle which includes each vertex exactly once.

Directed Hamiltonian Circuit
Input: A directed graph H
Property: H has a directed cycle which includes each vertex exactly once.

Details Given I = (N, E) in Undirected Hamiltonian Circuit, form f(I) as: N = N. E = $\{< u, v >, < v, u >| \ \{u, v\} \in E\}$.
Given a solution S = $n_0 n_1 \cdots n_m$ for Directed Hamiltonian Circuit, use the same sequence for Undirected Hamiltonian Circuit.

Theorem 4.3. *There is a solution for an instance I of Undirected Hamiltonian Circuit* $\iff$ *there is a solution for f(I) in Directed Hamiltonian Circuit.*

Proof. $\Rightarrow$ Assume you have a Undirected Hamiltonian Circuit solution, $n_0 n_1 \cdots n_m$. In f(I), there is a directed edge between each $n_{i \mod m+1} \rightarrow n_{i+1 \mod m+1}$. In that case, we have a Directed Hamiltonian Circuit.
$\Leftarrow$ Now suppose you have a Directed Hamiltonian Circuit, $n_0 n_1 \cdots n_m$. Then for each edge $n_{i \mod m+1} \rightarrow n_{i+1 \mod m+1}$, $\{n_{i \mod m+1}, n_{i+1 \mod m+1}\}$ is in the original edge set. In that case, our directed cycle forms a valid undirected cycle. □

4.3 Feedback Arc $\leq_P$ Set Covering

Algorithm for Feedback Arc

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Set Covering}} \ \frac{\rightarrow x \mapsto h(x)}{\rightarrow N.S. \mapsto N.S.}}$$

Feedback Arc Set
Input: A directed graph $H = (V, E)$ and a positive integer k
Property: There exists $S \subseteq E$ such that every directed cycle of H contains an arc in S and $|S| \leq k$

Set Covering
Input: A finite family of sets $\{S_j\}$ and a positive integer k
Property: There existsa subfamily $\{T_h\} \subseteq \{S_j\}$, $|\{T_h\}| \leq k$ and $\cup T_h = \cup S_j$

Details Given an instance I = $(G, k \in \mathbb{N})$ of feedback arc set, we form an instance of set covering as follows: Let $\{a_1, ..., a_n\}$ be the set of cycles in our graph, and $\{e_i\}$ be the edges. Let $\{S_j\} = \{a_i \mid a_i \text{ contains } e_j\}$, and let k = k.
Given a solution, s = $\{T_h\}$ of set covering, let h(s) = S = $\{e_j \mid S_j \in \{T_h\}\}$.
Note that finding cycles is known to be fast, and constructing $\{S_j\}$ as described here is of complexity $\mathcal{O}(|e_j| \sum |\{a_i\}|)$.

Theorem 4.4. *An instance I of feedback arc set has a solution exactly when f(I) has a solution in set covering.*

Proof. $\Rightarrow$ Assume that feedback arc has a solution. In that case, there is a set S of size $\leq k$ so that every cycle contains an arc in S. Now consider $\{T_h\} = \{S_j \mid e_j \in S\}$, which is of size k. Given $a \in \{a_i\}$, there must be some $e_j \in S$ contained in it, so $a \in \{T_h\}$.
$\Leftarrow$ Conversely, suppose that our instance of set covering has a solution. Consider the set of edges indexed by it to be S. Given a cycle in our graph, we must have some element in S appear in the cycle, or we would not have a proper cover. Thus we have a solution for feedback arc set of size $|S| = k$. □

4.4 Set Packing $\leq_P$ Node Cover

Algorithm for Set Packing

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Node Cover}} \;\frac{\rightarrow x \mapsto h(x)}{\rightarrow N.S. \mapsto N.S.}}$$

Set Packing
Input: A family of sets $\{S_j\}$ and a positive integer ℓ
Property: $\{S_j\}$ has ℓ mutually exclusive sets.

Node Cover (Vertex Cover)
Input: A graph $G' = (N', A')$ and a positive integer ℓ
Property: $\exists R \subseteq N'$ such that $|R| \leq \ell$ and every arc in incident to a vertex in R.

Details Given an instance I = $(\{S_j\}, k \in \mathbb{N})$ of set packing, let V = $\{j\}$, E = $\{\{i,k\} \mid S_i \cap S_k \neq \emptyset\}$, and k = $|\{S_j\}| - k$.
Given a solution to vertex cover, S = R, h(S) is given by $\{S_j \mid j \in R^C\}$.

Theorem 4.5. *There is a solution for instance I of set packing $\iff$ there is a solution for f(I) in node cover.*

Proof. $\Rightarrow$ Assume you have a set packing solution of size k given by $\{T_h\}$. In that case, any remaining edges in your graph must be adjacent to $\{k \mid S_k \notin \{T_h\}\}$. That set has magnitude $|\{S_j\}| - k$.
$\Leftarrow$ Alternatively, assume that you have a node cover, R, of size $|\{S_j\}| - k$. Consider the set $\{S_j \mid j \notin R\}$. This set has size $|\{S_j\}| - (|\{S_j\}| - k) = k$. Furthermore, from construction no items in that set may overlap. In that case, we have a set packing solution of size k. □

4.5 Exact Cover $\leq_P$ Undirected Hamiltonian Circuit

Algorithm for Exact Cover

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Undirected Hamiltonian Circuit}} \;\frac{\rightarrow x \mapsto h(x)}{\rightarrow N.S. \mapsto N.S.}}$$

Exact Cover
Input: A family $\{S_j\}$ of subsets of the set $\{u_1, u_2, \ldots u_t\}$
Property: there is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that $\{T_h\}$ are disjoint and $\cup T_h = \cup S_j = \{u_1, u_2, \ldots u_t\}$

Undirected Hamiltonian Circuit
Input: A graph G
Property: G has a cycle which includes each vertex exactly once.

Details Given an instance I = $\{S_j\}$ of exact cover, construct an instance f(I) of Undirected Hamiltonian Circuit as follows: Let V = $\{out - j, in - j\} \cup \{u_i \in \bigcup S_j\}$. For each S_j, let ϕ_j be a cycle on its elements. Let E = $\{\{u_i, \phi_j(u_i)\} \mid u_i \in S_j\} \cup \{\{u_i, in - j\}, \{\phi_j^m(u_i), out - j\} \mid u_i \in$

S_j, m is the degree of $\phi_j\} \cup \{\{in - j, out - k\}\}$.
Given a solution for Undirected Hamiltonian Circuit, let $\{T_h\}$ be equal to $\{S_j \mid in{-}j$ goes to some u_i in the path$\}$.

Theorem 4.6. *There is a solution for an instance I of exact cover* $\iff$ *there is a solution for f(I) in Undirected Hamiltonian Circuit.*

Proof. $\Rightarrow$ Assume you have an exact cover solution, $\{T_h\}$. For each $S_j \in \{T_h\}$, include $in - j \rightarrow u_{j_1} \rightarrow u_{j_2} \rightarrow \cdots \rightarrow out - j$ in the path. This is acceptable since the elements in S_j are adjacent to each other, and to the in-j, out-j nodes. Now let π be any cycle you please on the S_j, and complete the Undirected Hamiltonian Circuit by attaching $out - k$ to $in - \pi(k)$. Note that we have covered all of our elements, and reused none of our vertices.
$\Leftarrow$ Now let S = $n_0 n_1 \cdots n_m$ be a complete cycle on our path. Now consider our proposed solution, h(S). For a contradiction, suppose that we cannot cover all of the elements in $\bigcup S_j$ without overlap. In that case, for any two sets, they must have one element in common. Note that to include every element we must traverse through one of the sets containing it completely, due to our cycle restriction. But if we always have some overlap, during traversal of the cycle of elements in a set we must run into a re-used node, and thus not have a Undirected Hamiltonian Circuit. Then there must be some configuration of an exact cover. □

4.6 Hitting Set $\leq_P$ Steiner Tree

Algorithm for Hitting Set

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Steiner Tree}} \;\; \frac{\rightarrow x \mapsto h(x)}{\rightarrow N.S. \mapsto N.S.}}$$

Hitting Set
Input: A family $\{U_i\}$ of subsets of $\{s_1, s_2, \ldots s_r\}$
Property: there is a set W such that for each i, $|W \cap U_i| = 1$

Steiner Tree
Input: A graph $G = (N, A)$, $R \subseteq N$, a weighting function $w : A \rightarrow \mathbb{Z}$ and a positive integer k
Property: G has a subtree of weight $\leq k$ containing the set of vertices in R

Details Given an instance $\{U_i\}$ of hitting set, we construct our instance of steiner tree as follows via f: Let k = $|\{U_j\}|$. Let V = $\{n_0\} \cup \{u_i\} \cup \{U_{ij} \mid u_i \in U_j\} \cup \{U_j\} \cup \{U_j^{balancer}\}$. Let R = $\{n_0\} \cup \{U_j\}$. Draw the edges as: connect n_0 to each of the u_i via an edge with weight 0. For each u_i, create a chain between its U_{ij} nodes where each edge has weight 1. From the last node in that chain, connect it to each of the $\{U_j\}$ nodes which show up in the chain via edges of weight k + 1. Draw an edge between each $\{U_j\}$ and $\{U_j^{balancer}\}$ with weight -(k + 1).
Given a solution for steiner tree, construct W as the set of u_i connected to n_0.

Theorem 4.7. *There is a solution for an instance I of hitting set* $\iff$ *there is a solution for f(I) in steiner tree.*

Proof. $\Rightarrow$ Suppose you have a solution for hitting set, W. For each $u_i \in W$, include $\{u_i, n_0\}$ in our tree. So far, our tree has weight 0. Next, attach each $\{u_i\}$ to its entire chain, and then to the matching terminal vertices. We know the connection between the chain and the terminal vertices

will have weight 0. We are left to verify that all of them are covered, and that the total weight of our included chain sections is $\leq k$. Let's address the first requirement first. Because we have a hitting set, $W \cap U_i \neq \emptyset \forall i$. That is, for each terminal vertex like U_j, it must be connected to the chain of some $u_i \in W$. Now we verify the total weight of the included chains. Assume for a contradiction that it is $> k$. In that case, by the pigeonhole principle some distinct u_i, u_j have a shared U_j in their chain. But the chains are constructed from the sets they appear in! If this were the case, we would not have a hitting set. Then we must have a solution for Steiner Tree.
$\Leftarrow$ Assume you have a solution for f(I). In that case, we verify W. First, note that for each U_i, $U_i \cap W \neq \emptyset$, since we must get to the U_i after traversing a chain attached to one of our elements. Note that we cannot access the U_i except by first traversing a chain, since you would end up with an additional unbalanced k + 1 in your weight sum trying to skip between the nodes. Furthermore, if $|U_i \cap W| > 1$ then our tree would have weight $> k$ and would not be a solution in f(I). We conclude that we have a hitting set. $\square$

4.7 Satisfiability $\leq_P$ Steiner Tree

Algorithm for Satisfiability

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Steiner Tree}} \; \frac{\rightarrow x \mapsto h(x)}{\rightarrow N.S. \mapsto N.S.}}$$

Satisfiability
Input: Clauses $C_1, C_2, \ldots C_p$ with literals $x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}$.
Property:
$\exists S \subseteq \{x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}\}$ such that if $x_i \in S$ then $\overline{x_i} \notin S$, and if $\overline{x_i} \in S$ then $x_i \notin S$ and $S \cap C_j \neq \emptyset$, $\forall j \in \{1, 2, \ldots p\}$

Steiner Tree
Input: A graph $G = (N, A)$, $R \subseteq N$, a weighting function $w : A \rightarrow \mathbb{Z}$ and a positive integer k
Property: G has a subtree of weight $\leq k$ containing the set of vertices in R

Details Given an instance I = $(\{C_1, ..., C_p\}, \{x_1, ..., x_r, \bar{x}_1, ..., \bar{x}_r\}$, we form an instance f(I) of Steiner Tree as:
Let k = 1. Let N = $\{n_1, n_0, x_1, ..., x_r, \bar{x}_1, ..., \bar{x}_r, x_1^{end-0}, x_1^{end-1}, ..., x_r^{end-0}, x_r^{end-1}, C_1, ..., C_p, C_1^{end}, ..., C_p^{end}\}$.
Let R = $\{n_0, n_1\} \cup \{x_i^{end}\} \cup \{C_j\}$. This is where it gets real wild folks, now we add an edge between n_0 and n_1 with weight 1. From n_1, add an edge to each x_i, $\bar{x}_i$ with weight 1. For each $x_i, \bar{x}_i$ connect them to x_i^{end-0} with weight 1. Connect x_i^{end-0} to x_i^{end-1} with weight -2. Now for each C_p, connect it with weight 1 to any literal contained in it. Furthermore, connect it with weigth -1 to C_p^{end}. This completes our construction.
Given a solution for Steiner Tree, we form a solution S for Satisfiability as the set of literals included in the path to the $\{C_j\}$ nodes.

Theorem 4.8. *There is a solution for an instance I of Satisfiability $\iff$ there is a solution for f(I) in Steiner Tree.*

Proof. $\Rightarrow$ Assume Satisfiability has a solution, S. We construct a solution for Steiner Tree: for each C_j, connect it to C_j^{end} and one of the literals in S which satisfy it. This has net weight 0. For x_i

in S, include the edges between n_0 to x_i, and from x_i to x_i^{end-0}, and between x_i^{end-1}. It is quick to check that this has net weight 0. Note that the C_j will be accessible from the set of literals in S. For any unassigned literals, pick either path from n_1 to the end-0, and then to end-1. This has net weight 0. Finally, include the edge between n_0 and n_1 for a net weight of 1. The sum of all weights is 1. Furthermore, every vertex in R has been covered and is accessible from a tree starting at n_0.
$\Leftarrow$ Now suppose that you have a solution for f(I) in Steiner Tree. Let S = $\{\sigma \mid \{\sigma, C_j\}$ is a path in our solution for some $C_j\}$. We verify that this is a solution for Satisfiability. First, for each C_i, $C_i \cap S \neq \emptyset$, since the C_i are required vertices in our tree and are only accessible from x_i or $\bar{x}_i$ nodes. Now we confirm that there are no conflicting literals. Assume for a contradiction that some x_i and $\bar{x}_i$ are in S. In that case, either each $\{n_1, x_i\}$ and $\{n_1, \bar{x}_i\}$ both show up in the tree, or (WLOG) $\bar{x}_i$ is accessed by a C_j that is already accessed by another literal node. In the first case, we end up with a positive balance of weight, because we use 2 to arrive at each literal and we need +1 to arrive at x_i^{end}. The available offset is -2, so we net +1. In the second, the other literal accessing C_j adds +1, and $\bar{x}_i$ adds + 1. The offset available is -1, so we net + 1. Finally, we note that using a literal must result in a net weight of at least 0 in every case. Our Steiner Tree path would have weight ≥ 2, which is above our threshold. In that case, we may have no conflicting literals in S. $\square$

4.8 Knapsack $\leq_P$ 0-1 Integer Programming

Algorithm for Knapsack

$$\boxed{\text{I} \mapsto \text{f(I)} \to \boxed{\text{Algorithm for 0-1 Integer Programming}} \begin{array}{c} \to x \mapsto h(x) \\ \hline \to N.S. \mapsto N.S. \end{array}}$$

Knapsack
Input: $(a_1, a_2, \ldots a_r, b) \in \mathbb{Z}^{r+1}$
Property: $\sum a_j x_j = b$ has a 0-1 solution

0-1 Integer Programming
Input: A integer matrix C and an integer vector $\vec{d}$
Property: there is a 0-1 vector $\vec{x}$ such that $C\vec{x} \geq \vec{d}$

Details Given an instance $(a_1, ..., a_r, b)$ of knapsack, we form an instance of 0-1 integer programming as follows: Let C be a (2 × r) matrix. In that case, d must be a (2 × 1) matrix. Define each as follows:

$$\begin{aligned} c_{1i} &= a_i \\ d_{11} &= b \\ c_{2i} &= -a_i \\ d_{21} &= -b \end{aligned}$$

Given a solution s = $\vec{x}$ of integer programming, we form h(s) as $x_i = x_i$.

Theorem 4.9. *There is a solution for instance I of knapsack exactly when there is a solution for f(I) in integer programming.*

Proof. You have a solution for knapsack $\iff \sum a_i x_i = b \iff b \leq \sum a_i x_i \leq b \iff \sum a_i x_i \leq b$ and $\sum a_i x_i \geq b \iff \sum -a_i x_i \geq -b$ and $\sum a_i x_i \geq b$. We see that these are precisely the conditions required to solve our instance of 0-1 programming. □

4.9 Clique Cover $\leq_P$ Chromatic Number

Algorithm for Clique Cover

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Chromatic Number}} \; \frac{\rightarrow x \mapsto h(x)}{\rightarrow N.S. \mapsto N.S.}}$$

Clique Cover
Input: A graph $G' = (N', E')$ and a positive integer ℓ
Property: N' is the union of ℓ or fewer cliques

Chromatic Number
Input: A graph $G = (N, E)$ and a positive integer k
Property: there is a function $\phi : N \rightarrow \mathbb{Z}_k$ such that if u and v are adjacent then $\phi(u) \neq \phi(v)$

Details Given an instance of clique cover, I = (G', l) we form an instance of chromatic number f(I) as: G = G'^C, k = l.
Given a solution for chromatic number, S, h(S) is the sets of nodes colored the same under S.

Theorem 4.10. *There is a solution for instance I of clique cover $\iff$ there is a solution for f(I) in chromatic number.*

Proof. $\Rightarrow$ Assume you have a solution for clique cover. In that case, you have l distinct cliques which union to cover the nodes. Enumerate these sets as $C_1, C_2, ..., C_l$. We construct $\phi : N \rightarrow \mathbb{Z}_k$ as $\phi(n) =$ the largest j so that $n \in C_j$. Since our clique cover covers the nodes, our map is defined over the domain. Further, since k = l the codomain for our map is appropriate. Finally, we verify that for u, v that are adjacent, $\phi(u) \neq \phi(v)$. Since u,v are adjacent in the complement, they must not have been in G'. In that case, they do not show up in any of the same C_j. Then we must not have $\phi(u) = \phi(v)$. We have verified our solution for chromatic number.
$\Leftarrow$ Now suppose you have a solution for chromatic number. Let your clique cover, $\{C_j\}$ be defined so that $C_j = \phi^{-1}(j)$. There may be no more than l such sets, since the domain of ϕ is $\mathbb{Z}_k = \mathbb{Z}_l$. We verify that each C_j is a clique: Given u, v $\in C_j$, u and v were not adjacent in the complement to receive the same coloring. This implies that they are adjacent in our graph. Since they were arbitrary, any two nodes in a C_j are adjacent, verifying each is a clique. Finally, each node must be included since ϕ is defined from N. Thus we have a solution for clique cover. □

4.10 Feedback Arc Set $\leq_p$ Feedback Node Set

Algorithm for Feedback Arc Set

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Feedback Node Set}} \; \frac{\rightarrow x \mapsto h(x)}{\rightarrow N.S. \mapsto N.S.}}$$

Feedback Arc Set
Input: A directed graph $H = (V, E)$ and a positive integer k
Property: There exists $S \subseteq E$ such that every directed cycle of H contains an arc in S and $|S| \leq k$

Feedback Node Set
Input: A directed graph $H = (V, E)$ and a positive integer k
Property: There exists $R \subseteq V$ such that every directed cycle of H contains a vertex in R and $|R| \leq k$

Details I: A graph $H = (V, E)$ and a positive integer k.

f: Function f takes the input I and convert it into an instance of Feedback Node Set problem where H' is a directed line graph of H with

$V' = \{(u, v)|(u, v) \in E\}$
$E' = \{< (u, v), (v, w) > |\{u, v, w\} \in V \text{ and } < u, v >, < v, w > \in E\}$
$k = l$

Algorithm for Feedback Node Set: Suppose there is a polynomial time algorithm for the Feedback Node Set problem.

S: $R \subseteq E'$ such that every directed cycle of H' contains a vertex in R and $|R| \leq k$

h(S): function h takes S as an input and return the respective edges from G.

Lemma 4.11. *If there exist a cycle between nodes of H, then there exist a cycle in H' between nodes corresponding to the edges between the nodes in H which form a cycle.*

Proof. ($\Rightarrow$) Suppose that we construct graph H' as described above. Suppose that there is graph that has a path from node u to node w with n number of nodes in between u and w in H. The path from u to w will be followed by directed edges $\{< u, 1 >, < 1, 2 >, \ldots, < n-1, n >, < n, w >\}$. According to our construction the line graph H' will have a sub graph with the nodes $\{(u, 1), (1, 2), \ldots, (n-1, n), (n, w)\}$. There will be edges $\{< (u, 1), (1, 2) >, < (2, 3), (3, 4) >, \ldots, < (n-1, n), (n, w) >\}$. Thus if there exists a path from u to w with n nodes between them then there will be a path in the line graph H' whose nodes represents the edges (path) of H. Lets add an edge between$< w, u >$ between nodes w and u in graph H. Thus there exists a cycle between nodes u, the n nodes and w back to u in H. If we add an edge in H we have to add a node representing that edge in H' and connect it to nodes it needs to be connected with. According to our constructed graph node (w, u) will have a directed edge coming in to it from node (n, w) and (w, u) will have a directed edge going out from itself to node $(u, 1)$. Since there already exists a path from $(u, 1)$ to (n, w) and we added a path from (n, w) to $(u, 1)$ through (w, u) it will form a cycle between nodes $\{(u, 1), (1, 2), \ldots, (n, w), (w, u)\}$. Therefor H' has a cycle. □

Theorem 4.12. *$H = (V, E)$ has a Feedback Arc Set of size l, iff there is a feedback arc set of size k on the graph, $H' = (V', E')$, as defined above.*

Proof. ($\Rightarrow$) Let us assume there exist a feedback arc set, S, for the graph H of size k. Lets construct graph H' as described above. Lets claim that for H', $R = \{(u, v)|(u, v) \in S\}$. Let us assume that removing all the nodes and their connected edges from R will not break all the cycles in the graph. If that is true then R will not be a feedback node set for the graph H'. According to our construction any node (u, v) in graph H' will have directed edges connecting other nodes which represent edges connected to the nodes u and v in H. So if edge $< u, v >$ was a part of a cycle (i.e.

nodes u and v were part of a cycle) with n edges in H it's corresponding node in H' should also be a part of a cycle with n nodes. Thus if removing edge $< u, v > \in S$ from H breaks all the cycles in the graph it should also break all the cycles in H', which is a contradiction to our assumption. Thus $R = \{(u, v) | (u, v) \in S\}$ is the feedback node set for H'.

($\Leftarrow$) Let us assume that H' has a feedback node set R of size l. According to our construction a node (u, v) in H' is connected to nodes which represent directed edges which are connected to either u or v in H. Let us assume that for H, $S =< u, v > |(u, v) \in R$. Further for the sake of contradiction let us assume that removing all the edges in S will not break all the cycles in H. If that is true then S is not a feedback arc set for the graph H. According to our construction if any node (u, v) is a part of a cycle in H' with n other nodes, its corresponding edge $< u, v >$ will also be in a cycle with the edges represented by the n nodes in H. And if removing nodes $(u, v) \in R$ breaks all the cycles in graph H', it should also break all the cycles if the same edges represented by the removed nodes in R are all the cycles in H should also be broken. This contradicts our assumption, thus edges $S =< u, v > |(u, v) \in R$ is the feedback arc set of H. ☐

4.11 Set Covering $\leq_P$ Steiner Tree

Algorithm for Set Covering

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Steiner Tree}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Set Covering
Input: A finite family of sets $\{S_j\}$ and a positive integer k
Property: There existsa subfamily $\{T_h\} \subseteq \{S_j\}$, $|\{T_h\}| \leq k$ and $\cup T_h = \cup S_j$

Steiner Tree
Input: A graph $G = (N, A)$, $R \subseteq N$, a weighting function $w : A \to \mathbb{Z}$ and a positive integer k
Property: G has a subtree of weight $\leq k$ containing the set of vertices in R

Details -Let $I = (\{S_j\}, k)$, where $\{S_j\}$ is a family of subsets and k is a positive integer. Then f will define a graph G, with vertices,
$V = \{n_o\} \cup \{S_j\} \cup \{u_i\}$, and Edges,
$E = \{\{n_o, S_j\}\} \cup \{\{S_j, u_i\} | u_i \in S_j\}$
f will also define $R \subseteq V = \{n_o\} \cup \{u_i\}$
f will also define w, a weighting function, $w(e) = 1$ for all $e \in E$
f will also define an integer, $W = |\{u_i\}| + k$.
-Suppose there is polynomial time algorithm for Steiner Tree, that returns Solution S, a subtree of G of weight $\leq k$ containing the set of vertices R.
-$h(S)$ will take solution S, a subtree of G. h will loop through this subtree, taking vertices labeled S_i with $1 \leq i \leq j$, and adding them to a set R. This list R will then be returned.
-No solution implies no solution.

Theorem 4.13. *$\{S_j\}$ has a set covering of size k iff Graph $G = (V, E)$, with weighting function $w : E \mapsto \mathbb{Z}$, and integer W, as defined above, form a Steiner Tree on Graph G.*

Proof. $\Rightarrow$ Suppose there is a set covering, T_h of size k on $\{S_j\}$. Now, take Graph G, as defined above, with $R \subseteq V$, W, and weighting function w. According to our construction, the nodes in

R are $\{n_o\}$, and $\{u_i\}$, $w : E \mapsto \mathbb{Z} = 1$ for all $e \in E$, and $W = k + |\cup\{S_j\}|$. In addition, there are edges between $\{n_o\}$ and each $\{S_j\}$, and each $\{S_j\}$ to each $\{u_i\}$ that represents an element of $\{S_j\}$. In order for each of the nodes in R to be covered by the Steiner Tree, there must be an edge chosen between each $\{u_i\}$ and a $\{S_j\}$. Since we know $\{S_j\}$ has a set covering, T_h, we can connect each $\{u_i\}$ with exactly one arbitrary $\{S_j\}$ that represents a set in T_h. Since T_h is a set covering on S_j, there is guaranteed to be at least one edge from every $\{u_i\}$ to an $\{S_j\}$ that is also in $\{T_h\}$. If exactly one edge is chosen for each node representing an element in $\cup S_j$, then the total weight will be equal to $|\cup\{S_j\}|$, since each edge has a weight of 1. Now, if we include the edges from $\{n_o\}$ to each $\{S_j\}$ in T_h, then every node in R will be contained in the subtree, and the edges chosen will form a complete subtree, as every set in $\{T_h\} \in \{S_j\}$ will be accessible from n_o, and every $\{u_i\}$ is accessible from the sets in $\{T_h\} \in \{S_j\}$. Further, since there are k sets in T_h, the total weight is now $k + |\{S_j\}|$, which is equal to W. Thus, there is a Steiner Tree on G.
$\Leftarrow$ Suppose there is a Steiner Tree on G, with set $R \subseteq V$, weighting function $w : A \mapsto \mathbb{Z}$, and integer W, all constructed as defined above. Since the elements chosen to be in R in our construction were $\{n_o\}$ and $\{u_i\}$, these must be accessible through the Steiner Tree. Further, since we have a Steiner Tree the total weight of the edges included must be $\leq W$. Since each of the $\{u_i\}$ must be accessed, and the only access to them is through edges of weight 1 to the clauses they are a part of, the Steiner Tree must have at least weight $\{u_i\}$. We know that the total weight must be $\leq W$, which is equal to $\{u_i\} + k$. Since we already have weight $\{u_i\}$, the remaining edges that were chosen must have weight $\leq k$. Since the weight from $\{n_o\}$ to each node representing a set is equal to 1, k or less of these nodes representing sets must have been chosen. These $\{S_j\}$ that were chosen must also have a connection to every $\{u_i\}$. Thus, k or less sets can be chosen in $\{S_j\}$ such that every element in $\{u_i\}$ is a part of the chosen sets, which, by definition is a set covering. □

4.12 Hitting Set $\leq_P$ Exact Cover

Algorithm for Hitting Set

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Exact Cover}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Hitting Set
Input: A family $\{U_i\}$ of subsets of $\{s_1, s_2, \ldots s_r\}$
Property: there is a set W such that for each i, $|W \cap U_i| = 1$

Exact Cover
Input: A family $\{S_j\}$ of subsets of the set $\{u_1, u_2, \ldots u_t\}$
Property: there is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that $\{T_h\}$ are disjoint and $\cup T_h = \cup S_j = \{u_1, u_2, \ldots u_t\}$

Details -Let $I = \{U_i\}$, a family of subsets of a set $\{s_j, j = 1, 2, \ldots t\}$. Then f will define a family of subsets, $\{S_j\}$, such that $u_i \in S_j$ iff $s_j \in U_i$. (i.e. $\{S_j\}$ will contain j sets, $j = |\cup U_i|$, with elements representing the sets that each element represented by S_j are in in $\{U_i\}$).
-Suppose there is a polynomial time algorithm for Exact Cover, that returns Solution S, a subset of S_j such that $\cup S = \cup S_j$ and S is disjoint.
-$h(S)$ will take in a set of integers, S, representing an Exact Cover on $\{S_j\}$. This same set can be

returned, as the same integers represent the elements on $\{U_i\}$ that form a hitting set.
-No solution implies no solution

Theorem 4.14. *$\{U_i\}$ has a hitting set iff $\{S_j\}$, as constructed above, has an exact cover.*

Proof. $\Rightarrow$ Suppose there is a solution, W, for hitting set on U_i such that $|W \cap U_i| = 1$. S_j is defined with $u_i \in S_j$ iff $s_j \in U_i$. Thus, since the elements in U_i represent sets in S_j, and the sets in U_i represent elements in S_j, if there is a subset of elements in $s_j \in U_i$ that can intersect with each set in U_i with a cardinality of 1, that means there is a subset of sets in S_j that contains each element in S_j exactly once, which, by definition, is an exact cover.
$\Leftarrow$ Suppose there is an exact cover on S_j. That is, there is some subset, T_h, that is disjoint and $\cup T_h = \cup S_j$. Given our construction, $s_j \in U_i$ iff $u_i \in S_j$. By definition, an exact cover on S_j means that some subset of sets can be chosen such that each element in $\cup S_j$ appears exactly once. On U_i, this means that some subset of elements in $\cup U_i$ can be chosen such that each set in $\cup U_i$ contains exactly one element. By definition, this is a solution for hitting set. ☐

4.13 Hitting Set $\leq_P$ Clique

Algorithm for Hitting Set

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Clique}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Hitting Set
Input: A family $\{U_i\}$ of subsets of $\{s_1, s_2, \dots s_r\}$
Property: there is a set W such that for each i, $|W \cap U_i| = 1$

Clique
Input: A graph G and a positive integer k
Property: G has a clique of size k

Details -Let $I = \{U_i\}$, a family of subsets of a set $\{s_j, j = 1, 2, \dots t\}$. Then f will define a Graph, $G = (V, E)$, with
$V = \{< i, j > |u_i \in S_j, S_j \in \{S_j\}\}$ (i.e. each element of each set gets a node, for example if $S_1 = (1, 2), S_2 = (1, 3, 5)$, $V = (1, 1), (2, 1), (1, 2), (3, 2), (5, 2))$. and
$E = \{<< i, j >, < k, l > |j \neq l$ and $(i = k$ or (if $i \in S_j, k \notin S_j$ for all $S_j \in \{S_j\}$ and if $k \in S_j, i \notin S_j$ for all $S_j \in \{S_j\}))\}$ (i.e. edges are only between nodes from different sets, and there are only edges if two nodes represent the same element in $\cup\{S_j\}$ or if the two elements do not appear in any of the same sets in $\{S_j\}$.
f will also define an integer, $k = |\{S_j\}|$.
-Suppose there is a polynomial time algorithm for Clique, that returns solution S, a subset of $G = (V, E)$ that represents a clique of size k on G.
-$h(S)$ will take in solution S, and loop through it, adding the element that each node represents to set W if it hasn't already been added (no duplicates). W will then be returned.
-No solution implies no solution.

Theorem 4.15. *$\{U_i\}$ has a hitting set iff $G = (V, E)$, constructed as defined above, has a clique of size k.*

Proof. $\Rightarrow$ Suppose $\{S_j\}$ has a hitting set. That is, there is some set, $W \subseteq \cup S_j$, such that $|W \cap S_j| = 1$ for all $S_j \in \{S_j\}$. By our construction, each element in every S_j has a node, and nodes are connected if they represent the same element in u_i, or if they do not appear in any of the same sets in S_j. Thus, if the nodes representing the element in each that is part of the solution W is chosen to be in the clique, there will be an edge to every other node representing elements in solution W. This is because the only way nodes are not connected is if the u_i they represent appear in any set together. Further, since we need a clique of size $|\{S_j\}|$, since a hitting set requires one element from each set, there will be $|\{S_j\}|$ elements in the clique. Thus, we have a clique of size $|\{S_j\}|$ on G.
$\Leftarrow$ Suppose we have a clique of size $|\{S_j\}|$ on G. Further suppose G was constructed as defined above. Since edges were only assigned if 2 elements were in different sets, had the same element, or don't appear in any sets together, we know that a clique of size $|\{S_j\}|$ means that each set contains exactly one element in the clique. If each element in the clique is chosen to be in W, then $|W \cap S_j| = 1$ for all $S_j \in \{S_j\}$, which, by definition, is a hitting set on $\{S_j\}$. $\square$

4.14 Set Packing $\leq_P$ Clique

Algorithm for Set Packing

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Clique}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Set Packing
Input: A family of sets $\{S_j\}$ and a positive integer ℓ
Property: $\{S_j\}$ has ℓ mutually exclusive sets.

Clique
Input: A graph G and a positive integer k
Property: G has a clique of size k

Details -Let $I = \{S_j\}, k$, a family of subsets $\{S_j\}$ and an integer k. Then f will define a Graph, $G = (V, E)$, with
$V = \{S_j | S_j \in \{S_j\}\}$ (i.e. one node for each set in S_j) and
$E = \{< S_j, S_i > | S_j \text{ and } S_i \text{ are mutually exclusive}\}$
-Suppose there is a polynomial time algorithm for Clique, that returns solution S, a subset of $G = (V, E)$ that represents a clique of size k on G.
-$h(S)$ will take in solution S, a subset of G representing a clique of size k on G. $h(S)$ will return this same subset, as the nodes were named to represent sets in $\{S_j\}$, and the same nodes in this clique represent a set packing for $\{S_j\}$.
-No solution implies no solution.

Theorem 4.16. *$\{S_j\}$ has a set packing of size k iff $G = (V, E)$, constructed as defined above, has a clique of size k.*

Proof. $\Rightarrow$ Suppose there is a solution for set packing on $\{S_j\}$. That is, there are k sets in $\{S_j\}$ that are mutually exclusive. G, constructed as defined above, has a node for each set in S_j, and edges only if the sets the nodes represent are mutually exclusive. Thus, since the nodes representing sets in the solution for set packing are mutually exclusive, there will be edges between each node, and they will form a clique of size k.

$\Leftarrow$ Suppose G, constructed as defined above, has a clique of size k. Since edges in G are only there if the two sets are mutually exclusive, this means there are k mutually exclusive sets on $\{S_j\}$, which, by definition, is a Set Packing. $\square$

4.15 Exact Cover $\leq_P$ Clique

Algorithm for Exact Cover

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Clique}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Exact Cover
Input: A family $\{S_j\}$ of subsets of the set $\{u_1, u_2, \dots u_t\}$
Property: there is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that $\{T_h\}$ are disjoint and $\cup T_h = \cup S_j = \{u_1, u_2, \dots u_t\}$

Clique
Input: A graph G and a positive integer k
Property: G has a clique of size k

Details -Let $I = \{S_j\}$, a family of subsets of a set $\{s_j, j = 1, 2, \dots t\}$. Then f will define a Graph, $G = (V, E)$, with
$V = \{< i, j > | u_i \in S_j, S_j \in \{S_j\}\}$ (i.e. each element of each set gets a node, for example if $S_1 = (1, 2), S_2 = (1, 3, 5)$, $V = (1, 1), (2, 1), (1, 2), (3, 2), (5, 2))$. and
$E = \{<< i, j >, < k, j >> \mid \text{for all } u_i, u_j \in S_j\} \cup \{<< i, j >, < k, l > \mid$ if S_j and S_l are mutually exclusive. f will also define an integer, $k = |\{u_i\}|$ (i.e. the number of elements in $\cup S_j$
-Suppose there is a polynomial time algorithm for Clique, that returns solution S, a subset of $G = (V, E)$ that represents a clique of size k on G.
-$h(S)$ will take in solution S, and loop through it, adding the set that each node represents to set R if it hasn't already been added (no duplicates). W will then be returned.
-No solution implies no solution.

Theorem 4.17. *$\{S_j\}$ has an exact cover iff $G = (V, E)$, constructed as defined above, has a clique of size k.*

Proof. $\Rightarrow$ Suppose there is an exact cover, T_h on $\{S_j\}$. That is, there is some set, T_h, such that $\cup T_h = \cup S_j$ and T_h is disjoint. By our construction, nodes are only connected if they are in the same set, or if their set is mutually exclusive with another nodes set. Thus, if the sets in the exact cover's elements are chosen to represent a clique, each node from these sets will be connected because the sets are mutually exclusive. Further, since the sets in the exact cover cover every element in $\cup S_j$, there will be a clique of size $|\{u_i\}|$.
$\Leftarrow$ Suppose there is a clique of size k on G. Further suppose G was constructed as defined above. By our construction, edges are only between elements of the same set, and between elements of mutually exclusive sets. Thus, in order for there to be a clique of size k, there must be a subset of sets that is disjoint and that covers k elements. Since k is defined to equal $|\{u_i\}|$, each element is covered by the sets in the clique, thus there is an exact cover on $\{S_j\}$. $\square$

4.16 Feedback Arc Set $\leq_p$ Set Covering

Algorithm for Feedback Arc Set

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Set Covering}} \; \frac{\rightarrow x \mapsto h(x)}{\rightarrow N.S. \mapsto N.S.}}$$

Feedback Arc Set
Input: A directed graph $H = (V, E)$ and a positive integer k
Property: There exists $S \subseteq E$ such that every directed cycle of H contains an arc in S and $|S| \leq k$

Set Covering
Input: A finite family of sets $\{S_j\}$ and a positive integer k
Property: There existsa subfamily $\{T_h\} \subseteq \{S_j\}$, $|\{T_h\}| \leq k$ and $\cup T_h = \cup S_j$

Details Given an instance I = $(G, k \in \mathbb{N})$ of feedback arc set, we form an instance of set covering as follows: Let $\{a_1, ..., a_n\}$ be the set of cycles in our graph, and $\{e_i\}$ be the edges. Let $\{S_j\} = \{a_i \mid a_i$ contains $e_j\}$, and let k = k.
Given a solution, s = $\{T_h\}$ of set covering, let h(s) = S = $\{e_j \mid S_j \in \{T_h\}\}$.
Note that finding cycles is known to be fast, and constructing $\{S_j\}$ as described here is of complexity $\mathcal{O}(|e_j| \sum |\{a_i\}|)$.

Theorem 4.18. *An instance I of feedback arc set has a solution exactly when f(I) has a solution in set covering.*

Proof. $\Rightarrow$ Assume that feedback arc has a solution. In that case, there is a set S of size $\leq k$ so that every cycle contains an arc in S. Now consider $\{T_h\} = \{S_j \mid e_j \in S\}$, which is of size k. Given $a \in \{a_i\}$, there must be some $e_j \in S$ contained in it, so $a \in \{T_h\}$.
$\Leftarrow$ Conversely, suppose that our instance of set covering has a solution. Consider the set of edges indexed by it to be S. Given a cycle in our graph, we must have some element in S appear in the cycle, or we would not have a proper cover. Thus we have a solution for feedback arc set of size $|S| = k$. □

4.17 Feedback Node Set $\leq_P$ Set Covering

Algorithm for Feedback Node Set

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Set Covering}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Feedback Node Set
Input: A directed graph $H = (V, E)$ and a positive integer k
Property: There exists $R \subseteq V$ such that every directed cycle of H contains a vertex in R and $|R| \leq k$

Set Covering
Input: A finite family of sets $\{S_j\}$ and a positive integer k
Property: There existsa subfamily $\{T_h\} \subseteq \{S_j\}$, $|\{T_h\}| \leq k$ and $\cup T_h = \cup S_j$

Details -Let $I = (G = (V, E), k)$, where G is a directed graph, and k is an integer. Then f will define $\{S_j\}$ a family of sets. Each $v_j \in V$ will correspond to a set, $S_j \in \{S_j\}$, with the elements of the set being the cycles in G that the node represented by the set is a part of in G. f will also define an integer, $k = k$.
-Suppose there is a polynomial time algorithm for Set Covering, that returns solution S, a subset of sets in S_j that represent a set covering of size k on S_j.
-$h(S)$ will take in solution S, a subset of sets in S_j that represents a set covering of size k on S_j. Since each set in S_j represents a node, each set in S will be converted back to the node that it represents on G, and added to set R. The set R will be returned.
-No solution implies no solution.

Theorem 4.19. *$G = (V, E)$ has a Feedback Node Set iff $\{S_j\}$, constructed as defined above, has a Set Covering of size k.*

Proof. $\Rightarrow$ Suppose there is a Feedback Node Set of size k on G. That is, there are k nodes that are contained in every cycle in G. By our construction, sets in S_j are constructed for each node in G, with elements being the cycles that that node appears in. Thus, since the elements of S_j are cycles, $\cup S_j$ is the set of all cycles on G. Therefore, if the same nodes in the Feedback Node Set are chosen for the set covering, k sets can be chosen such that every element in $\cup S_j$ is chosen.
$\Leftarrow$ Suppose there is a set covering on S of size k. Since the sets represent nodes and the elements represent the cycles that node is in, there is some set of k nodes contained in each cycle, which, by definition, is a Feedback Node Set. □

4.18 Max Cut $\leq_P$ Steiner Tree

Algorithm for Max Cut

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Steiner Tree}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Max Cut
Input: A graph $G = (N, A)$, a weight function $w : A \to \mathbb{Z}$ and a positive integer W
Property: $\exists S \subseteq N$ such that $\sum_{\{u,v\} \in A, u \in S, v \notin S} w(\{u, v\}) \geq W$

Steiner Tree
Input: A graph $G = (N, A)$, $R \subseteq N$, a weighting function $w : A \to \mathbb{Z}$ and a positive integer k
Property: G has a subtree of weight $\leq k$ containing the set of vertices in R

Details -Let $I = (G = (V, E), k \in \mathbb{Z}^+, W : E \mapsto \mathbb{Z})$, where G is an undirected, weighted graph, k is a positive integer, and W is a weighting function for the edges of G. Then f will define a weighted graph, $H = (N, A)$, with Vertexes,
$N = \{n_i | 0 \leq i \leq 1\} \cup \{S_i | \forall V_i \in V\} \cup \{W_i | \forall V_i \in V\} \cup \{< i, j > | V_i \in V, V_j \in V, i \neq j\}$ and Edges, $A = \{< n_0, n_1 >\} \cup \{< n_1, S_i > | S_i \in N\} \cup \{< S_i, W_i > | S_i, W_i \in N\} \cup \{< W_k, < i, j >> | k = i$ or $k = j\}$, with weighting function $W : A \mapsto Z$ defined as follows. $W(< n_0, n_1 >) = 1, W(< n_1, S_i >= k - 1), W(< S_i, W_i >) = -\Sigma w(V_i)$ (ie the negative of the sum of all the edges incident on $V_i \in V$), $W(< W_k, < i, j >>) = w(V_i, V_j)$ if $< i, j > \in E$, and $W(< W_k, < i, j >>) = 0$ if $< i, j > \notin E$. f will also define an integer, $l = 0$. f will also define a set $R \subseteq N = \{n_0, n_1\}$
-Suppose there is a polynomial time algorithm for Steiner Tree, that returns solution S, a subtree

of H that contains each vertex in R, with weight $\leq l$.
-$h(S)$ will take in solution S, a subtree of H that represents a Steiner Tree. $h(S)$ will loop through each edge of the solution, and for each edge of negative weight, will find the corresponding V_i that is represented by the edge of negative weight labeled $< W_i, S_i >$. It will then add V_i to the solution set, R. R will then be returned.
-No solution implies no solution (if we can find no solution for steiner tree).

Theorem 4.20.

Proof. $\Rightarrow$
$\Leftarrow$ $\square$

4.19 Exact Cover $\leq_P$ Satisfiability

Algorithm for Exact Cover

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Satisfiability}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Exact Cover
Input: A family $\{S_j\}$ of subsets of the set $\{u_1, u_2, \dots u_t\}$
Property: there is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that $\{T_h\}$ are disjoint and $\cup T_h = \cup S_j = \{u_1, u_2, \dots u_t\}$

Satisfiability
Input: Clauses $C_1, C_2, \dots C_p$ with literals $x_1, x_2, \dots x_n, \overline{x_1}, \overline{x_2}, \dots \overline{x_n}$.
Property:
$\exists S \subseteq \{x_1, x_2, \dots x_n, \overline{x_1}, \overline{x_2}, \dots \overline{x_n}\}$ such that if $x_i \in S$ then $\overline{x_i} \notin S$, and if $\overline{x_i} \in S$ then $x_i \notin S$ and $S \cap C_j \neq \emptyset$, $\forall j \in \{1, 2, \dots p\}$

Details -Let $I = \{S_j\}$, a family of subsets with elements u_i. Then f will define Clauses $C_1, C_2, \dots C_k$ and literals $x_1, x_2, \dots x_j$. Each element, $u_i \in \cup S_j$ will have a corresponding clause C_i. If $u_i \in S_j$, then $x_j \in C_i$. Further, if C_i has 2 or more members, then there will be Clauses added containing the negation of every combination of 2 literals in C_i. For example, if $C_i = (1 \cup 2 \cup 3)$, the Clauses added will be $C_l = (\bar{1} \cup \bar{2}), C_m = (\bar{1} \cup \bar{3}), C_n = (\bar{2} \cup \bar{3})$.
-Suppose there is a polynomial time algorithm for Satisfiability, that returns solution S, a satisfying assignment for the literals x_j in Clauses C_k.
-$h(S)$ will take in Solution S, a satisfying assignment for the literals x_j in Clauses C_k. Then h will loop through S, and if x_j is set to be true, it will add the corresponding set S_j to solution set R. R will then be returned.
-No solution implies no solution.

Theorem 4.21. *$\{S_j\}$ has an exact cover iff Clauses $C_1, C_2, \dots C_k$ with literals $x_1, x_2, \dots x_j$, constructed as defined above, has a satisfying assignment.*

Proof. $\Rightarrow$ Suppose we have an exact cover on sets $\{S_j\}$. That is, there is some family of subsets, T_h, that are disjoint such that $\cup T_h = \cup S_j$. Based on our construction, each element in $\cup S_j$ has a clause with the literals of the clause being the sets that the elements u_i appears in. Further, for each clause of size 2 or more are clauses containing each combination of 2 literals in the original

clause that contains the negation of these literals. Thus, if each set that is part of the exact cover is chosen to be true, then each clause representing an element will be satisfied. Further, if the literals representing the remaining sets are chosen to be false, the negation clauses will be satisfied, as the remaining clauses only have literals that represent sets not chosen in the exact cover left, and since they are negated, they must be set to false. Thus, there is a satisfying assignment for the Clauses.
$\Leftarrow$ Suppose we have a satisfying assignment for Clauses $C_1, C_2, \dots C_p$. Based on our construction, the clauses represent elements, u_i, of $\cup S_j$ and the literals represent sets in S_j. Each clause has literals representing the sets that the corresponding clauses element appears in. Further, each clause that has 2 or more literals has corresponding clauses containing the negation of every corresponding combination of 2 literals from the original clause. Since each element has a clause, and each of these clauses must be satisfied, we know that each element has at least one set that can be chosen that contains the element. Further, in order for the clauses with negated literals to be satisfied, each other literal that is not chosen must be set to false, as they will all appear in a set with the chosen literal. For example, if the clause representing an element is, $(A \cup B \cup C)$, the negated clauses will be $(\bar{A} \cup \bar{B}), (\bar{A} \cup \bar{C}), (\bar{B} \cup \bar{C})$. If A is set to be true, both B and C must be set to false to satisfy C. However, we already know that we have a satisfying assignment, which means each element must only have exactly one set that can be chosen such that $\cup S_j = \cup T_h$ and T_h is disjoint. By definition, that's an exact cover on S_j. $\square$

4.20 Hitting Set $\leq_P$ Satisfiability

Algorithm for Hitting Set

$$\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Satisfiability}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)$$

Steiner Tree
Input: A graph $G = (N, A)$, $R \subseteq N$, a weighting function $w : A \to \mathbb{Z}$ and a positive integer k
Property: G has a subtree of weight $\leq k$ containing the set of vertices in R

Satisfiability
Input: Clauses $C_1, C_2, \dots C_p$ with literals $x_1, x_2, \dots x_n, \overline{x_1}, \overline{x_2}, \dots \overline{x_n}$.
Property:
$\exists S \subseteq \{x_1, x_2, \dots x_n, \overline{x_1}, \overline{x_2}, \dots \overline{x_n}\}$ such that if $x_i \in S$ then $\overline{x_i} \notin S$, and if $\overline{x_i} \in S$ then $x_i \notin S$ and $S \cap C_j \neq \emptyset$, $\forall j \in \{1, 2, \dots p\}$

Details -Let $I = \{S_j\}$, a family of subsets with elements u_i. Then f will define Clauses $C_1, C_2, \dots C_k$ and literals $x_1, x_2, \dots x_j$. Each set, S_j, will have a corresponding clause C_i. If $u_i \in S_j$, then $x_i \in C_j$. Further, if C_j has 2 or more members, then there will be Clauses added containing the negation of every combination of 2 literals in C_j. For example, if $C_j = (1 \cup 2 \cup 3)$, the Clauses added will be $C_l = (\bar{1} \cup \bar{2}), C_m = (\bar{1} \cup \bar{3}), C_n = (\bar{2} \cup \bar{3})$.
-Suppose there is a polynomial time algorithm for Satisfiability, that returns solution S, a satisfying assignment for the literals x_j in Clauses C_k.
-$h(S)$ will take in Solution S, a satisfying assignment for the literals x_j in Clauses C_k. Then h will loop through S, and if x_j is set to be true, it will add j to solution set R. R will then be returned.
-No solution implies no solution.

Theorem 4.22. *$\{S_j\}$ has a hitting set iff Clauses $C_1, C_2, \ldots C_k$ with literals $x_1, x_2, \ldots x_i$, constructed as defined above, has a satisfying assignment.*

Proof. $\Rightarrow$ Suppose we have a hitting set on sets $\{S_j\}$. That is, there is some set W, such that $|W \cap S_j| = 1$ for all $S_j \in \{S_j\}$. Based on our construction, each set in $\{S_j\}$ has a clause with the literals of the clause being the elements of S_j. Further, for each clause of size 2 or more are clauses containing each combination of 2 literals in the original clause that contains the negation of these literals. Thus, if each literal representing an element that is part of W is set to be true, then each clause representing a set in $\{S_j\}$ will be satisfied. Further, if the literals representing the remaining elements $\notin W$ are set to be false, the negation clauses will be satisfied, as the remaining clauses only have literals that represent elements $\notin W$, and since they are negated, they must be set to false. Thus, there is a satisfying assignment for the Clauses.
$\Leftarrow$ Suppose we have a satisfying assignment for Clauses $C_1, C_2, \ldots C_p$. Based on our construction, the clauses represent sets, $S_j \in \{S_j\}$ and the literals represent the elements of the corresponding set. Further, each clause that has 2 or more literals has corresponding clauses containing the negation of every corrresponding combination of 2 literals from the original clause. Since each set has a clause, and each of these clauses must be satisfied, we know that each set has at least one element that can be chosen that is part of W. Further, in order for the clauses with negated literals to be satisfied, each other literal that is not chosen to be true to satisfy the set clauses must be set to false, as they will all appear in a set with the chosen literal. For example, if the clause representing an element is, $(A \cup B \cup C)$, the negated clauses will be $(\bar{A} \cup \bar{B}), (\bar{A} \cup \bar{C}), (\bar{B} \cup \bar{C})$. If A is set to be true, both B and C must be set to false to satisfy each of the clauses. However, we already know that we have a satisfying assignment, which means that each set must only have exactly one element that can be chosen to be in W such that $|W \cap S_j| = 1$. By definition, that's a hitting set on S_j. $\square$

4.21 Satisfiability $\leq_p$ Knapsack

Algorithm for Satisfiability

$$\boxed{\mathrm{I} \mapsto \mathrm{f(I)} \rightarrow \boxed{\text{Algorithm for Knapsack}} \;\frac{\rightarrow x \mapsto h(x)}{\rightarrow N.S. \mapsto N.S.}}$$

Satisfiability
Input: Clauses $C_1, C_2, \ldots C_p$ with literals $x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}$.
Property:
$\exists S \subseteq \{x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}\}$ such that if $x_i \in S$ then $\overline{x_i} \notin S$, and if $\overline{x_i} \in S$ then $x_i \notin S$ and $S \cap C_j \neq \emptyset$, $\forall j \in \{1, 2, \ldots p\}$

Knapsack
Input: $(a_1, a_2, \ldots a_r, b) \in \mathbb{Z}^{r+1}$
Property: $\sum a_j x_j = b$ has a 0-1 solution

Details Given an instance I $= (C_1, C_2, ..., C_P)$ of Satisfiability, we form an instance f(I) of knapsack

as follows: Let m be (the maximum number of literals that appear in a single C_i) + 1. Assign to each C_i the label m^i. Each x_j for knapsack will correspond to the variables in Satisfiability in the obvious way. We define a_j to be:

$$\left(\sum_{C_i | x_j \in C_i} m^i\right) - \left(\sum_{C_i | \bar{x}_j \in C_i} m^i\right)$$

We let

$$b = \left(m^0 + m^1 + \cdots + m^P\right) - \left(\sum_{\bar{x}_j} \sum_{C_i | \bar{x}_j \in C_i} m^i\right)$$

Additionally, for each C_i, we add (m - 1) DONT NEED THAT MANY filler variables $x_{filler-i-h}$ with associated $a_{filler-i-h} = -m^i$.
We map a solution, S, via h(S) using the assignment of x_i to select true literals.

Theorem 4.23. *There is a solution to an instance I of Satisfiability* $\iff$ *there is a solution to f(I) in Knapsack.*

Proof. Label each clause with a distinct power of m by $C_i \to m^i$. For every literal, σ, map σ to the sum of powers of m for each clause σ appears in. We use this to require an exact satisfying argument for Satisfiability that has no conflicting literals as:

$$\begin{aligned}&(1 - x_1)(\sum_{C_i | \bar{x}_1 \in C_i} m^i) + x_1(\sum_{C_i | x_1 \in C_i} m^i) + \cdots + (1 - x_r)(\sum_{C_i | \bar{x}_r \in C_i} m^i) + x_r(\sum_{C_i | x_r \in C_i} m^i)\\ &= \left(m^0 + m^1 + \cdots + m^P\right)\end{aligned}$$

Note that this holds exactly when

$$\begin{aligned}&x_1(\sum_{C_i | x_1 \in C_i} m^i - \sum_{C_i | \bar{x}_1 \in C_i} m^i) + \cdots + x_r(\sum_{C_i | x_r \in C_i} m^i - \sum_{C_i | \bar{x}_r \in C_i} m^i)\\ &= m_0 + \cdots + m^P - \left(\sum_{\bar{x}_j} \sum_{C_i | \bar{x}_j \in C_i} m^i\right)\end{aligned}$$

Which shows that an exact solution for Satisfiability occurs exactly when the second equality holds. Because we seek any solution for Satisfiability, we relax the constraints to allow for subtracting up to (m - 1) copies of m^i for each C_i on the left side of the first equation. Because of our choice of base, we do not run the risk of overlapping by subtracting too much, and we retain the requirement that every clause is satisfied. We have added the possibility for a clause to be over-satisfied. Readjusting the second equation to compensate for this yields the formula we provide for knapsack. □

4.22 Satisfiability $\leq_p$ Feedback Node Set

Algorithm for Satisfiability

$$\boxed{\text{I} \mapsto \text{f(I)} \to \boxed{\text{Algorithm for Feedback Node Set}} \begin{array}{c} \to x \mapsto h(x) \\ \hline \to N.S. \mapsto N.S. \end{array}}$$

Satisfiability
Input: Clauses $C_1, C_2, \ldots C_p$ with literals $x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}$.
Property:
$\exists S \subseteq \{x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}\}$ such that if $x_i \in S$ then $\overline{x_i} \notin S$, and if $\overline{x_i} \in S$ then $x_i \notin S$ and $S \cap C_j \neq \emptyset$, $\forall j \in \{1, 2, \ldots p\}$

Feedback Node Set
Input: A directed graph $H = (V, E)$ and a positive integer k
Property: There exists $R \subseteq V$ such that every directed cycle of H contains a vertex in R and $|R| \leq k$

Details First verify no clauses contain conflicting literals. Now, given an instance I = $(C_1, \cdots, C_P)$ of Satisfiability, we construct an instance f(I) of feedback node set as follows: For a fixed C_j, let ϕ_J be a simple cycle on all of the literals in C_j. ADD NOTATION EXPLANATION

$$\begin{aligned}
N &= \{x_1, \cdots, x_r, \bar{x}_1, \cdots, \bar{x}_r\} \\
E &= \{< x_i, \bar{x}_i >\} \\
&\quad \cup \{< \bar{x}_i, x_i >\} \\
&\quad \cup \{< \phi_J(\sigma_1), \phi_J(\sigma_2) >, < \phi_J(\sigma_2), \phi_J(\sigma_3) >, \\
&\qquad \cdots, < \phi_J(\sigma_{n-1}), \phi_J(\sigma_n) >, < \phi_J(\sigma_n), \phi_J(\sigma_1) > \mid \sigma_i \text{ are all of the literals in } C_j\} \\
k &= |\{x_i\}|
\end{aligned}$$

Additionally, for any clauses with a single literal σ, add an auxillary node σ_0 and the edges $\{< \sigma, \sigma_0 >, < \sigma_0, \sigma >\}$.
Given a solution, S for Feedback Node Set, let h(S) be the set of literals in the image of S under the forgetful map (that is, $h(\sigma_0) = \sigma$ and otherwise $h(\sigma) = \sigma$).

Theorem 4.24. *There is a solution for an instance I of Satisfiability $\iff$ there is a solution for f(I) in Feedback Node Set.*

Proof. $\Rightarrow$ Let $C_1, \cdots, C_P$ be an instance of Satisfiability with solution S. Construct an instance of feedback node set as above. Let R = $\{\bar{x}_i \in S\} \cup \{x_i \mid \bar{x}_i \notin S\}$. Note first that S $\subseteq R$. Furthermore, we observe that $|R| = |\{x_i\}| = k$. We now verify that for every cycle in our graph, there is some $n \in R$ contained in it. For every cycle between $x_i, \bar{x}_i$ we have a node in R from construction. We consider larger cycles, of which there are two types. True clause cycles, and incidental cycles. For each true clause cycle, of course one of the satisfying literals is already included in R. Incidental cycles are formed when edges between nodes allow for a cycle not representing a set of literals that appear in a clause together, and not between complementary literals. For such cycles, each edge represents a pair of literals which appear in some clause together. Assume for a contradiction that none of the literals in an incidental cycle I are in S. In that case, we can modify our instance of Satisfiability to include the clause $\bigwedge_{\sigma \in I} \bar{\sigma}$ without consequence. As a matter of fact, this is not a contradiction but a valid counterexample. Consider $(a \vee b \vee c \vee d) \wedge (a \vee b \vee d) \wedge (b \vee c \vee d) \wedge (a \vee c \vee d) \wedge (\neg a) \wedge (\neg b) \wedge (\neg c)$, where an incidental cycle may form between a, b, and c and have no satisfying arguments possible to use within the cycle.

□

4.23 Undirected Hamiltonian Circuit $\leq_p$ Hitting Set

Algorithm for Undirected Hamiltonian Circuit

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Hitting Set}} \begin{array}{c} \rightarrow x \mapsto h(x) \\ \hline \rightarrow N.S. \mapsto N.S. \end{array}}$$

Undirected Hamiltonian Circuit
Input: A graph G
Property: G has a cycle which includes each vertex exactly once.

Hitting Set
Input: A family $\{U_i\}$ of subsets of $\{s_1, s_2, \dots s_r\}$
Property: there is a set W such that for each i, $|W \cap U_i| = 1$

Details Given an instance I = (N, A) of Undirected Hamiltonian Circuit, we form instance f(I) of hitting set as follows:

$$\begin{aligned} S_J = \{\{\{v_i, u_{i+1}\} \mid u \in N, u \neq v\} \mid v \in N\} \mid i \in \{1, ..., |N|\} \;\; (1) \\ \cup \{\{\{v_i, u_{i+1}\} \mid u \in N, i \in \{1, \cdots, |N|\}\} \mid v \in N\} \;\; (2) \\ \cup \{\{\{u_i, v_{i+1(mod|N|)}\} \mid \{u, v\} \in A\} \mid i \in \{1, \cdots, |N|\}\} \;\; (3) \end{aligned}$$

Given a solution, S = $\{\{v_1, v_2\}, \{v_2, v_3\}, \cdots, \{v_{|N|}, v_1\}\}$ for hitting set, we let h(S) be the sequence of v_i provided in our solution.

Theorem 4.25. *There is a solution for I in Undirected Hamiltonian Circuit $\iff$ there is a solution for f(I) in hitting set.*

Proof. We use the fact that a graph (N, A) has an Undirected Hamiltonian Circuit $\iff$ there is a sequence, $n_1 n_2 \cdots n_{|N|}$ so that each n_i is distinct, and for each $n_i n_{i+1(mod|N|)}$ the vertices are adjacent. We proceed to verify our construction.

Our first type of set requires that for each $i \in \{1, \cdots, |N|\}$ exactly one vertex is chosen for that i. The second type of set requires that for each vertex, exactly one place is selected. This gives us a simple permutation of our nodes. The third requirement verifies that each level selected as $i \rightarrow i + 1(mod|N|)$ corresponds to an edge in our graph. This restriction forces the selected permutation to exist $\iff$ there is an Undirected Hamiltonian Circuit in our graph from our previously established criterion. □

5 Conclusions

I haven't written this section, or future work yet because what we say will largely depend on if we end up with a cycle or not. I am happy to write something up when we find a cycle/give up.

6 Future Work

7 Appendix

7.1 Other Edges in the Cycle

7.1.1 Chromatic Number $\propto$ Clique Cover

Algorithm for Chromatic Number

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Clique Cover}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Chromatic Number
Input: A graph $G = (N, E)$ and a positive integer k
Property: there is a function $\phi : N \to \mathbb{Z}_k$ such that if u and v are adjacent then $\phi(u) \neq \phi(v)$

Clique Cover
Input: A graph $G' = (N', E')$ and a positive integer ℓ
Property: N' is the union of ℓ or fewer cliques

Details -Let $I = (G = (V, E), k)$ where G is a graph and k is a positive integer. Then f will create a new Graph, G', equal to the complement of G. f will also define an integer $l = k$.
-Suppose there is a polynomial time algorithm for Clique Cover, that returns solution S, a set of cliques representing a Clique Cover of size l on G'.
-$h(S)$ will take in a set of cliques, S, and define a function, ϕ. For each clique in S, $h(S)$ will map ϕ to the same element for each node in the same clique. ϕ will map each clique to a different result. ϕ will be returned.
-No solution implies no solution.

Theorem 7.1. *Graph $G = (V, E)$ has a chromatic number, k, iff its complement, G', has a clique cover of size $l = k$.*

Proof. $\Rightarrow$ Suppose G has a chromatic number, k. That is, there is a function, ϕ, that colors the vertices of G with k colors such that no two adjacent vertices have the same color. Thus, no two vertices who share the same color can have any edges between them on G. Now, take G', the complement of G, which includes every edge that wasn't in G, and doesn't include any edge that was in G. In this graph, vertices of the same color now must have an edge connecting them, as there were no edges between vertices of the same color in G. Thus, there is now a clique between each group of vertices with the same color on G'. Since there are k colors on G, G' has k cliques, thus G' has a clique cover of size $k = l$.
$\Leftarrow$ Suppose G', the complement of G, has a clique cover of size k. By definition, a clique contains edges between every member of the clique. In addition, even though a vertex, $v \in V$ can be in multiple cliques on G', each vertex can only be assigned to one clique in order for there to be a clique cover on G'. Therefore, v can be assigned to either of the cliques it is a part of, as long as there is a still a clique cover on G' of size k. Since G' is the complement of G, none of the vertices that formed cliques on G' will have any edges between each other on G. Thus, if the vertices that formed the cliques of the clique cover on G' are given the same color by ϕ on G, then no adjacent vertices will have the same color. Thus, G has a coloring of size k. $\square$

7.1.2 Knapsack $\propto$ Partition

Algorithm for Knapsack

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Partition}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Knapsack
Input: $(a_1, a_2, \ldots a_r, b) \in \mathbb{Z}^{r+1}$
Property: $\sum a_j x_j = b$ has a 0-1 solution

Partition
Input: $(c_1, c_2, \ldots c_s) \in \mathbb{Z}^s$
Property: there is an $I \subseteq \{1, 2, \ldots s\}$ such that $\sum_{h \in I} c_h = \sum_{h \notin I} c_h$

Details -Let $I = (a_1, a_2, \ldots a_r, b)$, where a is a list of integers and b is an integer. Then f will define a set, c, with size $s = r + 2$. $c_i = a_i$, with $1 \leq i \leq r$. $c_{r+1} = b + 1$, and $c_{r+2} = (\Sigma_{i=1}^r a_i) + 1 - b$.
-Suppose there is a polynomial time algorithm for Partition that returns solution S, a set such that $S \subseteq \{1, 2, \ldots s\}$ and $\Sigma_{h \in S} c_h = \Sigma_{h \notin S} c_h$.
-$h(S)$ will take in set S, a solution to Partition. It will loop through S, adding elements that represent elements of a to a new set R. Any vertices that were added to the set S as part of the construction and that do not represent elements of C are not included in R. $h(S)$ will then define a new set, x, of size r. For $1 \leq i \leq r$, $x_i = 1$ if $a_i \in R$, $x_i = 0$ otherwise. x will then be returned.
-No solution implies no solution.

Theorem 7.2. *$a = (a_1, a_2, \ldots a_r)$ and b have a 0-1 solution for $\Sigma a_j x_j = b$ iff set c, of size s, both constructed as defined above, have a set $I \subseteq \{1, 2, \ldots s\}$ such that $\Sigma_{h \in I} c_h = \Sigma_{h \notin I} c_h$.*

Proof. $\Rightarrow$ Suppose that $a = (a_1, a_2, \ldots a_r)$ and integer b have a set x of size r that has a 0-1 solution such that $\Sigma a_r x_j = b$. That is, there is some subset, S, $S \subseteq a$ such that $\Sigma_{h \in S} a_h = b$. Based on our construction, c is equivalent to a for all i such that $1 \leq i \leq r$. Thus, that same subset S, can be used, with $\Sigma_{h \in S} a_h = b$ and $\Sigma_{h \notin S} a_h = \Sigma_{i=1}^r a_i - b$. However, in c there are two additional elements, $I_{r+1} = b + 1$ and $I_{r+2} = (\Sigma_{i=1}^r a_i) + 1 - b$. If I_{r+1} is added to $h \notin S$ and I_{r+2} is added to $h \in S$, we now have $\Sigma_{h \in S} c_h = b + \Sigma_{i=1}^r a_i + 1 - b$ and $\Sigma_{h \notin S} c_h = \Sigma_{i=1}^r a_i - b + b + 1$. Simplify, and we have $\Sigma_{h \in S} c_h = \Sigma_{i=1}^r a_i + 1 = \Sigma_{h \notin S} c_h$. By definition, c has a partition, I, such that $\Sigma_{h \in I} c_h = \Sigma_{h \notin I} c_h$.
$\Leftarrow$ Suppose that set c, of size s, as constructed above, has a set, $I \subseteq \{1, 2, \ldots s\}$ such that $\Sigma_{h \in I} c_h = \Sigma_{h \notin I} c_h$. Based on our construction, $s = r + 2$, and for $1 \leq i \leq r$, $c_i = a_i$. Additionally $c_{r+1} = b + 1$ and $c_{r+2} = (\Sigma_{i=1}^r a_i) + 1 - b$. The remaining elements have value $\Sigma_{i=1}^r a_i$. If c_{r+1} and c_{r+2} are in the same half of the partition, $c_{r+1} + c_{r+2} = \Sigma_{i=1}^r a_i + 1 - b + b + 1 = \Sigma_{i=1}^r a_i + 2 > \Sigma_{i=1}^r a_i$. Thus, there is no possible partition if c_{r+1} and c_{r+2} are both included in the set I, or if both are not included in the set I, so exactly one must be included. (i.e c_{r+1} and c_{r+2} must be in different halves of the partition). However, a partition was supposed to exist, so we have
$b + 1 + \Sigma_{h \notin S} a_h = (\Sigma_{i=1}^r a_i) + 1 - b + \Sigma_{h \in S} a_h$
$2b + \Sigma_{h \notin S} a_h = \Sigma_{i=1}^r a_i + \Sigma_{h \in S} a_h$
$2b + \Sigma_{h \notin S} a_h + \Sigma_{h \in S} a_h = \Sigma_{i=1}^r a_i + 2\Sigma_{h \in S} a_h$
By definition, $\Sigma_{h \notin S} a_h + \Sigma_{h \in S} a_h = \Sigma_{i=1}^r a_i$. Thus,
$2b + \Sigma_{i=1}^r a_i = \Sigma_{i=1}^r a_i + 2\Sigma_{h \in S} a_h$
$2b = 2\Sigma_{h \in S} a_h$

$b = \Sigma_{h \in S} a_h$
Thus, if the elements $a_i \in a \cap a_i \in S$ are given a 1 value for $x_i \in x$, and the elements $a_i \in a \cap a_i \notin S$ are given a 0 value for $x_i \in x$, then x_j will be a 0-1 solution for $\Sigma a_r x_j = b$. □

7.1.3 Clique $\propto$ Node Cover

Algorithm for Clique

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Node Cover}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Clique
Input: A graph G and a positive integer k
Property: G has a clique of size k

Node Cover (Vertex Cover)
Input: A graph $G' = (N', A')$ and a positive integer ℓ
Property: $\exists R \subseteq N'$ such that $|R| \leq \ell$ and every arc in incident to a vertex in R.

Details
-Let $I = (G = (V, E), k)$, where G is a graph and k is a positive integer. Then f will define a graph G' that is equal to the complement of G. (i.e. all of the nodes in G' are the same as in G, edges that did not appear in G appear in G', and edges that did appear in G do not appear in G'). f will also define a positive integer, l, that is equal to $|V| - k$.
-Suppose that there is a polynomial time algorithm for Node Cover, that returns solution S, which is a set of Nodes, S, that represents a Node Cover of G'.
-$h(s)$ will take in a set, S, of Nodes representing a Node Cover, and use the original set of Nodes, V, from the original graph G. h will define a new set of Nodes, $N = V - S$, and return this. (i.e. the set of Nodes that are in V, but that are not in S.
-No solution implies no solution

Theorem 7.3. *$G = (V, E)$ has a clique of size k iff there is a node cover of size $|V| - k$ on G'.*

Proof. $\Rightarrow$ Suppose $G = (V, E)$ has a clique of size k. Thus, G' will not have any edges between the members of the clique $(v_1, v_2, \ldots v_k)$. Each vertex in the clique $(v_1, v_2, \ldots v_k)$ will be accessible in G' by any vertices they are not connected to in G. (If G is completely connected, there will be no edges in G', so the empty set will be a node cover) Therefore, if the vertices not in the clique are chosen to be in the Node Cover, all edges will be covered and the size will be $|V| - k$.
$\Leftarrow$ Suppose there is a node cover, S, on G' of size $l = |V| - k$. Since S is a node cover, all edges can be accessed by the vertices in S. Thus, no edges in the set in the set $V - S$ can be connected to each other in G', or S wouldn't be a valid node cover. Thus, the vertices in $V - S$ form a clique of size $|V| - |S|$ on G. $|S|$ is defined to be l, or $|V| - k$, so the size of the clique is
$|V| - (|V| - k) = |V| - |V| + k, = k$.
Thus, if there is a Node Cover, S, on G' of size $|V| - k$, then there is a clique on G of size k. □

7.1.4 Exact Cover $\propto$ Hitting Set

Algorithm for Exact Cover

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Hitting Set}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Exact Cover
Input: A family $\{S_j\}$ of subsets of the set $\{u_1, u_2, \ldots u_t\}$
Property: there is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that $\{T_h\}$ are disjoint and $\cup T_h = \cup S_j = \{u_1, u_2, \ldots u_t\}$

Hitting Set
Input: A family $\{U_i\}$ of subsets of $\{s_1, s_2, \ldots s_r\}$
Property: there is a set W such that for each i, $|W \cap U_i| = 1$

Details -Let $I = \{S_j\}$, a family of subsets of a set $\{u_i, i = 1, 2, \ldots t\}$. Then f will define a family of subsets, $\{U_i\}$, such that $s_j \in U_i$ iff $u_i \in S_j$. (i.e. $\{U_i\}$ will contain i sets, $i = |\cup S_j|$, with elements representing the sets that each element represented by U_i are in in $\{S_j\}$).
-Suppose there is a polynomial time algorithm for Hitting Set, that returns Solution S, a set such that, for each $S_i \in \{U_i\}$, $|W \cap U_i| = 1$.
-$h(S)$ will take in a set of integers, S, representing a Hitting Set on $\{U_i\}$. This same set can be returned, as the same integers represent the indices on $\{S_j\}$ that form an exact cover.
-No solution implies no solution

Theorem 7.4. *$\{S_j\}$ has an exact cover iff $\{U_i\}$, as constructed above, has a hitting set.*

Proof. $\Rightarrow$ Suppose $\{S_j\}$ has an exact cover. By our construction, each element in $\cup S_j$ gets a set in $\{U_i\}$, with elements of U_i being the sets in $\{S_j\}$ that the element represented by U_i appears in. Suppose, for the sake of contradiction, that U_i does not have a hitting set. That is, for every $W \in U$ there is a U_i where $|W \cap U_i| \neq 1$. Since each $U_i \in \{U_i\}$ represents elements, this means that some element, u_i, appears in multiple or zero sets, S_j, and there is no group of sets, $S_j \in \{U_i\}$, that can be chosen in $\{S_j\}$ such that this element does not appear exactly once. Thus, there is no exact cover in $\{S_j\}$, which contradicts what was previously supposed.
$\Leftarrow$ Suppose $\{U_i\}$ has a hitting set. By our construction, each set in $\{U_i\}$ represents an element in $\cup S_j$, with elements of U_i being the sets that the elements represented by it appears in in $\{S_j\}$. There exists some W, such that $|W \cap U_i| = 1$ for all $U_i \in \{U_i\}$. Since the sets in $\{U_i\}$ represent elements in $\{S_j\}$, and each element in U_i represents the sets that element is in in $\{S_j\}$, $|W \cap U_i| = 1$ implies that the subset of sets represented by the elements of W can be chosen such that every element in $\{S_j\}$ appears exactly once, which by definition is an exact cover for $\{S_j\}$. □

7.1.5 Clique $\propto$ Set Packing

Algorithm for Clique

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Set Packing}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Clique
Input: A graph G and a positive integer k
Property: G has a clique of size k

Set Packing
Input: A family of sets $\{S_j\}$ and a positive integer ℓ
Property: $\{S_j\}$ has ℓ mutually exclusive sets.

Details -Let $I = (G = (V, E), k)$, where G is a graph and k is a positive integer, where $V = \{1, 2, \dots, k\}$. Then f will define $\{S_j\}$, a family of n sets, with elements...
$S_i = \{\{i, j\} \mid \{i, j\} \notin E\}, 1 \leq i \leq n$ (i.e. each node gets a set, with elements representing edges that aren't in G). f will also define an integer, l, with $l = k$.
-Suppose that there is a polynomial time algorithm for Set Packing that returns solution S, which is a family of sets representing mutually exclusive sets of $\{S_j\}$.
-$h(s)$ will take the sets in S, and convert them back to their respective nodes from G. Each node will be added to the subgraph, H, of G. For example, if $S = \{S_1, S_2, S_4\}, h(s)$ will return the subgraph $H = (V, E)$ with $V = \{1, 2, 4\}$.
-No Solution implies no solution.

Theorem 7.5. *$G = (V, E)$ has a clique of size n iff there are k mutually exclusive sets in $\{S_j\}$ as defined above.*

Proof. $\Rightarrow$ Suppose G has a clique of size k. Let $\{v_1, v_2, \dots v_k\}$ be the vertices in this clique. If $i \neq j$ and $v_i, v_k \in \{v_1, v_2, \dots v_k\}$, then $S_i \cap S_j = 0$, or there would be two members of the clique that do not have an edge between them, which is impossible. Thus, $S_i \cap S_j = 0, i \neq k, v_i, v_k \in \{v_1, v_2, \dots v_k\}$. By definition, this means there is a set packing of size k on $\{S_j\}$.
$\Leftarrow$ Suppose there are k mutually exclusive sets in $\{S_j\}$. WLOG, let $S_1, S_2, \dots S_k$, be the sets that form the exact cover in $\{S_j\}$. Let G be the graph constructed as directed above. Suppose, for contradiction, there is no clique of size k in G, between vertexes 1 and k. Therefore, for vertices i in $1 \leq i \leq k$, there exists a vertex j and a vertex i with no edge between them. Therefore, (i, j) will be added to S_i and S_j. Thus, S_i and S_j are not mutually exlusive, as both contain the edge (i, j). This contradicts what was previously supposed. □

7.1.6 Satisfiability $\propto$ Satisfiability With At Most 3 Literals Per Clause

Algorithm for Satisfiability

$$\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for Satisfiability With At Most 3 Literals Per Clause}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)$$

Satisfiability
Input: Clauses $C_1, C_2, \dots C_p$ with literals $x_1, x_2, \dots x_n, \overline{x_1}, \overline{x_2}, \dots \overline{x_n}$.
Property: $C_1 \wedge C_2 \wedge \cdots \wedge C_p$ is *satisfiable*: if $\exists S \subseteq \{x_1, x_2, \dots x_n, \overline{x_1}, \overline{x_2}, \dots \overline{x_n}\}$ such that if $x_i \in S$ then $\overline{x_i} \notin S$, and if $\overline{x_i} \in S$ then $x_i \notin S$ and $S \cap C_j \neq \emptyset$, $\forall j \in \{1, 2, \dots p\}$

Satisfiability With At Most Three Literals Per Clause
Input: Clauses $D_1, D_2, \dots D_r$, each consisting of at most 3 literals from $\{u_1, u_2, \dots u_m, \overline{u_1}, \overline{u_2}, \dots \overline{u_m}\}$
Property: $\{D_1, D_2, \dots D_r\}$ is satisfiable

Details -Let $I = ((C_1, C_2, \dots C_p), (x_1, x_2, \dots x_n))$ where C are clauses and x are literals. For every clause, C_j in I with more than 3 literals, we say $C_j = \sigma_1 \cup \sigma_2 \cup \dots \sigma_m$. Replace each C_j with

$(\sigma_1 \cup \sigma_2 \cup u_1)(\sigma_3 \cup \dots \sigma_m \cup \bar{u_1})(\bar{\sigma_3} \cup u_1) \dots (\bar{\sigma_m} \cup u_1)$
-Suppose there is a polynomial time algorithm for Satisfiability With at Most 3 Literals Per Clause that returns solution S, a satisfying assignment for $f(I)$.
-$h(S)$ will take in the satisfying assignment for $f(I)$, remove all of the variables labeled "u", and return what is remaining in the satisfying assignment.
-No Solution implies no solution.

Theorem 7.6. *Clauses $C_1, C_2, \dots C_p$ are satisfiable with literals $x_1, x_2, \dots x_n$ iff Clauses $D_1, D_2, \dots D_k$ with literals $x_1, x_2, \dots x_n, u_1, \dots u_t$, constructed as directed above, are satisfiable.*

Proof. $\Rightarrow$ Suppose Clauses $C_1, C_2, \dots C_p$ are satisfiable with literals $x_1, x_2, \dots x_n$. By induction on n, the number of clauses with more than 3 literals, we can show that our instance of Satisfiability With at Most 3 Literals Per Clause will also be satisfied.
Base Case: n=1
Suppose Clauses $C_1, C_2, \dots C_p$ are satisfiable and have exactly one clause with more than 3 literals. Since the clauses with 3 or less literals are kept the same in $f(I)$, they will still be satisfiable in $f(I)$ if the same literals are chosen. Let $C_j = (x_1, x_2, \dots x_m)$ be the clause, satisfied in I, with more than 3 literals. At least one of these literals must be satisfied for C_j to be true. Based on the construction, and WLOG, if x_1 and x_2 are false, then u_1 can be set to true to satisfy the clause. Then, since C_j is satisfiable, even though $\bar{u_1}$ is false in the next clause, the literal that satisfies C_j is guaranteed to be in it. Thus, the new construction will be satisfiable, and the same literals can be used to satisfy $f(I)$.
Inductive Hypothesis
Suppose I is satisfiable and has n clauses with more than 3 literals. Further suppose that the instance of Satisfiability With at Most 3 Literals Per Clause that is constructed from I is satisfiable.
Inductive Step
Suppose I is satisfiable and has $n+1$ clauses with more than 3 literals. In our construction of $SatisfiabilityWithatMost3LiteralsPerClause$, the n clauses with more than 3 literals from our inductive step will have the same construction, and will be satisfied by Satisfiability With at Most 3 Literals Per Clause, as they were satisfied in our Inductive Hypothesis. Let $C_j = (x_1, x_2, \dots x_m)$ be the remaining clause, satisfied in I, with more than 3 literals. At least one of these literals must be satisfied for C_j to be true. Based on the construction, and WLOG, if x_1 and x_2 are false, then u_1 can be set to true to satisfy the clause. Then, since C_j is satisfiable, even though $\bar{u_1}$ is false in the next clause, the literal that satisfies C_j is guaranteed to be in it. Thus, the new construction will be satisfiable, and the same literals can be used to satisfy $f(I)$. $\Leftarrow$ Suppose $f(I) = ((D_1, D_2, \dots D_k)$ with literals $(x_1, x_2, \dots x_n, u_1, \dots u_t))$ is satisfiable, and was constructed as directed above. Further suppose Clauses $D_1, D_2, \dots D_k$ were constructed based on the instructions above, as there is at least one clause C_j from I that had more than 3 literals. Thus, $D_1 = \{x_1, x_2, u_1\}$ and $D_2 = \{x_3, x_4, \bar{u_1}\}$. For $f(I)$ to be satisfiable, both $D1$ and $D2$ must be satisfiable. Since $u_1 \in D$ and $\bar{u_1} \in D_2$, u_1 can only be used to satisfy one of these clauses. Thus, one of x_1, x_2, x_3, x_4 must be true in order for D_1 and D_2 to be satisfied. Since D_1 and D_2 were constructed from the clause $C_j = \{x_1, x_2, x_3, x_4\}$, and one of these must be satisfied, C_j must be satisfied as with any other clauses with greater than 3 literals. Further, since any clauses in $f(I)$ with less than 4 literals per clause were kept identical from I, they will be satisfied as well. Thus, I is satisfiable if $f(I)$ is satisfiable and was constructed as directed above. □

7.1.7 Exact Cover $\propto$ 3-Dimensional Matching

Algorithm for Exact Cover

$$\boxed{\text{I} \mapsto \boxed{\text{f}} \mapsto f(I) \mapsto \boxed{\text{Algorithm for 3-D Matching}} \mapsto S \mapsto \boxed{\text{h}} \mapsto h(s)}$$

Exact Cover
Input: A family $\{S_j\}$ of subsets of the set $\{u_1, u_2, \ldots u_t\}$
Property: there is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that $\{T_h\}$ are disjoint and $\cup T_h = \cup S_j = \{u_1, u_2, \ldots u_t\}$

3-Dimensional Matching
Input: $U \subseteq T \times T \times T$ where T is a finite set
Property: there is a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of W agree in any coordinate

Details -Let $I = \{S_j\}$, a family of subsets. Then f will define a set $T = \{< i, j > | u_i \in S_j\}$. f will also define α, an arbitrary one-to-one function from $\{u_i\}$ into T. f will also define $\pi : T \mapsto T$, a permutation such that for each fixed j, $\{< i, j > | u_i \in S_j\}$ is a cycle of π. (i.e π creates a cycle between the elements u_i in a particular set in $\{S_j\}$. For example, if $S_2 = \{1, 2, 5\}$, π could create a cycle as follows: $(1, 2) \mapsto (2, 2) \mapsto (5, 2) \mapsto (1, 2)$). f will define a set, $U = \{\alpha(u_i), < i, j >, < i, j >> | < i, j > \in T\} \cup \{< \beta, \sigma, \pi(\sigma) > | \text{ for all } i, \beta \neq \alpha(u_i)\}$
-Suppose there is a polynomial time algorithm for 3-D Matching, that returns solution S, such that $|S| = |T|$, and such that the elements of S are mutually exclusive at each coordinate in the elements of S.
-$h(S)$ will take in solution S, a set representing a solution to 3-D Matching. Then h will create a set, R. h will loop through S, determining if each elements second and third coordinate pairs, $< i, j >$, and equal to each other. If they are equal, then h will add j to the set R if it is not already in R. R will be returned.
-No solution implies no solution.

Theorem 7.7. *$\{S_j\}$ has an exact cover iff sets T and U, constructed as defined above, have a 3-D Matching. That is, there is some subset $S \subseteq U$ with $|S| = |T|$, that is mutually exlusive between the 3 element coordinate pairs.*

Proof. $\Rightarrow$ Suppose $\{S_j\}$ has an exact cover, $\{T_h\}$. That is, there is some subset, $T_h \subseteq S_j$ such that $\cup T_h = \cup S_j$, and the sets in T_h are mutually exlusive. Suppose sets T and U are constructed as defined above, with $|T| = i$, with i being the number of elements in $\cup T_h$. In our construction of U, there are two types of elements, U_m represents elements that were constructed using α, U_c represents elements that were constructed with elements of T that were not mapped to by α, utilizing the permutation, π. Since $|T| = |S|$, each permutation of U_m and U_c must have 1 instance with a unique element in each position. For the elements in U_m, we take the element in U for each U_m that represents an element that is part of the exact cover. Since each element in S_j has a unique representation with its element value and set number, and each element u_i maps to exactly one element in T, the sets we chose in U_m must be mutually exclusive. Now, we are left with the elements in T that weren't mapped to in position 1, so if we take one element from each group in U_c, then $|S| = |T|$. Now, we just need to worry about positions 2 and 3. In positions 2 and 3, each set in

$\{S_j\}$ has a cycle of our permutation between the elements representing its members. Further, since it is a cycle, each element in T will be in position 2 and position 3 exactly once for every possible element in position 1 in U_c. Further, since the cycles are only between members of the same sets in S_j, there is no concern about choosing elements from T that were already chosen in U_m. Thus, if we start at the beginning of each cycle representing a set that not in the set cover, and move through the cycle with each new element in position 1 of U_c, then every possible combination of elements representing elements not in T_h will be covered. Thus, we have a 3-Dimensional Matching for T and U.
$\Leftarrow$ Suppose T and U have a 3-Dimensional Matching. That is, there is some subset $S \subseteq U$ such that $|S| = |T|$, and S is mutually exclusive over its coordinates of size 3. Further, assume T and U are constructed as defined above. Now, take U_m, representing the nodes that were constructed using α. Thus, since each node in $u_i \in \cup S_j$ maps to an element in T with α, each u_i must have successfully mapped to an element in T. Further, since U_m is constructed with $(<\alpha(u_i)>, <i, j>, <i, j>)$, where j is the set(s) i appears in in S_j, and since π, the permutation responsible for the elements in U_c, only creates cycles between elements of the same set in S_j, in order for T to be mutually exclusive there can be no elements in U_m chosen for the 3-Dimensional Matching representing the same set in S_j as an element in U_c that is chosen. Thus, since each element in U_i is represented by the elements chosen from U_m, and the sets represented by U_m do not overlap with the sets chosen from U_c, the sets represented by the elements in U_m that were chosen must form an exact cover on S_j. $\square$

7.1.8 Satisfiability $\leq_p$ Clique

Algorithm for Satisfiability

$$\boxed{\text{I} \mapsto f(I) \rightarrow \boxed{\text{Algorithm for Clique}} \rightarrow S \mapsto h(S)}$$

Satisfiability
Input: Clauses $C_1, C_2, \ldots C_p$ with literals $x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}$.
Property: $C_1 \wedge C_2 \wedge \cdots \wedge C_p$ is *satisfiable*: if $\exists S \subseteq \{x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}\}$ such that if $x_i \in S$ then $\overline{x_i} \notin S$, and if $\overline{x_i} \in S$ then $x_i \notin S$ and $S \cap C_j \neq \emptyset$, $\forall j \in \{1, 2, \ldots p\}$

Clique
Input: A graph G and a positive integer k
Property: G has a clique of size k

Details I: Clauses $C_1, C_2, \ldots C_p$ with literals $x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}$. **f:** Function f takes the input I and convert it into a graph G = (V, E) where V = $\{<\sigma$, i$>$ $|\sigma$ is a literal in $C_i\}$, E = $\{\{<<\sigma, i>, <\delta, i>>\} | \, i \neq j$ and $\sigma \neq \delta\}$ and $k = p$, the number of clauses. **Algorithm for Clique:** Suppose there is a polynomial time algorithm for the Clique problem which returns a sub-graph of G which is a k-clique. **S:** Solution S is the sub-graph of G which is a k-clique. **h(S):** function h takes S as an input and transform the vertices of the solution to the literals that they represent in G.

Theorem 7.8. *Clauses $C_1, C_2, \ldots C_p$ with literals $x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}$ are satisfiable if and only if there exists a graph $G = (V, E)$ with a clique of size k, as defined above.*

Proof. ($\Rightarrow$) Let us assume that $\exists S, S$ is a solution to a given instance of Satisfiability. Given for every clause in C there exists at least 1 literal which satisfies $C_i \Rightarrow x_i \cap C_i \neq \emptyset$. We then construct a graph G = (V, E) where $V = \{< x, i > | x$ is a literal in $C_i\}$ and $E = \{< x, i >, < y, j > | i \neq j$ and $x \neq \overline{y}\}$ and $k = p$. According to our construction, each vertex (which represents a literal) is connected to all the other vertexes representing the literals from other clauses, except the vertexes which represents the compliment of the literal and vertexes which represents literals of the same clause. Since vertexes are not connected with their compliments in the graph, it will give us a clique of size k.
($\Leftarrow$) Suppose G is a graph with K-clique constructed as directed above from the instance I of Satisfiability problem. In the graph constructed nodes which are in the same clause(group of nodes) and nodes which represent the compliment of the graph must appear in distinct cliques and thus each of the k clauses contain exactly of the clique nodes. This assignment satisfies the Boolean formula because each group of nodes contains a clique node and thus each clause contains at least one literal which is true. □

7.1.9 Node Cover $\leq_p$ Feedback Arc Set

I think there might be something wrong here. We know that arc deletion is fast, and it seems like we have false equivalence to the arc deletion problem occuring but I haven't had enough time to really dig into it.

Algorithm for Node Cover

$$\boxed{\text{I} \mapsto f(I) \rightarrow \boxed{\text{Algorithm for Feedback Arc Set}} \rightarrow S \mapsto h(S)}$$

Node Cover (Vertex Cover)
Input: A graph $G' = (N', A')$ and a positive integer ℓ
Property: $\exists R \subseteq N'$ such that $|R| \leq \ell$ and every arc in incident to a vertex in R.

Feedback Arc Set
Input: A directed graph $H = (V, E)$ and a positive integer k
Property: there exists $S \subseteq E$ such that every directed cycle of H contains an arc in S and $|S| \leq k$

Details I: A graph $G' = (N', A')$ and a positive integer ℓ

f: Function f takes the input I and convert it into an instance of Feedback Arc Set problem where
$V = N'x0, 1$
$E = \{<< u, 0 >, < u, 1 >> | u \in N\} \cup \{<< u, 1 >, < v, 0 >> | \{u, v\} \in E'$
$k = l$

Algorithm for Feedback Arc Set: Suppose there is a polynomial time algorithm for the Feedback Node Set problem.

S: $S \subseteq E$ such that every directed cycle of H contains an arc in S and $|S| \leq k$

h(S): function h takes S as an input and return the respective edge from G'.

Theorem 7.9. *$G' = (V', E')$ has a node cover of size l, iff there is a feedback arc set of size k on the graph, H, as defined above.*

Proof. ($\Rightarrow$) Suppose there is a node cover R on the graph G' of size l. Suppose we construct the digraph H as described above. For every $v \in R$ lets remove the edge (v_0, v_i) from H. According to our construction if a cycle in H enters a vertex from v_0 can only leave through the edge (v_0, v_1) and it can only enter edge v_1 through the edge (v_0, v_1). Thus if a cycle in H uses the edge (u_1, v_0), it must use the edge (u_0, u_1) and (v_0, v_1). At least one of these two edges have been removed since at least u or v must be in the vertex cover to cover the edge $(u, v) \in E'$. which implies removing either u or v makes the graph acyclic. Thus if G has a vertex cover of a size l then H has a feedback arc set of size k.

($\Leftarrow$) Suppose H has a Feedback Arc Set, S, of size k. WLOG we assume that the only edges removed are of the form (u_0, u_1) because if some other edge (u_1, v_0) is removed then all cycles in which this edge participated would have included the edge (v_0, v_1) too and we could remove this edge instead. We assume that the set if vertices $v \in V'$ for which the corresponding edge (v_0, v_1) is removed in E' forms a vertex cover for G'. Suppose there is some edge $(u, v) \in E'$ not covered. then in H the cycle $(u_0, u_1), (u_1, v_1), (v_0, v_1), (v_1, u_0)$ is unbroken by the removal of the edges contradicting the assumption. Thus G' must have a vertex cover of size $l = k$. ☐

7.1.10 Node Cover $\leq_p$ Feedback Node Set

Algorithm for Node Cover

$$\boxed{\text{I} \mapsto f(I) \rightarrow \boxed{\text{Algorithm for Feedback Node Set}} \rightarrow S \mapsto h(S)}$$

Node Cover (Vertex Cover)
Input: A graph $G' = (N', A')$ and a positive integer ℓ
Property: $\exists R \subseteq N'$ such that $|R| \leq \ell$ and every arc in incident to a vertex in R.

Feedback Node Set
Input: A directed graph $H = (V, E)$ and a positive integer k
Property: there exists $R \subseteq V$ such that every directed cycle of H contains a vertex in R and $|R| \leq k$

Details

I: A graph $G' = (N', A')$ and a positive integer ℓ

f: Function f takes the input I and convert it into an instance of Feedback Node Set problem where
$V = N'$
$E = \{< u, v > | \{u, v\} \in A'\}$
$k = l$

Algorithm for Feedback Node Set: Suppose there is a polynomial time algorithm for the Feedback Node Set problem.

S: $S \subseteq E$ such that every directed cycle of H contains a vertex in R and $|R| \leq k$

h(S): function h takes S as an input and return the respective vertices from G'.

Theorem 7.10. *$G' = (V', E')$ has a node cover of size l, iff there is a feedback arc set of size k on the graph, H, as defined above.*

Proof. (⇒) Let us assume there exists a vertex cover R of size $|R| \leq l$ on G'. Lets construct graph H as described above. Lets remove nodes from H which corresponds to the nodes in R, along with the edges connected to them. For any edge $(u, v) \in A'$ in G', there will be a cycle between nodes u and v in graph H. Therefore after removing these l vertices, which were the node cover of G', and their incident edges from H and all the nodes which were connected with the removed nodes and formed cycles with the removed nodes will no longer form a cycle. Thus since removing the R nodes from H makes the graph acyclic, it proves that these nodes are the feedback node set of H.

(⇐) Assume we construct H as defined above and it has a feedback node set $|S|$ of size k, and if they are removed all the cycles in the graph break. According to our construction each pair of vertices $u, v \in G'$ that has an edge between them have a cycle between $(u, v), (v, u)$ in H. Since removal of the k vertices in the feedback node set breaks all the cycle in the digraph H, therefore for each $\{u, v\} \in G'$ S must contain either u or v (or both). Thus S is a vertex cover of G'. □

7.1.11 Node Cover $\leq_p$ Set Covering

$$\text{Algorithm for Node Cover}$$
$$\boxed{\text{I} \mapsto f(I) \rightarrow \boxed{\text{Algorithm for Set Covering}} \rightarrow S \mapsto h(S)}$$

Node Cover (Vertex Cover)
Input: A graph $G' = (N', A')$ and a positive integer ℓ
Property: $\exists R \subseteq N'$ such that $|R| \leq \ell$ and every arc in incident to a vertex in R.

Set Covering
Input: A finite family of sets $\{S_j\}$ and a positive integer k
Property: there exists a subfamily $\{T_h\} \subseteq \{S_j\}$, $|\{T_h\}| \leq k$ and $\cup T_h = \cup S_j$

Details
I: A graph $G' = (N', A')$ and a positive integer ℓ

f: Function f takes the input I and convert it into a A family of sets S_j with one set for each vertex in G'. elements of S_j are defines to be the elements incident to j in G'. And $k = l$

Algorithm for Set Covering: Suppose there is a polynomial time algorithm for the Node cover problem which returns a set of vertices $R \subseteq V'$, $|R| \leq l$ and every arc in incident to a vertex in R.

S: Solution S is a subfamily $\{T_h\} \subseteq \{S_j\}$, $|\{T_h\}| \leq k$ and $\cup T_h = \cup S_j$

h(S): function h takes S as an input and Converts the set in $\{T_h\}$ back to their respective vertexes and puts them in a set.The set of nodes represents a Node Cover for G'.

Theorem 7.11. *$G' = (N', A')$ has a node cover of size l iff there exist a set cover of size $k = l$ on the family of sets S_j built as described above.*

Proof. (⇒) Suppose there exists a vertex cover of size l for the graph G' and we built a family of sets $\{S_j\}$ as described above. Let R be the set of vertices representing the vertex cover for G' then the vertices in R covers all the edges in A'. Since any subset T_j is the set of edges incident on vertex j, the union of T_j such that $j \in R$ must contain all the edges in A'. Since union of all the edges in A' forms the set U, $\{T_j : j \in R\}$ is a set cover if U, with size $k = l$. (⇐) Suppose there is a family of sets $\{S_j\}$, as described above, and there is a set cover for $\{S_j\}$ of size k. This means that there exists k sets, $\{T_h\}$, such that $\{T_h\} \subseteq \{S_j\}$ and $\cup\{T_h\} = \cup\{S_j\}$. Since the sets $\{S_j\}$ represents the nodes and the elements of the sets represents the edges, a node cover of the graph implies that each element of the graph G' is covered by the k sets or the l nodes. Thus there exist a node cover of size l in the graph G'. □

7.1.12 Directed Hamiltonian Circuit $\leq_p$ Undirected Hamiltonian Circuit

Algorithm for Directed Hamiltonian Circuit

$$\boxed{\text{I} \mapsto f(I) \rightarrow \boxed{\text{Algorithm for Undirected Hamiltonian Circuit}} \rightarrow S \mapsto h(S)}$$

Directed Hamiltonian Circuit
Input: A directed graph H
Property: H has a directed cycle which includes each vertex exactly once.

Undirected Hamiltonian Circuit
Input: A graph G
Property: G has a cycle which includes each vertex exactly once.

Details
I: A directed graph H

f: Function f takes the input I and convert it into an instance of Undirected Hamiltonian Circuit where $G = (N, E)$
$N = v$ x $\{0, 1, 2\}$
$E = \{\{<u, 0>, <u, 1>\}, \{<u, 1>, <u, 2>\} — u \in V\} \cup \{\{<u, 2>, <v, 0>\} \| <u, v> \in E\}$

Algorithm for Undirected Hamiltonian Circuit where:

S: Cycle which covers all the nodes in G exactly once.

h(S):

Theorem 7.12. *Graph H has a Directed Hamiltonian Circuit iff graph G has a Undirected Hamiltonian Circuit.*

Proof. (⇒) Suppose H has a Directed Hamiltonian Circuit. We construct G as defined above. We replace vertex $v \in V$ with $(v, 0), (v, 1), (v, 2) \in N$. In H to travel from (v, 0) to (v, 2) you have to go through (v, 1) and (v, 1) is connected to only (v, 0), (v, 2), further (v, 2) is connected to (u, 0) which represents the directed edge in G. Thus all vertices are visited once in the cycle. thus G has a Undirected Hamiltonian Circuit.

(⇐) Suppose G = (N, A) has a Undirected Hamiltonian Circuit. The $(u, 0)$ node can be considered as the "in-node" and the $(u, 2)$ can be considered as the "out-node". The edge between $(u, 2)$ to $(v, 0)$ in G represents a path from the node u to v in H. Nodes $(u, 0), (u, 1), (u, 2)$ in G represent the node u in H. since (u, 0) is represents the in-node and the (u, 2) represents the out node, the Directed Hamiltonian circuit in H is guaranteed to give a corresponding Undirected Hamiltonian circuit in G. □

7.1.13 Partition $\leq_p$ Max Cut

Algorithm for Partition

$$\boxed{\mathrm{I} \mapsto \mathrm{f(I)} \to \boxed{\text{Algorithm for Max Cut}} \begin{array}{c} \to x \mapsto h(x) \\ \hline \to N.S. \mapsto N.S. \end{array}}$$

Partition
Input: $(c_1, c_2, \dots c_s) \in \mathbb{Z}^s$
Property: there is an $I \subseteq \{1, 2, \dots s\}$ such that $\sum_{h \in I} c_h = \sum_{h \notin I} c_h$

Max Cut
Input: A graph $G = (N, A)$, a weight function $w : A \to \mathbb{Z}$ and a positive integer W
Property: $\exists S \subseteq N$ such that $\sum_{\{u,v\} \in A, u \in S, v \notin S} w(\{u, v\}) \geq W$

Details Given an instance I of partition, we form an instance f(I) of max cut using:

$$\begin{aligned} N &= \{1, ..., s\} \\ A &= \{\{i, j\} \mid i \in N, j \in N, i \neq j\} \\ w(\{i, j\}) &= c_i \cdot c_j \\ W &= \frac{1}{4}\left(\sum c_i\right)^2 \end{aligned}$$

Theorem 7.13. *There is a solution for instance I of partition $\iff$ there is a solution for f(I) in max cut.*

Proof. $\Rightarrow$ Assume there is a solution for partition. Let d = $\{d_1, ..., d_n\}$ enumerate the digits in the solution, and f = $\{f_1, ..., f_m\}$ the digits in the complement. Note that $d_1 + ... + d_n = f_1 + ... + f_m = \frac{1}{2}\sum c_i$. Because our graph is totally connected, we are guaranteed a cut on our graph with weight equal to:

$$\begin{aligned}
&\sum d_i(f_1 + ... + f_m)\\
&= \sum d_i \left(\frac{1}{2}\sum c_i\right)\\
&= (d_1)\left(\frac{1}{2}\sum c_i\right) + ... + (d_n)\left(\frac{1}{2}\sum c_i\right)\\
&= (d_1 + ... + d_n)\left(\frac{1}{2}\sum c_i\right)\\
&= \left(\frac{1}{2}\sum c_i\right)\left(\frac{1}{2}\sum c_i\right)\\
&= \frac{1}{4}\left(\sum c_i\right)^2\\
&= \left(\frac{1}{2}\sum c_i\right)^2\\
&= *\left(\frac{1}{2}\sum c_i\right)^2 \text{ since we have a partition, } 2|\sum c_i\text{. Note the square of an integer is an integer.}
\end{aligned}$$

Our cut can be given explicitly as $\{d_i\}$. $\Leftarrow$ Now assume that there is a max cut with weight greater or equal to W. Then there are digits $\{d_1, ..., d_n\}$ corresponding to vertices in the max cut, and $\{f_1, ..., f_m\}$ corresponding to the rest of the vertices, such that $\sum d_i(f_1 + ... + f_m) \geq W$. That is, $(d_1 + ... + d_n)(f_1 + ... + f_m) \geq W = *\left(\frac{1}{2}\sum c_i\right)^2 = *\left(\frac{1}{2}(d_1 + ... + d_n + f_1 + ... + f_m)\right)^2$. Assume for a contradiction (and without loss of generality) that $(d_1 + .. + d_n) = (f_1 + ... + f_m) + n, n \in$.

Let $\sum d_i = d, \sum f_j = f$. Then you have:

$$\begin{aligned}\sum d_i \sum f_j &= d \cdot f \\ &= (f+n)f \\ &= f^2 + nf. \\ &\text{but notably, with that assumption we have:} \\ W &= *\frac{1}{4}(f+d)^2 \\ &= *\frac{1}{4}(2f+n)^2 \\ &= *\frac{1}{4}\left(4f^2 + 4fn + n^2\right) \\ &= *f^2 + nf + \frac{n^2}{4} \\ &> f^2 + nf \\ &\implies W > \sum d_i \sum f_j\end{aligned}$$

which of course couldn't be, since we wouldn't have a max cut. We conclude that our digits are a partition. □

7.1.14 Satisfiability $\leq_P$ 0-1 Integer Programming

Algorithm for Satisfiability

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for 0-1 Integer Programming}} \begin{array}{c} \rightarrow x \mapsto h(x) \\ \hline \rightarrow N.S. \mapsto N.S. \end{array}}$$

Satisfiability
Input: Clauses $C_1, C_2, \ldots C_p$ with literals $x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}$.
Property: $C_1 \wedge C_2 \wedge \cdots \wedge C_p$ is *satisfiable*: if $\exists S \subseteq \{x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2}, \ldots \overline{x_n}\}$ such that if $x_i \in S$ then $\overline{x_i} \notin S$, and if $\overline{x_i} \in S$ then $x_i \notin S$ and $S \cap C_j \neq \emptyset$, $\forall j \in \{1, 2, \ldots p\}$

0-1 Integer Programming
Input: A integer matrix C and an integer vector $\vec{d}$
Property: there is a 0-1 vector $\vec{x}$ such that $C\vec{x} \geq \vec{d}$

Details Given an instance of Satisfiability, we form f(I) an instance of 0-1 integer programming as follows: Let the entries of C be defined so for i $\in \{1, ..., p\}$ and $j \in \{1, ..., n\}$, $c_{ij} = \begin{cases} 1 \text{ if } x_j \in C_i \\ -1 \text{ if } \bar{x}_j \in C_i \\ 0 \text{ otherwise} \end{cases}$

Let d_i = 1 - (the number of complemented variables in C_i).

Theorem 7.14. *There is a solution for an instance I of Satisfiability $\iff$ there is a solution for an instance f(I) of 0-1 Integer Programming.*

Proof. $\Rightarrow$ First assume there is a solution for an instance I of Satisfiability. In that case, let $x_j = 1 \iff x_j \in S$. We verify that for each row, $C_i \cdot x_j \geq d_i$. We do so inductively on the number of complemented variables, in a given C_i, n_i.

Base Case

$$\text{Let n} = 0.$$

If there are no complemented variables in C_i, then at least one satisfying literal, say x_j must have the value 1. In that case, $C_{ij}x_j = 1$. The rest of the products will be ≥ 0, so the sum of products is ≥ 1.

Inductive Step

$$\text{Assume } C_i \cdot \vec{x} \geq 1 - n_i \text{ holds for } n_i \leq n$$

Consider a clause C_i with n + 1 negated variables. Let f be a permutation that arranges our x_j so 1 to a are positive in the clause, a + 1 to b are negative, and b + 1 to m are those not included in the clause. Now, we write:

$$\begin{aligned}
C_i \cdot \vec{x} = {} & C_{if(1)}x_{f(1)} + ... + C_{if(a)}x_{f(a)} + C_{if(a+1)}x_{f(a+1)} \\
& + ... + C_{if(b)}x_{f(b)} + C_{if(b+1)}x_{f(b+1)} + ... + C_{if(m)}x_{f(m)} \\
& = C_{if(1)}x_{f(1)} + ... + C_{if(a)}x_{f(a)} + C_{if(a+1)}x_{f(a+1)} + ... \\
& + C_{if(b-1)}x_{f(b-1)} + C_{if(b+1)}x_{f(b+1)} + ... + C_{if(m)}x_{f(m)} + C_{if(b)}x_{f(b)}. \\
& \text{we assumed that:} \\
& C_{if(1)}x_{f(1)} + ... + C_{if(a)}x_{f(a)} + C_{if(a+1)}x_{f(a+1)} + ... \\
& + C_{if(b-1)}x_{f(b-1)} + C_{if(b+1)}x_{f(b+1)} + ... + C_{if(m)}x_{f(m)} \geq 1 - n. \\
& \text{Now we note that:} \\
& C_{if(1)}x_{f(1)} + ... + C_{if(a)}x_{f(a)} + C_{if(a+1)}x_{f(a+1)} + ... \\
& + C_{if(b-1)}x_{f(b-1)} + C_{if(b+1)}x_{f(b+1)} + ... + C_{if(m)}x_{f(m)} + C_{if(b)}x_{f(b)}. \\
& \geq C_{if(1)}x_{f(1)} + ... + C_{if(a)}x_{f(a)} + C_{if(a+1)}x_{f(a+1)} + ... \\
& + C_{if(b-1)}x_{f(b-1)} + C_{if(b+1)}x_{f(b+1)} + ... + C_{if(m)}x_{f(m)} + (-1) \text{ (since } C_{if(b)}x_{f(b)} \text{ is at least -1)} \\
& \geq (1-n) - 1 \text{ (from our assumption)} \\
& = 1 - (n+1) \\
& \implies C_i \cdot \vec{x} \geq 1 - (n+1).
\end{aligned}$$

Thus our instance of 0-1 Integer Programming is satisfied.

$\Leftarrow$ Now assume that our instance of 0-1 I.P. has a solution, $\vec{x}$. We verify first that there are no

contradicting literals, which is immediately true since the literal assignments are encoded with binary values. Next we verify that for each C_i, $C_i \cap S \neq \emptyset$. Assume this is not true. Then for some C_i, $C_i \cap S = \emptyset$. Let n be the number of negative literals in C_i. Since $C_i \cap S = \emptyset$, we do not have a satisfying literal for the clause. Then every positive literal appearing in the clause must be assigned false, and every false literal that appears in the clause must have been assigned true. Recall that for $C_{ij} = \begin{cases} 1 \text{ if } x_j \in C_i \\ -1 \text{ if } \bar{x}_j \in C_i \\ 0 \text{ otherwise} \end{cases}$ from before. Because the variable assignment comes from $\vec{x}$, and using the row in our matrix corresponding to our unsatisfied C_i, we have: $C_{ij}x_j = \begin{cases} 0 & \text{if } C_{ij} = 1 \\ -1 & \text{if } C_{ij} = -1 \\ 0 & \text{if } C_{ij} = 0 \end{cases}$.

Then $C_i \cdot \vec{x} = -n$, which is strictly less than $(1 - n) = d_i$. But then our inequality does not hold! We conclude that we have a solution for Satisfiability. □

7.1.15 Chromatic Number $\leq_p$ Exact Cover

$$\text{Algorithm for Chromatic Number}$$
$$\boxed{\text{I} \mapsto \text{f(I)} \to \boxed{\text{Algorithm for Exact Cover}} \;\frac{\to x \mapsto h(x)}{\to N.S. \mapsto N.S.}}$$

Chromatic Number
Input: A graph $G = (N, E)$ and a positive integer k
Property: there is a function $\phi : N \to \mathbb{Z}_k$ such that if u and v are adjacent then $\phi(u) \neq \phi(v)$

Exact Cover
Input: A family $\{S_j\}$ of subsets of the set $\{u_1, u_2, \dots u_t\}$
Property: there is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that $\{T_h\}$ are disjoint and $\cup T_h = \cup S_j = \{u_1, u_2, \dots u_t\}$

Details Given an instance I of chromatic number, f(I) goes as follows: The set of elements is defined as

$$N \cup A \cup \{(u, e, f) \mid \text{ u is incident with e and } 1 \leq f \leq k\}$$

The sets S_j are equal to:

$\{U_i\} \cup \{E_h\}$
where U_i, E_h are defined as:
$U_i = \{\{u\} \cup \{(u, e, f)\} \mid 1 \leq f \leq k, u \in N, \text{ e is incident with u}\}$
$E_h = \{\{e\} \cup \{(u, e, f) \mid f \neq f_1\} \cup \{(v, e, f) \mid f \neq f_2\} \mid e \in A, f_1, f_2 \in \mathbb{Z}_k, f_1 \neq f_2 \text{ and u, v are adjacent with e.}\}$

Given a solution x = $\{T_j\}$ to exact cover, we define h(x) as follows: For n $\in N$, if n has no edges let $\phi(n) = 0$. If n does have an edge, choose $\phi(n)$ to be f, where $\{n\} \cup \{(n, e, f)\}$ is included in the exact cover. Note that this must exist, or we would not have a complete cover of the nodes.

Theorem 7.15. *There is a solution for an instance I of chromatic number* $\iff$ *there is a solution for f(I) in exact cover.*

Proof. $\Rightarrow$ Assume that you have a solution, ϕ, for chromatic number. We generate an exact cover as follows:

$$\bigcup_{n \in N} A_n \cup B_n$$

where A_n and B_n are defined as:

$A_n = \{\{n\} \cup \{(n, e, \phi(n)) \mid \text{ e is an edge of n}\}\}$

$B_n = \{\{e\} \cup \{(n, e, f) \mid f \neq \phi(n)\} \cup \{(v, e, f) \mid f \neq \phi(v)\} \mid \phi(n) > \phi(v) \text{ and n, v are connected by edge e.}\}$

Note that $A_n \in \{U_i\}$ by construction. Further since we require $\phi(v) \neq \phi(n)$ for any v adjacent to n, the B_n are guaranteed to be in $\{E_h\}$. What's more, each (n, e, f) exists in exactly one set from our requirement that $\phi(n) > \phi(v)$ and the construction requirement that $f \neq \phi(n), g \neq \phi(v)$. In addition to the $\{(n, e, f)\}$, we find that the nodes and edges are covered by one set in our constructed cover (if you are unsure for the edges, recall our restriction that $\phi(n) > \phi(v)$). Thus we have an exact cover for our elements!

$\Leftarrow$ Assume that our instance in exact cover has a solution, $\{T_j\}$. Let u and v be adjacent vertices in N, connected by edge e $\in E$. To cover our u, v elements we must have $\{u, (u, e, m)\}, \{v, (v, e, n)\} \in \{T_j\}$. Assume for a contradiction that m = n, or from our definition of ϕ that $\phi(u) = \phi(v)$. Assume WLOG that $\{e\} \cup \{(u, e, f)\} \cup \{(v, e, g)\}, f \neq m, g \neq k$ for some $k \neq m$, is included to cover e. We verify that this is WLOG by noting that the only other option is $\{e\} \cup \{(u, e, f)\} \cup \{(v, e, g)\}, f \neq k, g \neq m$ for some $k \neq m$. But then we must include another set like $\{v, (v, e, k)\}$ to cover (v, e, k), and then we have no exact cover.
Then $\phi(u) \neq \phi(v)$ as desired. □

7.1.16 Satisfiability with at Most 3 Literals Per Clause $\leq_p$ Chromatic Number

Algorithm for Satisfiability With at Most 3 Literals Per Clause

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Chromatic Number}} \begin{array}{c} \rightarrow x \mapsto h(x) \\ \hline \rightarrow N.S. \mapsto N.S. \end{array}}$$

Satisfiability With At Most Three Literals Per Clause
Input: Clauses $D_1, D_2, \ldots D_r$, each consisting of at most 3 literals from $\{u_1, u_2, \ldots u_m, \overline{u_1}, \overline{u_2}, \ldots \overline{u_m}\}$
Property: $\{D_1, D_2, \ldots D_r\}$ is satisfiable

Chromatic Number
Input: A graph $G = (N, E)$ and a positive integer k
Property: there is a function $\phi : N \rightarrow \mathbb{Z}_k$ such that if u and v are adjacent then $\phi(u) \neq \phi(v)$

Details Given an instance I of Satisfiability With at Most 3 Literals Per Clause, we form an instance f(I) of Chromatic number as follows:

Assume WLOG that m $\geq$ 4. Let

$$\begin{aligned}
N &= \{u_1, ..., u_m\} \\
&\cup \{\bar{u}_1, ..., \bar{u}_m\} \\
&\cup \{v_1, ..., v_m\} \\
&\cup \{D_1, ..., D_r\} \\
A &= \{\{u_i, \bar{u}_i\} \mid i = 1, ..., n\} \\
&\cup \{\{v_i, v_j\} \mid i \neq j\} \\
&\cup \{\{v_i, u_j\} \mid i \neq j\} \\
&\cup \{\{v_i, \bar{u}_j\} \mid i \neq j\} \\
&\cup \{\{u_i, D_f\} \mid u_i \notin D_f\} \\
&\cup \{\{\bar{u}_i, D_f\} \mid \bar{u}_i \notin D_f\} \\
k &= m + 1
\end{aligned}$$

Theorem 7.16. *There is a solution for an instance I of Satisfiability With at Most 3 Literals Per Clause $\iff$ there is a solution for f(I) in chromatic number.*

Proof. First we acknowledge our assumption that m $\geq$ 4. If this is not the case, our satisfiability problem is bounded by a polynomial time solving algorithm.
$\Rightarrow$ Assume you have a solution for Satisfiability With at Most 3 Literals Per Clause, S. Let J = $S \cup \{u_i \mid \bar{u}_i \notin S\}$. We construct the m + 1 equivalence classes as follows: for each $j \in J$, $[j] = \{v_j, u_j\} \cup \{D_i \mid j = max(\{j \in J \mid u_j \in D_i\})\}$. There are m such classes, because J has m elements. Let the last equivalence class ($[m+1]$) consist of all of the remaining vertices. This guarantees a partition on our set of vertices. We now confirm that $\phi : \mathbb{Z}_{m+1} \to V$ where $\{1, ..., m{+}1\}$ enumerates our equivalence classes, and $\phi(k)$ is equal to the equivalence class enumerated by k, is a solution for chromatic color. Given any of the $[j]$ classes, we know that v_j and u_j are not adjacent from construction. Furthermore, if there are any D_i in the equivalence classes, then we must have $j \in D_i$, so again from construction they cannot be adjacent. We now consider the equivalence class $[m+1]$. Because we have a Satisfiability With at Most 3 Literals Per Clause solution, this will not contain any of the D_i. To see why, note that we constructed our equivalence class to ensure that every D_i corresponds to exactly one of the $[j]$ classes. Our $[m+1]$ class cannot contain v_j from our choice of equivalence classes. The literal vertices are not connected to each other, so we are guaranteed an independent set. We have verified our solution for chromatic color.
$\Leftarrow$ Now assume you have a solution for chromatic number. Define S = $\{u_j \mid \phi(u_j) = \phi(D_i)\} \cup \{\bar{u}_j \mid \phi(\bar{u}_j) = \phi(D_i)\}$ for any of the D_i vertices in our graph. It's quick to check that there is a literal from each clause in S, so we verify that there are no complementary literals. Assume without loss of generality that $u_j \in S$. Then $\phi(u_j) = \phi(D_i)$ for some D_i in our graph. For a contradiction, assume that $\bar{u}_j$ is also in S. In that case, one of the D_i have to be colored with our leftover color class $[m+1]$ from the m v_i vertices. For each pair of literals, one of them has to be in the $[m+1]$ class. Each literal in the $[m+1]$ expression must appear in the expression, or they would be adjacent to the D_i vertex. If we have more than 3 literals, this becomes a contradiction from the construction of our graph. □

7.1.17 Knapsack $\leq_p$ Sequencing

Algorithm for Knapsack

$$\boxed{\text{I} \mapsto \text{f(I)} \to \boxed{\text{Algorithm for Sequencing}} \;\begin{array}{c} \to x \mapsto h(x) \\ \hline \to N.S. \mapsto N.S. \end{array}}$$

Knapsack
Input: $(a_1, a_2, \dots a_r, b) \in \mathbb{Z}^{r+1}$
Property: $\sum a_j x_j = b$ has a 0-1 solution

Job Sequencing
Input: $(T_1, T_2, \dots T_p) in \mathbb{Z}^p$, the execution time, $(D_1, D_2, \dots D_p) in \mathbb{Z}^p$, the deadline, $(P_1, P_2, \dots P_p) in \mathbb{Z}^p$, the penalty and a positive integer k
Property: there is a permutation, π of $\{1, 2, \dots p\}$ such that

$$k \geq \begin{cases} P_{\pi(j)} & \text{if } T_{\pi(1)} + \cdots + T_{\pi(j)} > D_{\pi(j)} \\ 0 & \text{otherwise} \end{cases}$$

Details Given an instance of knapsack, produce an instance of sequencing as follows: Let p = r, $T_i = P_i = a_i$, $D_i = b$, and $k = \sum a_i - b$.
Given a solution π for sequencing, choose the first n values for which $a_{\pi(1)} + ... + a_{\pi(n)} = b$. Let $x_{\pi(1)}, ..., x_{\pi(n)} = 0$, and the remaining $x_{\pi(n+1)}, ..., x_{\pi(r)} = 0$.

Theorem 7.17. *There is a solution $(x_1, ..., x_r)$ for knapsack $\iff$ there is a solution π for our instance of Sequencing.*

Proof. $\Rightarrow$ Assume there is a solution for knapsack. Choose a permutation π so that $a_\pi(1) + ... + a_\pi(n) = b$. Then the first n penalties are not counted. The sum is reduced to $\sum_{j=n+1}^{r} a_j = \sum a_i - b$. Then there is a solution for our instance of sequencing.

$\Leftarrow$ Now assume that there is a solution for sequencing. That is, $\exists\, \pi$ so $\sum a_i - b \geq \sum \begin{cases} a_{\pi(j)} \text{ if } a_{\pi(1)} + ... + a_{\pi(j)} > b \\ 0 \text{ otherwise} \end{cases}$

We consider two cases for this, mostly due to index gymnastics. If $a_{\pi(1)} + ... + a_{\pi(j)}$ is never greater than b, then $\sum a_i - b \geq 0$. Since b is never larger than our partial sum, we have

$$\begin{aligned} \sum a_i - \sum a_i &= 0 \\ &\geq \sum a_i - b \\ &\geq 0 \end{aligned}$$

Thus $\sum a_i = b$. Otherwise, we have

$$\begin{aligned} &\sum a_i - b \geq a_{\pi(n+1)} + ... + a_r \\ \implies &\sum a_i - \left(a_{\pi(n+1)} + ... + a_r\right) \geq b \\ &\leq a_{\pi(1)} + ... + a_{\pi(n)} \\ \implies &a_{\pi(1)} + ... + a_{\pi(n)} = b. \end{aligned}$$

. We have found a subset of our a_i which sum to b, as we originally sought. □

7.1.18 Knapsack $\leq_p$ Partition

$$\begin{array}{c} \text{Algorithm for Knapsack} \\ \boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Partition}} \begin{array}{c} \rightarrow x \mapsto h(x) \\ \hline \rightarrow N.S. \mapsto N.S. \end{array}} \end{array}$$

Knapsack
Input: $(a_1, a_2, \ldots a_r, b) \in \mathbb{Z}^{r+1}$
Property: $\sum a_j x_j = b$ has a 0-1 solution

Partition
Input: $(c_1, c_2, \ldots c_s) \in \mathbb{Z}^s$
Property: there is an $I \subseteq \{1, 2, \ldots s\}$ such that $\sum_{h \in I} c_h = \sum_{h \notin I} c_h$

Details Given an instance I of knapsack, we form an instance f(I) of partition as follows:

$$\begin{aligned} s &= r + 2 \\ c_i &= a_i;\ i = 1, ..., r \\ c_{r+1} &= b + 1 \\ c_{r+2} &= \sum a_i + 1 - b \end{aligned}$$

Given a solution I of knapsack, we define h(I) as follows. If $c_{r+2} \in I$, then let $X = I$. Otherwise, let $X = I^C$. Then

$$x_i = \begin{cases} 1 & \text{if } c_i \in X \\ 0 & \text{otherwise} \end{cases}$$

and h(I) = $\{x_i\}$.

Theorem 7.18. *There is a solution $h(I) = (x_1, ..., x_r)$ for an instance I of Knapsack $\iff$ there is a solution $I \subseteq \{1, ..., r+2\}$ for f(I) in partition.*

Proof. There is a solution for Knapsack $\iff$ some subset of the a_i, let's assume they are indexed by K, sum to b. This is true exactly when

$$\begin{aligned}
c_{r+2} + \sum_{k \in K} a_k &= \sum a_i + 1 - b + \sum_{k \in K} a_k \\
&= \sum a_i + 1 - b + b \\
&= \sum a_i + 1. \\
&\text{but we also have:} \\
c_{r+1} + \sum_{j \notin K} a_j &= b + 1 + \sum_{j \notin K} a_j \\
&= \sum_{k \in K} a_k + 1 + \sum_{j \notin K} a_j \\
&= \sum a_i + 1
\end{aligned}$$

Which is true $\iff$ we have a partition on our set, formed by $K \cup \{r+2\}$. □

7.1.19 Node Cover $\leq_p$ Directed Hamilton Circuit

Algorithm for Node Cover

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Directed Hamiltonian Circuit}} \;\; \frac{\rightarrow x \mapsto h(x)}{\rightarrow N.S. \mapsto N.S.}}$$

Node Cover (Vertex Cover)
Input: A graph $G' = (N', A')$ and a positive integer ℓ
Property: $\exists R \subseteq N'$ such that $|R| \leq \ell$ and every arc in incident to a vertex in R.

Directed Hamiltonian Circuit
Input: A directed graph H
Property: H has a directed cycle which includes each vertex exactly once.

Details Given an instance ((N', A'), l) of node cover, we form a directed graph as follows: without loss of generality, assume that A' = $\mathbb{Z}_m$. Now, let

$$V = \{a_1, ..., a_l\} \cup \{(u, i, \alpha) \mid u \in N' \text{ is incident with i } \in A' \wedge \alpha \in \{0, 1\}\}.$$

The a_i correspond to the total nodes we will allow in our node cover solution. Define the edges as:

$$\begin{aligned}
E =&\{< (u,i,0),(u,i,1) >| (u,i,0) \in V\} \\
&\cup\{< (u,i,\alpha),(v,i,\alpha) >| i \in A', \text{ u and v are incident with i, } \alpha \in \{0,1\}\} \\
&\cup\{< (u,i,1),(u,j,0) >| i \leq j, \text{ u is incident with i and j, and } \nexists h, i < h < j, \\
&\quad \text{where u is incident with h.}\} \\
&\cup\{< (u,i,1),a_f >| 1 \leq f \leq l \wedge \nexists h > i \text{ so that u is incident with h}\} \\
&\cup\{< a_f,(u,i,0) >| 1 \leq f \leq l \wedge \nexists h < i \text{ so that u is incident with h}\}
\end{aligned}$$

Given a solution $\{e_1, e_2, ..., e_m\}$ (where m = $|V|$) for our instance of Directed Hamiltonian Circuit, the set $\{u \mid < a_f, (u,i,0) > \in \{e_1, ..., e_n\}\}$ is a node cover for G'.

Theorem 7.19. *There is a node cover $\{v_1, ..., v_l\}$ for G' $\iff$ There exists a Directed Hamilton Circuit in our instance.*

Proof. $\Rightarrow$ Let $\{v_1, ..., v_l\}$ be a node cover for G'. Pick an onto mapping from $\{v_i\}$ to $\{a_j\}$ so we may index them together. Now fix an a_i, v_i pair. Let $\{e_1, ..., e_n\}$ be the set of edges between v_i and any x_j in our graph. Construct a path from $(v_i, e_1, 0)$ to $(v_i, e_n, 1)$ as follows: start at $(v_i, e_j, 0)$. If the other end of e_j is in $\{v_i\}$ simply move $(v_i, e_j, 0) \rightarrow (v_i, e_j, 1) \rightarrow (v_i, e_{j+1}, 0)$. Note that those nodes will be addressed later when we get to their corresponding a_i in the path. Otherwise, let x_i be the other node adjacent to e_j and move $(v_i, e_j, 0) \rightarrow (x_j, e_j, 0) \rightarrow (x_j, e_j, 1) \rightarrow (v_i, e_j, 1) \rightarrow (v_i, e_{j+1}, 0)$. Glue together the previously described path and the path $a_i \rightarrow (v_i, e_1, 0)$, as well as $(v_i, e_n, 1) \rightarrow a_{i+1(mod\ l)}$. This must cover every node in our graph by construction, and is a cycle. $\Leftarrow$ Assume there is a Directed Hamiltonian Circuit in our instance. Let v_i be the set of vertices labeling the nodes that each of the a_i point to. Note these are unique, as we have a Directed Hamiltonian Circuit. Now assume BWOC that there is some a, b $\in G'$ where $a, b \notin \{v_i\}$ have an edge between them. Let the edge between them be n. This introduces the nodes (a, n, 0), (a, n, 1), (b, n, 0), and (b, n, 1). There is no way to reach these vertices from our other circuit, however, since the n edge is not shared with any of the $\{v_i\}$ and the $\{a_i\}$ are already accounted for. Then there must be no vertices in G' that are both not in $\{v_i\}$ and share an edge. We conclude that $\{v_i\}$ is a vertex cover for G'. $\square$

7.1.20 Exact Cover $\leq_p$ Steiner Tree

Algorithm for Exact Cover

$$\boxed{\text{I} \mapsto \text{f(I)} \rightarrow \boxed{\text{Algorithm for Steiner Tree}} \begin{array}{c} \rightarrow x \mapsto h(x) \\ \hline \rightarrow N.S. \mapsto N.S. \end{array}}$$

Exact Cover
Input: A family $\{S_j\}$ of subsets of the set $\{u_1, u_2, \ldots u_t\}$
Property: there is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that $\{T_h\}$ are disjoint and $\cup T_h = \cup S_j = \{u_1, u_2, \ldots u_t\}$

Steiner Tree
Input: A graph $G = (N, A)$, $R \subseteq N$, a weighting function $w : A \rightarrow \mathbb{Z}$ and a positive integer k
Property: G has a subtree of weight $\leq k$ containing the set of vertices in R

Details Given an instance, I, of Exact Cover, form an instance f(I) of Steiner Tree as follows:

$$\begin{aligned}
k &= |\{u_i\}| \\
N &= \{n_0\} \cup \{S_j\} \cup \{u_i\} \cup \{u_i^*\} \\
R &= \{n_0\} \cup \{u_i\} \\
A &= \{\{n_0, S_j\}\} \cup \{\{S_j, u_i\} \mid u_i \in S_j\} \cup \{\{u_i, u_i^*\}\} \\
w(\{n_0, S_j\}) &= |S_j| \\
w(\{S_j, u_i\}) &= k + 1 \\
w(\{u_i, u_i^*\}) &= -(k + 1)
\end{aligned}$$

Given a solution x = T = (N_T, E_T) for our instance of Steiner Tree, define h(x) as $\{S_j\} \cap N_T$.

Theorem 7.20. *There is a solution for an instance I of exact cover $\iff$ there is a solution for f(I) in Steiner tree.*

Proof. $\Rightarrow$ Assume there is some $\{T_j\}$ which is an exact cover for $\{u_i\}$. Construct a tree of height three as $n_0 \to T_j$ for each T_j, $T_j \to u_i$ for each $u_i \in T_j$, and u_i to u_i^* for each $u_i \in T_j$. Since $\{T_j\}$ is an exact cover, there are no distinct T_j, T_k pointing to the same u_i and certainly no distinct u_i, u_h point to the same u_f^*. Then we have a tree. Now note that since the $\{T_j\}$ cover our set, each of the required u_i are included in our tree. The T_j form a partition on our set, so $\sum |T_j| = |\{u_i\}|$. In that case, $\sum_{T_j} w(\{n_0, T_j\}) = k$. Now note that each u_i is accessible from a single edge from one of the T_j we have selected. Those paths have weight k + 1, which we offset by using the path with weight -(k + 1) to u_i^*. The total weight on our tree is k.
$\Leftarrow$ Assume you have a solution for f(I) in Steiner Tree. To access the u_i, we must come from a set node, say T_k. To access T_k, we may either "hop" over from a distinct u_j also in T_k that was accessed from a distinct T_h, or directly from n_0. We rule out the first option, since such a traversal would add a net (k + 1) weight to our tree. In that case, each u_i must be accessed from a T_h connected to n_0. Now assume for a contradiction that there is some u_i in $T_m, T_n, n \neq m$. Since every u_i is in one T_p, $\sum |T_p| \geq |u_i|$, but if we have a repeat $\sum |T_p| > |u_i|$. Then we do not have a solution for our instance of Steiner Tree. We conclude that each u_i exists in a unique T_p. ☐

7.1.21 Exact Cover $\leq_p$ Knapsack

Algorithm for Exact Cover

$$\boxed{\text{I} \mapsto f(I) \to \boxed{\text{Algorithm for Knapsack}} \to S \mapsto h(S)}$$

Exact Cover
Input: A family $\{S_j\}$ of subsets of the set $\{u_1, u_2, \ldots u_t\}$
Property: there is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that $\{T_h\}$ are disjoint and $\cup T_h = \cup S_j = \{u_1, u_2, \ldots u_t\}$

Knapsack
Input: $(a_1, a_2, \ldots a_r, b) \in \mathbb{Z}^{r+1}$
Property: $\sum a_j x_j = b$ has a 0-1 solution

Details
I: A family $\{S_j\}$ of subsets of the set $\{u_1, u_2, \dots u_t\}$

f: Function f takes the input I and convert it into an instance of Knapsack where
$d = |S_j| + 1$
$\varepsilon_{ij} = \begin{cases} 1 \text{ if } u_i \in S_j \\ 0 \text{ if } u_i \notin S_j \end{cases}$
$a_j = \sum \varepsilon_{ji} d^{i-1}$
$b = \dfrac{d^t - 1}{d - 1}$

Algorithm for Knapsack: Suppose there is a polynomial time algorithm for the Knapsack problem which returns a 0-1 solution for $\sum a_j x_j = b$.

S: Solution S is a 0-1 solution for f(I).

h(S):

Theorem 7.21. *S_j has an Exact Cover iff $(a_1, a_2, \dots a_r, b)$, constructed as stated above, has a $0-1$ solution for $\sum a_j x_j = b$*

Proof. ($\Rightarrow$) Suppose there is a solution $T_h \subseteq S_j$ for an instance I of exact cover. Based on our construction for knapsack as defined above, for all $u_i \in S_j$, $a_j = \sum(|S_j + 1|)^{i-1}$ and $b = \dfrac{d^t - 1}{d - 1}$. Suppose there is a 0-1 solution for our constructed instance of Knapsack such that for any $a \in a_j$ that represents the corresponding $t \in T_h$, then $\sum a_j x_j = \Sigma_{i=1}^{t}(|S_j + 1|)^{i-1} = \dfrac{(|S_j + 1|)^t - 1}{(|S_j + 1|) - 1}$. As defined above $d = |s_j| +$ thus $\sum a_j x_j = \sum d^{i-1} = \dfrac{d^t - 1}{d - 1}$. expanding the left hand side of the equation we get $(d^{1-1} + d^{2-1} \cdots + d^{t-1}) = \dfrac{d^t - 1}{d - 1}$.By multiplying the equation with $(d - 1)$, we get $(d - 1)(d^{1-1} + d^{2-1} \cdots + d^{t-1}) = d^t - 1$ on solving we get $d - 1 = d - 1$, thus there exists a 0-1 solution for Knapsack.

($\Leftarrow$) Suppose there is a 0-1 Knapsack solution for Knapsack for $x \in x_j$ such that $\sum a_j x_j = b$, where $\sum a_j x_j$ and b are as defined above. As per our construction only one $a_j \in a_j$ represents an element $u_i\ inS_j$. Therefore for there to be a 0-1 solution for Knapsack each $u_i\ inS_j$ must appear only once in $\cup T_h$ and if each set in T_h is disjoint and $\cup S_j = \cup T_h$ then by definition T_h is an exact cover of S_j. ◻

7.2 Problem Statements

Satisfiability
Input: Clauses $C_1, C_2, \dots C_p$ with literals $x_1, x_2, \dots x_n, \overline{x_1}, \overline{x_2}, \dots \overline{x_n}$.
Property: $C_1 \wedge C_2 \wedge \cdots \wedge C_p$ is *satisfiable*: if $\exists S \subseteq \{x_1, x_2, \dots x_n, \overline{x_1}, \overline{x_2}, \dots \overline{x_n}\}$ such that if $x_i \in S$ then $\overline{x_i} \notin S$, and if $\overline{x_i} \in S$ then $x_i \notin S$ and $S \cap C_j \neq \emptyset$, $\forall j \in \{1, 2, \dots p\}$

0-1 Integer Programming
Input: A integer matrix C and an integer vector $\vec{d}$
Property: there is a 0-1 vector $\vec{x}$ such that $C\vec{x} \geq \vec{d}$

Clique
Input: A graph G and a positive integer k
Property: G has a clique of size k

Set Packing
Input: A family of sets $\{S_j\}$ and a positive integer ℓ
Property: $\{S_j\}$ has ℓ mutually exclusive sets.

Node Cover (Vertex Cover)
Input: A graph $G' = (N', A')$ and a positive integer ℓ
Property: $\exists R \subseteq N'$ such that $|R| \leq \ell$ and every arc in incident to a vertex in R.

Set Covering
Input: A finite family of sets $\{S_j\}$ and a positive integer k
Property: there exists a subfamily $\{T_h\} \subseteq \{S_j\}$, $|\{T_h\}| \leq k$ and $\cup T_h = \cup S_j$

Feedback Node Set
Input: A directed graph $H = (V, E)$ and a positive integer k
Property: there exists $R \subseteq V$ such that every directed cycle of H contains a vertex in R and $|R| \leq k$

Feedback Arc Set
Input: A directed graph $H = (V, E)$ and a positive integer k
Property: there exists $S \subseteq E$ such that every directed cycle of H contains an arc in S and $|S| \leq k$

Directed Hamiltonian Circuit
Input: A directed graph H
Property: H has a directed cycle which includes each vertex exactly once.

Undirected Hamiltonian Circuit
Input: A graph G
Property: G has a cycle which includes each vertex exactly once.

Satisfiability With At Most Three Literals Per Clause
Input: Clauses $D_1, D_2, \ldots D_r$, each consisting of at most 3 literals from $\{u_1, u_2, \ldots u_m, \overline{u_1}, \overline{u_2}, \ldots \overline{u_m}\}$
Property: $\{D_1, D_2, \ldots D_r\}$ is satisfiable

Chromatic Number
Input: A graph $G = (N, E)$ and a positive integer k
Property: there is a function $\phi : N \to \mathbb{Z}_k$ such that if u and v are adjacent then $\phi(u) \neq \phi(v)$

Clique Cover
Input: A graph $G' = (N', E')$ and a positive integer ℓ
Property: N' is the union of ℓ or fewer cliques

Exact Cover
Input: A family $\{S_j\}$ of subsets of the set $\{u_1, u_2, \ldots u_t\}$
Property: there is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that $\{T_h\}$ are disjoint and $\cup T_h = \cup S_j = \{u_1, u_2, \ldots u_t\}$

Hitting Set
Input: A family $\{U_i\}$ of subsets of $\{s_1, s_2, \ldots s_r\}$
Property: there is a set W such that for each i, $|W \cap U_i| = 1$

Steiner Tree
Input: A graph $G = (N, A)$, $R \subseteq N$, a weighting function $w : A \to \mathbb{Z}$ and a positive integer k
Property: G has a subtree of weight $\leq k$ containing the set of vertices in R

3-Dimensional Matching
Input: $U \subseteq T \times T \times T$ where T is a finite set
Property: there is a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of W agree in any coordinate

Knapsack
Input: $(a_1, a_2, \ldots a_r, b) \in \mathbb{Z}^{r+1}$
Property: $\sum a_j x_j = b$ has a 0-1 solution

Job Sequencing
Input: $(T_1, T_2, \ldots T_p) in \mathbb{Z}^p$, the execution time, $(D_1, D_2, \ldots D_p) in \mathbb{Z}^p$, the deadline, $(P_1, P_2, \ldots P_p) in \mathbb{Z}^p$, the penalty and a positive integer k
Property: there is a permutation, π of $\{1, 2, \ldots p\}$ such that

$$k \geq \begin{cases} P_{\pi(j)} & \text{if } T_{\pi(1)} + \cdots + T_{\pi(j)} > D_{\pi(j)} \\ 0 & \text{otherwise} \end{cases}$$

Partition
Input: $(c_1, c_2, \dots c_s) \in \mathbb{Z}^s$
Property: there is an $I \subseteq \{1, 2, \dots s\}$ such that $\sum_{h \in I} c_h = \sum_{h \notin I} c_h$

Max Cut
Input: A graph $G = (N, A)$, a weight function $w : A \to \mathbb{Z}$ and a positive integer W
Property: $\exists S \subseteq N$ such that $\sum_{\{u,v\} \in A, u \in S, v \notin S} w(\{u, v\}) \geq W$

7.3 Corrections on Original Reductions

7.3.1 Corrections

We have made notable corrections and adjustments to Karp's work in the following areas. For several problems the exact reduction provided was not correct. Accordingly we have sub-divided our corrections into issues with problem statements or with reductions. Note that further modifications were made on syntax on several problems as we saw fit.

1. **Problem Statements**
   (a) 0-1 Integer Programming
      i. we found the original 0-1 Integer Programming specifications to be inadequate for the reduction supplied. We modified the problem in our NP-Complete subset to use the inequality version of 0-1 Integer Programming, which allows for a clean reduction from Satisfiability into 0-1 Integer programming.
   (b) Feeback Node Set
      i. In the original problem statement for Feedback Node Set, it is not stated that $|R| \leq k$.
   (c) Feedback Arc Set
      i. In the original problem statement for Feedback Arc Set, it is not stated that $|S| \leq k$.
   (d) Knapsack
      i. A small typo was corrected to say $(a_1, a_2, ..., a_r, b) \in \mathbb{Z}^{r+1}$.
2. **Reductions**
   (a) Exact cover to Steiner Tree
      i. This was the largest correction we made to a reduction. In the original paper, the reduction provided has a counter example given by:

$$S_1 = \{1, 5\}$$
$$S_2 = \{1, 3\}$$
$$S_3 = \{1, 2\}$$
$$S_4 = \{2, 3\}$$
$$S_5 = \{2, 5\}$$
$$S_6 = \{3, 4\}$$

      We present a modified reduction for exact cover to Steiner Tree.

(b) 3-Satisfiability to Chromatic Number

   i. There were minor transcription errors on this reduction.

(c) Exact Cover to 3-Dimensional Matching

   i. In our paper, we assumed a value for σ that was sufficient for the reduction. It was not originally specified.

(d) Knapsack to Sequencing

   i. The value for k was significant to this reduction, and not previously provided.

(e) SAT to 0-1

   i. A minor labeling error (b_i vs. d_i) was corrected.

References

[1] Giorgio Ausiello, Alessandro D'Atri, and Marco Protasi. On the structure of combinatorial problems and structure preserving reductions. *International Colloquium on Automata, Languages, and Programming*, pages 45–60, 1977.

[2] Vicky Choi. Different adiabatic quantum optimization algorithms for the np-complete exact cover problem. *Proceedings of the National Academy of Sciences of the United States of America*, 108(7):E19–E20, 2011.

[3] Stephen A. Cook. The complexity of theorem-proving procedures. *The Third Annual ACM Symposium*, pages 151–158, 1971.

[4] Jerzy A Filar, Michael Haythorpe, and Richard Taylor. Linearly-growing reductions of karp's 21 np-complete problems. 2019.

[5] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[6] Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509.

[7] Luc Longpre and Paul Young. Cook reducibility is faster than karp reducibility in np. *Journal of Computer Science and System Sciences*, 41:389–401, 1990.

[8] Neeraj Kayal Manindra Agrawal and Nitin Saxena. Primes is in p. *Annals of Mathematics*, 160:781–793, 2004.

[9] Haiko Muller and Andreas Brandstadt. The np-completeness of steiner tree and dominating set for chordal bipartite graphs. *Theoretical Computer Science*, 53:257–265.

[10] Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete Comput Geom*, 44:883–895, 2010.