COMP 382: Reasoning about algorithms **Fall 2025**

December 4, 2025 *Instructors: Aliakbarpour, Chida, Mamouras*

Lab Worksheet 8: NP-Completeness

Subset Sum

We are given a multiset of integers $Z = \{z_1, \ldots, z_n\}$ and a target $T \in \mathbb{Z}$. The question is whether there exists an index set $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} z_i = T$. Despite its simple statement, this problem is computationally intractable in general.

Proof of NP-completeness

Our goal is to prove that Subset Sum is NP-complete. The proof has two steps.

1. Show that Subset Sum is in NP.

2. Show Subset Sum is NP-hard via a polynomial-time reduction from the NP-complete problem 3-SAT.

3-SAT Problem. Recal from lecture: Given a 3-CNF formula with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$, each clause having exactly three literals, decide whether there exists a truth assignment satisfying all clauses.

Connecting 3-SAT and Subset Sum. The strength of Subset Sum is that many different constraints can be encoded simultaneously by using *different digit positions* in a large-base number system. Each digit represents one constraint.

For example, suppose we include in our multiset two numbers $z_1 = 1000$ and $z_2 = 1000$, and assume all other z_i are either at least 10000 or together sum to less than 1000. If we make the thousands digit of the target T equal to exactly 1000, then any solution I to the Subset Sum instance must include **exactly one** of z_1 or z_2: including none gives 0 in that digit; including both gives 2000. This ability to force "choose exactly one" is precisely what allows us to reduce 3-SAT to Subset Sum. We can use this structure to ensure literals are selected consistently, meaning that x_i and $\neg x_i$ are in fact not equal.

A second key idea is that we can also allow *multiple acceptable values* in a digit, by adding *slack numbers*. For instance, if a certain digit of the target is 3, and our construction allows a chosen subset to contribute 1, 2, or 3 from the "meaningful" numbers, we can add two slack numbers that each contribute 1 in that digit so that 1, 2, or 3 can all be "completed" up to the target value 3. This flexibility lets us express constraints of the form "at least one of these contributions must be present", because although slack numbers can fill in missing contributions, they can only fill in a limited amount. In the 3-SAT reduction, this is exactly how we ensure that every clause has at least one satisfied literal: each clause digit can be brought up to its target value using at most two slack numbers, so the chosen subset must provide at least one literal-contribution.

These two mechanisms are what make it possible to reduce 3-SAT to Subset Sum.

Construction for the 3-SAT to Subset Sum Reduction

We now outline the structure of the reduction and ask you to complete the missing pieces. Consider a 3-SAT instance with literals$\{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$ and clauses $C_1, \ldots, C_m$. We will construct a Subset Sum instance in base $B = 10$.[1]

Digits. Each constructed number will have exactly $n + m$ digits. Each digit encodes one constraint:

- **Variable digits**: the first n digits ensure that for every variable x_i, we choose *exactly one* of x_i or $\neg x_i$ to be true.
- **Clause digits**: the last m digits ensure that each clause has *at least one satisfied literal.*

Numbers. For each literal ℓ, we construct a number $z(\ell)$. Denote

$$z_{2i} := z(x_i), \qquad z_{2i-1} := z(\neg x_i).$$

For each clause C_j, we also create two slack numbers $s_{j,1}$ and $s_{j,2}$. These allow clause digits to reach their target value even when the selected literals contribute less than the target.

[1]A larger base works as well; the crucial requirement is that no carries occur.

Interpretation. We construct a target number T with $n + m$ digits. A subset I is a solution if and only if

$$\sum_{k \in I} z_k = T.$$

If $z(\ell)$ is selected in the sum (its index is in I), then we interpret ℓ as being set to TRUE; otherwise it is FALSE.

Part 1: Enforcing the variable constraint (digit i).

For each $i \in \{1, \dots, n\}$, digit i enforces that: "Exactly one of $z(x_i)$ or $z(\neg x_i)$ is selected". This implies that between x_i and $\neg x_i$ one of them is TRUE and the other one is FALSE.

To enforce this, fill in the i-th digit entries below.

Hint: Try assigning values to z_{2i} and z_{2i-1}. What happens if the subset includes: neither, exactly one, or both? Which of these should match the target digit?

Number	**Digit i**
$z(x_i)$	☐
$z(\neg x_i)$	☐
Any other number	☐
Target T	☐

Part 2: Enforcing the clause constraint (digit $n + j$).

For each clause $C_j = (\ell_1 \vee \ell_2 \vee \ell_3)$, digit $(n+j)$ ensures: "At least one literal in C_j is selected." Each literal ℓ_r that appears in the clause contributes to digit $(n + j)$. Complete the table:

Hint: Set the i-th digit of T such that selecting **no true literal** makes it impossible to hit the target digit. Using the slack variables if any of the literals are True, it should be possible to hit the target digit.

Number	**Digit $n + j$ (clause digit)**
$z(\ell_1)$	☐
$z(\ell_2)$	☐
$z(\ell_3)$	☐
$s_{j,1}$ (slack 1)	☐
$s_{j,2}$ (slack 2)	☐
Any other number	☐
Target T	☐

Mini Example: Fill in the literal numbers and the target. We use base $B = 10$ with $n = 2$ variables (x_1, x_2) and a single clause $C_1 = (x_1 \lor x_2 \lor \neg x_2)$. There are $n + m = 3$ digits per number: $(d_1,\ d_2 \mid d_3)$, where d_1 enforces "exactly one of x_1 or $\neg x_1$," d_2 enforces "exactly one of x_2 or $\neg x_2$," and d_3 enforces that C_1 is satisfied (with two slack numbers $s_{1,1}, s_{1,2}$).

Number	d_1	d_2	d_3 (clause C_1)
$z(x_1)$	$\square$	$\square$	$\square$
$z(\neg x_1)$	$\square$	$\square$	$\square$
$z(x_2)$	$\square$	$\square$	$\square$
$z(\neg x_2)$	$\square$	$\square$	$\square$
$s_{1,1}$	$\square$	$\square$	$\square$
$s_{1,2}$	$\square$	$\square$	$\square$
Target T	$\square$	$\square$	$\square$

Given one satisfying assignment here and the corresponding subset-sum solution I:

Assignment: $x_1 = \square$, $x_2 = \square$, $I = \boxed{}$

Proof of correctness

Next we focus on the proof of correctness. We show a one-to-one correspondence between satisfying assignments of the 3-SAT instance and solutions to the constructed SUBSET SUM instance.

($\Rightarrow$ direction) Assume the 3-SAT instance is a YES instance; that is, there exists a truth assignment that satisfies all clauses. Show that the constructed Subset Sum instance has a subset I whose numbers sum exactly to the target T.

($\Leftarrow$ direction) Assume the constructed Subset Sum instance admits a subset I such that $\sum_{k \in I} z_k = T$. Prove that the 3-SAT instance is a YES instance.

Conclusion. You have shown that a satisfying assignment of 3-SAT corresponds exactly to a subset summing to the target T, and vice versa. Thus, 3-SAT reduces to Subset Sum, proving the problem is NP-hard. Since Subset Sum is also in NP, it is NP-complete.

Job Scheduling on Identical Machines

You run a small compute cluster with two identical machines. A queue of jobs $J = \{1, \ldots, n\}$ has processing times $p_1, \ldots, p_n$. You want to assign each job to one of the two machines so that the *finishing time* (the time when the last machine stops) is as small as possible. This objective is called the *makespan*. Even this simple-looking problem hides computational hardness.

Decision version of two-machine job scheduling We are given positive integers $p_1, \ldots, p_n$, and a target makespan $K \in \mathbb{Z}_{>0}$. Can we schedule the jobs on two identical machines so that the makespan is at most K? Equivalently, is there a partition of the jobs into two sets A and B such that

$$\sum_{j \in A} p_j \leq K \quad \text{and} \quad \sum_{j \in B} p_j \leq K?$$

Proof of NP-completeness

Our goal is to prove that the decision version of two-machine job scheduling is NP-complete. The proof has two steps.

1. Show that job scheduling on two machines is in NP.

2. The problem is NP-hard via a polynomial-time reduction from the NP-complete problem Partition.

Partition Problem. Given positive integers $a_1, \ldots, a_n$ with total sum $S = \sum_i a_i$, decide whether there exists an index set $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = S/2$. For the purpose of this worksheet, we assume this problem is NP-complete. This can be proved by showing a polynomial reduction from Subset-Sum.

Construct the reduction. Given an instance $(a_1, \ldots, a_n)$ of Partition, define the corresponding scheduling instance $(p_1, \ldots, p_n; K)$ for two machines. Specify each p_j and K. Briefly argue that mapping is computable in polynomial time.

Proof of correctness.

Next we focus on the proof of correctness. We show a one and one correspondence between the solutions of these two problems.

($\Rightarrow$ direction) Assume the PARTITION instance is a YES instance; that is, there exists I with $\sum_{i \in I} a_i = S/2$. Show that the constructed scheduling instance has a schedule of makespan $\leq K$.

($\Leftarrow$ direction). Assume the scheduling instance admits a schedule with makespan $\leq K$. Prove that the PARTITION instance is a YES instance.

Conclusion. You have shown an NP-complete problem (PARTITION) can be reduced to the decision version of job scheduling. Hence, job scheduling is NP-hard. Since the problem is also in NP, it is NP-complete.