Integer and Combinatorial Optimization: NP-Completeness

John E. Mitchell

Department of Mathematical Sciences
RPI, Troy, NY 12180 USA

January 2019
Outline

1. The class \( \mathcal{NP} \)
2. \( \mathcal{NP} \)-Complete problems
3. Gadgets
Certificate of Feasibility

Given an instance $d \in D$ of a feasibility problem $X$, a certificate of feasibility $Q_d$ is information that can be used to check feasibility in polynomial time. Note that the length of $Q_d$ must be polynomial in the length of the data.

For example, a certificate of feasibility for an instance of the perfect matching problem would be a list of edges in the matching.

**Definition**

A feasibility problem $X = (D, F)$ is in $\mathcal{NP}$ if there exists a certificate of feasibility for any $d \in F$ that can be checked in polynomial time.

Note that $\mathcal{P} \subseteq \mathcal{NP}$, since the algorithm for solving the problem can used as a certificate.
Is \( P = NP? \)

This is the fundamental question in computational complexity. It is generally accepted that \( P \neq NP \), but not yet proven. This is one of the Clay Millennium Problems, http://www.claymath.org/millennium-problems/p-vs-np-problem

Why study problems in \( NP? \)

_Papadimitriou and Steiglitz, page 351:_ “The recognition versions of all reasonable combinatorial optimization problems are in \( NP \), \( \ldots \) (because) \( \ldots \) combinatorial optimization problems aim for the optimal design of objects. It is reasonable to expect that, once found, the optimal solution can be written down concisely, and thus serve as a certificate for the recognition version.”
A problem not in $\mathcal{NP}$

**Solution of quadratic inequalities:**

Given symmetric $n \times n$ matrices $Q_1, \ldots, Q_m$, and scalars $b_1, \ldots, b_m$, does there exist $x \in \mathbb{R}^n$ satisfying $x^T Q_i x \leq b_i$ for $i = 1, \ldots, m$?

Eg, the only solutions to $x^2 \leq 2, -x^2 \leq -2$ are $x = \pm \sqrt{2}$, neither of which can be written down in polynomial time.

If ask for a *rational* $x$ then the problem is in $\mathcal{NP}$. 
Outline

1. The class $\mathcal{NP}$

2. $\mathcal{NP}$-Complete problems

3. Gadgets
The hardest problems in $\mathcal{NP}$ are the $\mathcal{NP}$-complete problems. We will set up some definitions leading towards the characterization that

*If an $\mathcal{NP}$-complete problem can be solved in polynomial time then any problem in $\mathcal{NP}$ can be solved in polynomial time.*
Polynomial transformations

Definition

Let $X_1 = (D_1, F_1)$ and $X_2 = (D_2, F_2)$ be two feasibility problems in $\mathcal{NP}$. Assume there exists a function $g : D_1 \rightarrow D_2$ such that $d_1 \in F_1$ if and only if $g(d_1) \in F_2$, for any instance $d_1 \in D_1$.

If $g$ is computable in time that is polynomial in the length of the encoding of $d_1$ then $X_1$ is polynomially transformable to $X_2$. 
SAT

Definition

A **Boolean variable** $y$ is a variable that can assume only the values true or false. Boolean variables can be combined to form Boolean formulas using “negation” $\bar{y}$, “or” $y_1 + y_2$, “and” $y_1 \cdot y_2$. A **clause** is a formula containing some of the original variables together with only or and negation.

An instance of the **Satisfiability problem (SAT)** consists of a collection of clauses $C_1, \ldots, C_m$ and asks if there is a truth assignment to the Boolean variables so that all the clauses can be satisfied simultaneously.

For example:

- The formula $(y_1 + y_2) \cdot (\bar{y}_1)$ can be satisfied: set $y_1 = \text{FALSE}$, $y_2 = \text{TRUE}$.
- The formula $(y_1 + \bar{y}_2) \cdot (\bar{y}_1) \cdot (y_2)$ cannot be satisfied.
SAT

**Definition**

A **Boolean variable** $y$ is a variable that can assume only the values true or false. Boolean variables can be combined to form Boolean formulas using “negation” $\bar{y}$, “or” $y_1 + y_2$, “and” $y_1 \cdot y_2$. A **clause** is a formula containing some of the original variables together with only or and negation.

An instance of the **Satisfiability problem (SAT)** consists of a collection of clauses $C_1, \ldots, C_m$ and asks if there is a truth assignment to the Boolean variables so that all the clauses can be satisfied simultaneously.

For example:

- The formula $(y_1 + y_2). (\bar{y}_1)$ can be satisfied: set $y_1 =$FALSE, $y_2 =$TRUE.
- The formula $(y_1 + \bar{y}_2). (\bar{y}_1). (y_2)$ cannot be satisfied.
SAT

Definition

A **Boolean variable** \( y \) is a variable that can assume only the values true or false. Boolean variables can be combined to form Boolean formulas using “negation” \( \bar{y} \), “or” \( y_1 + y_2 \), “and” \( y_1 \cdot y_2 \). A **clause** is a formula containing some of the original variables together with only or and negation.

An instance of the **Satisfiability problem (SAT)** consists of a collection of clauses \( C_1, \ldots, C_m \) and asks if there is a truth assignment to the Boolean variables so that all the clauses can be satisfied simultaneously.

For example:

- The formula \((y_1 + y_2) . (\bar{y}_1)\) can be satisfied: set \( y_1 = \text{FALSE}, y_2 = \text{TRUE} \).
- The formula \((y_1 + \bar{y}_2) . (\bar{y}_1) . (y_2)\) cannot be satisfied.
An example of a polynomial transformation

**SAT** can be polynomially transformed into 0-1 linear feasibility:

- Given an instance of **SAT** with \( n \) Boolean variables \( y_1, \ldots, y_n \), and \( m \) clauses \( C_1, \ldots, C_m \).
- Define \( n \) binary variables \( x_1, \ldots, x_n \), with the intended mapping
  
  \[
  \begin{align*}
  x_j = 1 & \iff y_j = \text{TRUE} \\
  x_j = 0 & \iff y_j = \text{FALSE}
  \end{align*}
  \]
  \[\text{for } j = 1, \ldots, n\]

- For each clause \( C_i \), define a constraint:
  \[
  \sum_{j: y_j \in C_i} x_j + \sum_{j: \bar{y}_j \in C_i} (1 - x_j) \geq 1
  \]
An example of a polynomial transformation

\textbf{SAT} can be polynomially transformed into 0-1 linear feasibility:

- Given an instance of \textbf{SAT} with \(n\) Boolean variables \(y_1, \ldots, y_n\), and \(m\) clauses \(C_1, \ldots, C_m\).
- Define \(n\) binary variables \(x_1, \ldots, x_n\), with the intended mapping
  \[
  x_j = 1 \iff y_j = \text{TRUE} \\
  x_j = 0 \iff y_j = \text{FALSE}
  \]
  for \(j = 1, \ldots, n\)

- For each clause \(C_i\), define a constraint:
  \[
  \sum_{j:y_j \in C_i} x_j + \sum_{j:\bar{y}_j \in C_i} (1 - x_j) \geq 1
  \]
An example of a polynomial transformation

**SAT** can be polynomially transformed into 0-1 linear feasibility:

- Given an instance of **SAT** with $n$ Boolean variables $y_1, \ldots, y_n$, and $m$ clauses $C_1, \ldots, C_m$.
- Define $n$ binary variables $x_1, \ldots, x_n$, with the intended mapping
  \[
  x_j = 1 \iff y_j = \text{TRUE} \\
  x_j = 0 \iff y_j = \text{FALSE}
  \]
  for $j = 1, \ldots, n$
- For each clause $C_i$, define a constraint:
  \[
  \sum_{j: y_j \in C_i} x_j + \sum_{j: \bar{y}_j \in C_i} (1 - x_j) \geq 1
  \]
An example of a polynomial transformation (page 2)

- For example, the clause $y_1 + \bar{y}_3 + y_7$ maps to the constraint

$$x_1 + (1 - x_3) + x_7 \geq 1$$

The constraint holds if $x_1 = 1$, which corresponds to $y_1 = \text{TRUE}$, in which case the clause is satisfied. Similarly, the constraint holds if $x_3 = 0$, corresponding to $y_3 = \text{FALSE}$, when the clause is satisfied.

The only binary assignment which violates the constraint is $x_1 = 0, x_3 = 1, x_7 = 0$, which corresponds to $y_1 = \text{FALSE}, y_3 = \text{TRUE}, y_7 = \text{FALSE}$, which is the only truth assignment that violates the constraint.

- It follows that there is a truth assignment to $y_1, \ldots, y_n$ that satisfies all of $C_1, \ldots, C_m$ if and only if there is a binary solution to all of the linear inequalities.
An example of a polynomial transformation (page 2)

- For example, the clause $y_1 + \bar{y}_3 + y_7$ maps to the constraint

$$x_1 + (1 - x_3) + x_7 \geq 1$$

The constraint holds if $x_1 = 1$, which corresponds to $y_1 = \text{TRUE}$, in which case the clause is satisfied. Similarly, the constraint holds if $x_3 = 0$, corresponding to $y_3 = \text{FALSE}$, when the clause is satisfied.

The only binary assignment which violates the constraint is $x_1 = 0, x_3 = 1, x_7 = 0$, which corresponds to $y_1 = \text{FALSE}, y_3 = \text{TRUE}, y_7 = \text{FALSE}$, which is the only truth assignment that violates the constraint.

- It follows that there is a truth assignment to $y_1, \ldots, y_n$ that satisfies all of $C_1, \ldots, C_m$ if and only if there is a binary solution to all of the linear inequalities.
Polynomial reductions

Proposition

If $X_1$ is polynomially transformable to $X_2$ and if $X_2 \in \mathcal{P}$ then $X_1 \in \mathcal{P}$.

A polynomial transformation is a “one-shot” modification of the original problem. We can also think of using problem $X_2$ as a “subroutine”, so we solve multiple instances of problem $X_2$ in order to solve an instance of $X_1$.

Definition

Let $X_1 = (D_1, F_1)$ and $X_2 = (D_2, F_2)$ be two feasibility problems in $\mathcal{NP}$. The problem $X_1$ is polynomially reducible to $X_2$ if there exists an algorithm $A_1$ for $X_1$ that uses an algorithm $A_2$ for $X_2$ as a subroutine, and $A_1$ runs in polynomial time under the assumption that each call of the subroutine takes unit time.
Proposition

If $X_1$ is polynomially transformable to $X_2$ and if $X_2 \in \mathcal{P}$ then $X_1 \in \mathcal{P}$.

A polynomial transformation is a “one-shot” modification of the original problem. We can also think of using problem $X_2$ as a “subroutine”, so we solve multiple instances of problem $X_2$ in order to solve an instance of $X_1$.

Definition

Let $X_1 = (D_1, F_1)$ and $X_2 = (D_2, F_2)$ be two feasibility problems in $\mathcal{NP}$. The problem $X_1$ is polynomially reducible to $X_2$ if there exists an algorithm $A_1$ for $X_1$ that uses an algorithm $A_2$ for $X_2$ as a subroutine, and $A_1$ runs in polynomial time under the assumption that each call of the subroutine takes unit time.
More on polynomial reductions

For example, an instance of *Maximum Weight Matching with lower bound* can be solved by solving a polynomial number of *Maximum Cardinality Matching with lower bound* instances (see Chapter III.2 of the text).

Hence, *Maximum Weight Matching with lower bound* is polynomially reducible to *Maximum Cardinality Matching with lower bound*.

**Proposition**

*If* $X_1$ *is polynomially reducible to* $X_2$ *and if* $X_2 \in \mathcal{P}$ *then* $X_1 \in \mathcal{P}$. *

Polynomial transformation is a special case of polynomial reduction.

**Definition**

A feasibility problem $X \in \mathcal{NP}$ is \textit{\textbf{NP}-complete} if all other problems in \textit{\textbf{NP}} polynomially reduce to $X$. 
More on polynomial reductions

For example, an instance of *Maximum Weight Matching with lower bound* can be solved by solving a polynomial number of *Maximum Cardinality Matching with lower bound* instances (see Chapter III.2 of the text).

Hence, *Maximum Weight Matching with lower bound* is polynomially reducible to *Maximum Cardinality Matching with lower bound*.

**Proposition**

If $X_1$ is polynomially reducible to $X_2$ and if $X_2 \in \mathcal{P}$ then $X_1 \in \mathcal{P}$.

Polynomial transformation is a special case of polynomial reduction.

**Definition**

A feasibility problem $X \in \mathcal{NP}$ is \(\mathcal{NP}\)-complete if all other problems in \(\mathcal{NP}\) polynomially reduce to $X$. 
Classes of problems

\[ \mathcal{NP} \text{-Complete problems} \]

feasibility problems
Proving a feasibility problem $X$ is $NP$-complete

- First, need to verify the problem is in $NP$.
- Then, need to show that a known $NP$-complete problem can be polynomially reduced to $X$.

Note the direction!!

Why this direction?

We know every problem in $NP$ can be polynomially reduced to our known $NP$-complete problem, which in turn can be polynomially reduced to $X$.

Hence, every problem in $NP$ can be polynomially reduced to $X$.

any problem in $NP$ $\longrightarrow$ known $NP$-complete problem $\longrightarrow$ $X$
Proving a feasibility problem $X$ is $NP$-complete

- First, need to verify the problem is in $NP$.
- Then, need to show that a known $NP$-complete problem can be polynomially reduced to $X$.

Note the direction!!

Why this direction?

We know every problem in $NP$ can be polynomially reduced to our known $NP$-complete problem, which in turn can be polynomially reduced to $X$.

Hence, every problem in $NP$ can be polynomially reduced to $X$.

any problem in $NP$ $\rightarrow$ known $NP$-complete problem $\rightarrow$ $X$
Proving a feasibility problem $X$ is $\mathcal{NP}$-complete

- First, need to verify the problem is in $\mathcal{NP}$.
- Then, need to show that a known $\mathcal{NP}$-complete problem can be **polynomially reduced to** $X$.

Note the direction!!

Why this direction?

We know every problem in $\mathcal{NP}$ can be polynomially reduced to our known $\mathcal{NP}$-complete problem, which in turn can be polynomially reduced to $X$.

Hence, every problem in $\mathcal{NP}$ can be polynomially reduced to $X$.

\[
\text{any problem in } \mathcal{NP} \rightarrow \text{known } \mathcal{NP}-\text{complete problem} \rightarrow X
\]
Proving a feasibility problem $X$ is $NP$-complete

- First, need to verify the problem is in $NP$.
- Then, need to show that a known $NP$-complete problem can be polynomially reduced to $X$.

Note the direction!!

Why this direction?

We know every problem in $NP$ can be polynomially reduced to our known $NP$-complete problem, which in turn can be polynomially reduced to $X$.

Hence, every problem in $NP$ can be polynomially reduced to $X$.

\[
\text{any problem in } NP \rightarrow \text{known } NP\text{-complete problem} \rightarrow X
\]
Proving a feasibility problem $X$ is $\mathcal{NP}$-complete

- First, need to verify the problem is in $\mathcal{NP}$.
- Then, need to show that a known $\mathcal{NP}$-complete problem can be polynomially reduced to $X$.

Note the direction!!

Why this direction?

We know every problem in $\mathcal{NP}$ can be polynomially reduced to our known $\mathcal{NP}$-complete problem, which in turn can be polynomially reduced to $X$.

Hence, every problem in $\mathcal{NP}$ can be polynomially reduced to $X$.

any problem in $\mathcal{NP} \rightarrow$ known $\mathcal{NP}$-complete problem $\rightarrow X$
SAT and $\mathcal{NP}$-completeness

The first problem to be shown to be $\mathcal{NP}$-complete was SAT, by S. Cook in 1971.

Since SAT is polynomially transformable to 0-1 linear feasibility, the problem 0-1 linear feasibility must also be $\mathcal{NP}$-complete.

The problem 3-SAT is also $\mathcal{NP}$-complete. In this problem, each clause has at most 3 literals.

In the problem 2-SAT, each clause has at most 2 literals. This problem can be solved in polynomial time.

In weighted 2-SAT with lower bound, each clause has a weight, and we have a target total weight $M$. We desire a truth assignment where the total weight of the satisfied clauses is at least $M$. Weighted 2-SAT with lower bound is $\mathcal{NP}$-complete.
Outline

1. The class $\mathcal{NP}$

2. $\mathcal{NP}$-Complete problems

3. Gadgets
Proving other problems are $\mathcal{NP}$-complete

A “gadget” is a structure that is constructed when reducing one problem to another.

For example, in the reduction of 3-SAT to node packing, a clique is constructed from each clause.

And in the reduction of 3-SAT to Hamiltonian cycle, a subgraph is constructed for each clause; the subgraph can be traversed in multiple different ways, corresponding to different ways to satisfy the clause; but it cannot be traversed in a way that corresponds to the clause being False.

In a reduction from 3-SAT, each clause is typically used to construct some structure, and then the structures are linked together somehow so as to enforce consistency between the truth assignments used for each different clause and its corresponding structure. These methods to enforce consistency can also be regarded as gadgets.