Computational Complexity

All of the scheduling problems can be solved in a finite amount of time, since there are only a finite number of schedules. However, the running time could be huge.

Example: PARTITION problem

Given: A list A = (a₁, a₂, …, aₙ) of n integers

Question: Can A be partitioned into A₁ and A₂ such that

\[ \sum_{a_j \in A_1} a_j = \sum_{a_j \in A_2} a_j = \frac{1}{2} \sum_{i = 1}^{n} a_i \]

An instance A = (2, 4, 7, 10) n = 4

Answer = “No”

A = (3, 9, 9, 20, 21, 22) n = 6

A₁ = (20, 22)

A₂ = (3, 9, 9, 21)

Answer = “Yes”

One simple approach to solve this problem: Try all possible subsets

O(2ⁿ) time
If \( n = 100 \)
\[
2^{100} = 2^{10}, 2^{10} \ldots 2^{10}
\]
10 times

\[
\approx 10^3 \times 10^3 \times \ldots \times 10^3
\]
10 times

\[
= 10^{30}
\]

A fast computer runs approximately 1 trillion (\(10^{12}\)) instruction per second

Therefore, it takes \(10^{30} / 10^{12}\) seconds to run

\[
= 10^{18} \text{ seconds}
\]

\[
= 10^{18} / 3600 \text{ hours}
\]

\[
= 10^{18} / 3600 \times 24 \times 365 \text{ years}
\]

\[
= 10^{18} / 10^8 \text{ years}
\]

\[
= 10^{10} \text{ years}
\]

\[
= 10 \text{ billion years}
\]

IMPRACTICAL!
Def: A problem is a general question to be answered, with several parameters whose values are left unspecified.

A problem is described by giving:

1. A general description of all its parameters.
2. A statement of what properties the answer, or solution, is required to satisfy

An instance of a problem is obtained by specifying particular values for all problem parameters.

Ex: Hamilton Cycle Problem

Given: An undirected graph \( G = (V, E) \)

Question: Does \( G \) have a Hamilton cycle; i.e. a cycle that goes through each vertex exactly once

Instance \( G: \)

\[ \text{Answer = “Yes”} \]

\[ \text{Decision Problem: has only “Yes” or “No” answer} \]
Ex: Traveling Salesman (Optimization) problem TSO

Given: A set $C = \{C_1, C_2, \ldots, C_m\}$ of $m$ cities.

For each pair of cities $C_i$ and $C_j$ in $C$, the distance $d(C_i, C_j)$ between them

Question: Find an ordering $<C_{i_1}, C_{i_2}, \ldots, C_{i_m}>$ of the $m$ cities such that

$$\sum_{j=1}^{m-1} d(C_{i_j}, C_{i_{j+1}}) + d(C_{i_m}, C_{i_1})$$

is minimum.

Instance

10 9 3
5 9

$$(C_1, C_2, C_4, C_3)$$ is a solution

10 9 3
5

= 27

This is the minimum

[So it is an answer of the problem]
Ex: Traveling Salesman (Decision) problem **TSD**

Given: - A set \( C = \{C_1, C_2, \ldots, C_m\} \) of \( m \) cities.

For each pair of cities \( C_i \) and \( C_j \), the distance \( d(C_i, C_j) \) between them

- A bound \( B \)

Question: Is there an ordering \(<C_{i_1}, C_{i_2}, \ldots, C_{i_m}>\) of the \( m \) cities such that

\[
\sum_{j=1}^{m-1} d(C_{i_j}, C_{i_{j+1}}) + d(C_{i_m}, C_{i_1}) \leq B
\]

Instance

\[
\begin{array}{c}
\text{C}_1 \\
\text{C}_2 \\
\text{C}_3 \\
\text{C}_4 \\
\end{array}
\]

\[
\begin{array}{cccc}
10 & 5 & 9 & \\
6 & & & 3 \\
9 & & & \\
\end{array}
\]

\( B = 30: \)
Answer = “yes”

\( B = 24: \)
Answer = “No”
The time complexity of Traveling Salesman (Optimization) problem is related to Traveling Salesman (Decision) problem.

If there is an efficient algorithm to solve TSO, then we can solve TSD efficiently.

If there is an efficient algorithm to solve TSD, we can solve TSO efficiently.

Conduct a binary search between a lower bound LB and an upper bound UB for the optimal cost. Using the algorithm for TSD to guide the search.

The time requirement of an algorithm is expressed in terms of the “size” of the problem instance, n, which is the number of bits (symbols) necessary to represent the problem instance.

For Example:

\[ T(n) = 3n^2 + n + \log n \]

\[ T(n) = O(n^2) \]

Big “Oh” - dominating term

Polynomial time \( \Rightarrow \) tractable

Exponential time \( \Rightarrow \) intractable
Polynomial Reduction

A problem $P$ is polynomially reducible to another problem $Q$ if there is a transformation $f$ that maps every instance $x$ of $P$ into an instance $y$ of $Q$ such that

1. $x$ has answer “yes” iff $y$ has answer “yes”,
2. $f$ can be done in polynomial time

and the transformation can be done in polynomial time.

Ex. 1: Hamiltonian Circuit problem is polynomially reducible to Traveling Salesman Decision problem

Given an instance $G=(V,E)$ of the Hamilton Circuit problem

Let $V = \{V_1, V_2, \ldots, V_n\}$

$E = \{e_1, e_2, \ldots, e_m\}$
Construct an instance of TSD as follows:

Create n cities c1, c2, …, cn

\[
\begin{cases} 
1 & \text{if } (v_i, v_j) \in G \\
2 & \text{if } (v_i, v_j) \notin L
\end{cases}
\]

Let \( d(c_i,c_j) = \)

Let \( B = n \)

Clearly the transformation can be done in polynomial time

“⇒” (if)

Suppose \( V_1, V_2, …, V_n \) is a Hamilton Circuit then \((C_1, C_2, …, C_n)\) is an ordering of the cities such that total cost \( = B = n \).

“⇐” (only if)

Conversely if \((C_1, C_2, …, C_n)\) is tour \( \leq B \), then \( V_1, V_2, …, V_n \) is a Hamilton Circuit.

Ex. 2: Partition is reducible to \( P_2 || C_{\text{max}} \) (Decision Version)

Given an instance \( A = (a_1, a_2, …, a_n) \) of the partition problem, construct an instance of \( P2 || C_{\text{max}} \) problem as follows

Create \( n \) jobs; 1, 2, …, n. Job j has a processing time \( p_j = a_j \),

\[ B = \frac{1}{2} \sum_{i=1}^{n} a_i \]

Partition has answer “yes” iff \( P_2 || C_{\text{max}} \) (Decision) has answer “yes”

Proof: Leave it as an exercise.
Ex. 3: Knapsack

Given: A set $U = \{u_1, u_2, \ldots, u_n\}$ of $n$ items, with each item $u_i$

having a size $s_i$ and a value $v_i$

- A knapsack size $S$
- A value goal $V$

Question: Is there a subset $U' \subseteq U$ such that

$$\Sigma_{(u_i \in U')} s_i \leq S \quad \text{and} \quad \Sigma_{(u_i \in U')} v_i \geq V?$$

Theorem: Partition is reducible to Knapsack

Given an instance $A = (a_1, a_2, \ldots, a_n)$ of the partition problem, construct an instance of the Knapsack problem as follows:

Create $n$ items $U = \{u_1, u_2, \ldots, u_n\}$, with size $s_i = a_i$ and $v_j = a_i$.

Let $S = \frac{1}{2} \Sigma_{(i=1 \to n)} a_i$ and $V = \frac{1}{2} \Sigma_{(i=1 \to n)} a_i$.

Construction can be done in polynomial time.

$\Rightarrow$ (if)

Suppose $A_1$ and $A_2$ constitute a solution to the instance of partition. Then the set $U' = \{u_i \mid a_i \in A_1\}$ is a solution to the Knapsack since $\Sigma_{(a_i \in U')} s_i = S$ and $\Sigma_{(a_i \in U')} v_i = V$.

$\Leftarrow$ (only if)

Conversely, suppose $U' = \{u_i \mid a_i \in A_1\}$ is a solution to the Knapsack such that $\Sigma_{(a_i \in U')} s_i = S$ and $\Sigma_{(a_i \in U')} v_i = V$. Then $A_1 = \{a_i \mid u_i \in U'\}$ and $A_2 = \{a_i \mid u_i \not\in U'\}$ constitute a solution to partition.
NP is the class of decision problems that can be solved by a polynomial time algorithm.
(Or there is a polynomial time Deterministic Turing Machine program that solves it.)

A problem \( Q \) is NP-complete if

1. \( Q \) is in NP
2. Every problem in NP is reducible to \( Q \)

Notes

1. Reducibility is transitive
   
   i.e \( P \preceq Q \) and \( Q \preceq R \Rightarrow P \preceq R \)

2. If \( P \) and \( Q \) are NP-complete, then \( P \preceq Q \) and \( Q \preceq P \)

3. If \( P \) is NP-complete and \( P \) is solvable in polynomial time, then every problem in NP can be solved in polynomial time.

www.mathematik.uni-osnabrueck.de/research/OR/class/ has a comprehensive list of complexity of scheduling problems with references.