# Computability and Complexity Theory

## Computability and complexity

- *Computability theory*
  - What problems can be solved on a computer ?
  - What is a computable function ?
  - Decidable vs. undecidable problems
- *Complexity theory*
  - How much time and memory is needed to solve a problem ?
  - Tractable vs. intractable problems

## What is a computable function ?

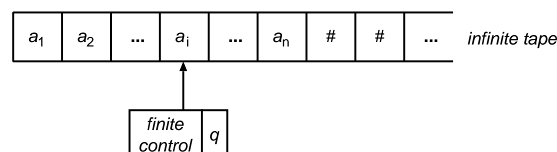
- Non-trivial question $\rightsquigarrow$ various formalizations, e.g.
  - General recursive functions *Gödel/Herbrand/Kleene 1936*
  - $\lambda$-calculus *Church 1936*
  - $\mu$-recursive functions *Gödel/Kleene 1936*
  - Turing machines *Turing 1936*
  - Post systems *Post 1943*
  - Markov algorithms *Markov 1951*
  - Unlimited register machines *Shepherdson-Sturgis 1963*

  ...
- All these approaches have turned out to be equivalent.

## Church's thesis

The class of intuitively computable functions is equal to the class of Turing computable functions.

## Turing machine




Depending on the symbol scanned and the state of the control, in each step the machine

- changes state,
- prints a symbol on the cell scanned, replacing what is written there,
- moves the head left or right one cell.

# Formal definition

- $M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$
- $Q$ is the finite set of *states.*
- $\Gamma$ is the finite alphabet of allowable *tape symbols.*
- $\# \in \Gamma$ is the *blank.*
- $\Sigma \subset \Gamma \setminus \{\#\}$ is the set of *input symbols.*
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the *next move function* (possibly undefined for some arguments)
- $q_0 \in Q$ is the *start state.*
- $F \subseteq Q$ is the set of *final (accepting) states.*

# Recognizing languages

- *Instantaneous description:* $\alpha_l\, q\, \alpha_r$, where
  - $q$ is the current state,
  - $\alpha_l \alpha_r \in \Gamma^*$ is the string on the tape up to the rightmost nonblank symbol,
  - the head is scanning the leftmost symbol of $\alpha_r$.
- *Move:* $\alpha_l\, q\, \alpha_r \vdash \alpha_l'\, q'\, \alpha_r'$, by one step of the machine.
- *Language accepted*

$$L(M) = \{w \in \Sigma^* \mid q_0\, w \vdash^* \alpha_l q \alpha_r,\ \text{for some } q \in F \text{ and } \alpha_l, \alpha_r \in \Gamma^*\}$$

- $M$ may not halt, if $w$ is not accepted.

# Example

- *Turing machine*

$$M = (\{q_0, \dots, q_4\}, \{0, 1\}, \{0, 1, X, Y, \#\}, \delta, q_0, \#, \{q_4\})$$

  accepting the language $L = \{0^n 1^n \mid n \geq 1\}$

| $\delta$ | 0 | 1 | $X$ | $Y$ | # |
|---|---|---|---|---|---|
| $q_0$ | $(q_1, X, R)$ | – | – | $(q_3, Y, R)$ | – |
| $q_1$ | $(q_1, 0, R)$ | $(q_2, Y, L)$ | – | $(q_1, Y, R)$ | – |
| $q_2$ | $(q_2, 0, L)$ | – | $(q_0, X, R)$ | $(q_2, Y, L)$ | – |
| $q_3$ | – | – | – | $(q_3, Y, R)$ | $(q_4, \#, R)$ |
| $q_4$ | – | – | – | – | – |

- *Example computation*

$$\begin{array}{llllllll} q_0 0011 & \vdash & Xq_1 011 & \vdash & X0q_1 11 & \vdash & Xq_2 0Y1 & \vdash \\ q_2 X0Y1 & \vdash & Xq_0 0Y1 & \vdash & XXq_1 Y1 & \vdash & XXYq_1 1 & \vdash \\ XXq_2 YY & \vdash & Xq_2 XYY & \vdash & XXq_0 YY & \vdash & XXYq_3 Y & \vdash \\ XXYYq_3 & \vdash & XXYY\#q_4 & & & & & \end{array}$$

# Recursive languages

- A language $L \subseteq \Sigma^*$ is *recursively enumerable* if $L = L(M)$, for some Turing machine $M$.

$$w \longrightarrow \boxed{\text{M}} \longrightarrow \begin{cases} \text{yes,} & \text{if } w \in L \\ \text{no,} & \text{if } w \notin L \\ M \text{ does not halt,} & \text{if } w \notin L \end{cases}$$

- A language $L \subseteq \Sigma^*$ is *recursive* if $L = L(M)$ for some Turing machine $M$ that halts on all inputs $w \in \Sigma^*$.

$$w \longrightarrow \boxed{\text{M}} \longrightarrow \begin{cases} \text{yes,} & \text{if } w \in L \\ \text{no,} & \text{if } w \notin L \end{cases}$$

- **Lemma.** $L$ is recursive iff both $L$ and $\overline{L} = \Sigma^* \setminus L$ are recursively enumerable.

## Enumerating languages

- An *enumerator* is a Turing machine $M$ with extra output tape $T$, where symbols, once written, are never changed.
- $M$ writes to $T$ words from $\Sigma^*$, separated by \$.
- Let $G(M) = \{w \in \Sigma^* \mid w \text{ is written to } T\}$.

## Some results

- **Lemma.** For any finite alphabet $\Sigma$, there exists a Turing machine that generates the words $w \in \Sigma^*$ in *canonical ordering* (i.e., $w \prec w' \Leftrightarrow |w| < |w|$ or $|w| = |w|$ and $w \prec_{lex} w'$).
- **Lemma.** There exists a Turing machine that generates all pairs of natural numbers (in binary encoding).

  *Proof:* Use the ordering $(0,0)$, $(1,0)$, $(0,1)$, $(2,0)$, $(1,1)$, $(0,2)$, ...
- **Proposition.** $L$ is recursively enumerable iff $L = G(M)$, for some Turing machine $M$.

## Computing functions

- Unary encoding of natural numbers: $i \in \mathbb{N} \mapsto \underbrace{||...|}_{i \text{ times}} = |^i$

  (binary encoding would also be possible)
- *$M$ computes $f : \mathbb{N}^k \to \mathbb{N}$* with $f(i_1, \ldots, i_k) = m$:
  - Start: $|^{i_1} 0 |^{i_2} 0 \ldots |^{i_k}$
  - End: $|^m$
- *$f$ partially recursive:*

$$i_1, \ldots, i_k \longrightarrow \boxed{\text{M}} \longrightarrow \begin{cases} \text{halts with } f(i_1, \ldots, i_k) = m, \\ \text{does not halt, i.e., } f \text{ undefined.} \end{cases}$$

- *$f$ recursive:*

$$i_1, \ldots, i_k \longrightarrow \boxed{\text{M}} \longrightarrow \text{halts with } f(i_1, \ldots, i_k) = m.$$

## Turing machines codes

- May assume

$$M = (Q, \{0,1\}, \{0,1,\#\}, \delta, q_1, \#, \{q_2\})$$

- Unary encoding

$$0 \mapsto 0, 1 \mapsto 00, \# \mapsto 000, L \mapsto 0, R \mapsto 00$$

- $\delta(q_i, X) = (q_j, Y, R)$ encoded by

$$0^i 1 \underbrace{0 \ldots 0}_{X} 1 0^j 1 \underbrace{0 \ldots 0}_{Y} 1 \underbrace{0 \ldots 0}_{R}$$

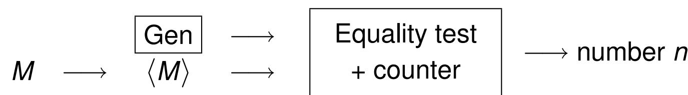
- $\delta$ encoded by

$$111\,\texttt{code}_1\,11\,\texttt{code}_2\,11 \ldots 11\,\texttt{code}_r\,111$$

- Encoding of Turing machine $M$ denoted by $\langle M \rangle$.

## Numbering of Turing machines

- **Lemma.** There exists a Turing machine that generates the natural numbers in binary encoding.
- **Lemma.** There exists a Turing machine *Gen* that generates the binary encodings of all Turing machines.
- **Proposition.** The language of Turing machine codes is recursive.
- **Corollary.** There exist a bijection between the set of natural numbers, Turing machine codes and Turing machines.




## Diagonalization

- Let $w_i$ be the $i$-th word in $\{0,1\}^*$ and $M_j$ the $j$-th Turing machine.
- Table $T$ with $t_{ij} = \begin{cases} 1, & \text{if } w_i \in L(M_j) \\ 0, & \text{if } w_i \notin L(M_j) \end{cases}$

| $i \downarrow$ / $j \longrightarrow$ | 1 | 2 | 3 | 4 | ... |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | ... |
| 2 | 1 | 1 | 0 | 1 | ... |
| 3 | 0 | 0 | 1 | 0 | ... |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | |

- *Diagonal language* $L_d = \{w_i \in \{0,1\}^* \mid w_i \notin L(M_i)\}$.
- **Theorem.** $L_d$ is not recursively enumerable.
- *Proof:* Suppose $L_d = L(M_k)$, for some $k \in \mathbb{N}$. Then

$$w_k \in L_d \Leftrightarrow w_k \notin L(M_k),$$

contradicting $L_d = L(M_k)$.

## Universal language

- $\langle M, w \rangle$: encoding $\langle M \rangle$ of $M$ concatenated with $w \in \{0,1\}^*$.
- *Universal language*

$$L_u = \{\langle M, w \rangle \mid M \text{ accepts } w\}$$

- **Theorem.** $L_u$ is recursively enumerable.
- A Turing machine $U$ accepting $L_u$ is called *universal Turing machine.*
- **Theorem** (Turing 1936). $L_u$ is not recursive.

## Decision problems

- Decision problems are problems with answer either yes or no.
- Associate with a language $L \subseteq \Sigma^*$ the decision problem $D_L$

  Input: $w \in \Sigma^*$

  Output: $\begin{cases} \text{yes}, & \text{if } w \in L \\ \text{no}, & \text{if } w \notin L \end{cases}$

  and vice versa.
- $D_L$ is *decidable* (resp. *semi-decidable*) if $L$ is recursive (resp. recursively enumerable).
- $D_L$ is *undecidable* if $L$ is not recursive.

## Reductions

- A *many-one reduction* of $L_1 \subseteq \Sigma_1^*$ to $L_2 \subseteq \Sigma_2^*$ is a computable function $f : \Sigma_1^* \to \Sigma_2^*$ with $w \in L_1 \Leftrightarrow f(w) \in L_2$.
- **Proposition.** If $L_1$ is many-one reducible to $L_2$, then
  1. $L_1$ is decidable if $L_2$ is decidable.
  2. $L_2$ is undecidable if $L_1$ is undecidable.

## Post's correspondence problem

- Given pairs of words

$$(v_1, w_1), (v_2, w_2), \dots, (v_k, w_k)$$

  over an alphabet $\Sigma$, does there exist a sequence of integers $i_1, \dots, i_m, m \geq 1$, such that

$$v_{i_1}, \dots, v_{i_m} = w_{i_1}, \dots, w_{i_m}.$$

- *Example*

| $i$ | $v_i$ | $w_i$ |
|---|---|---|
| 1 | 1 | 111 |
| 2 | 10111 | 10 |
| 3 | 10 | 0 |

$\Rightarrow v_2 v_1 v_1 v_3 = w_2 w_1 w_1 w_3 = 101111110$

- **Theorem** (Post 1946). Post's correspondence problem is undecidable.

## Hilbert's Tenth Problem

*Hilbert, International Congress of Mathematicians, Paris, 1900*

Given a diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.

**Theorem** (Matiyasevich 1970)
Hilbert's tenth problem is undecidable.

## Non-deterministic Turing machines

- *Next move relation:*
$$\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$$
- $L(M)$ = set of words $w \in \Sigma^*$ for which *there exists* a sequence of moves accepting $w$.
- **Proposition.** If $L$ is accepted by a non-deterministic Turing machine $M_1$, then $L$ is accepted by some deterministic machine $M_2$.

## Time complexity

- $M$ a (deterministic) Turing machine that halts on all inputs.
- Time complexity function $T_M : \mathbb{N} \to \mathbb{N}$
$$T_M(n) = \max\{m \mid \exists w \in \Sigma^*, |w| = n \text{ such that the computation of } M \text{ on } w \text{ takes } m \text{ moves}\}$$
(assume numbers are coded in binary format)
- A Turing machine is *polynomial* if there exists a polynomial $p(n)$ with $T_M(n) \leq p(n)$, for all $n \in \mathbb{N}$.
- The *complexity class P* is the class of languages decided by a polynomial Turing machine.

## Time complexity of non-deterministic Turing machines

- $M$ non-deterministic Turing machine
- The running time of $M$ on $w \in \Sigma^*$ is
  - the length of a shortest sequence of moves accepting $w$ if $w \in L(M)$
  - 1, if $w \notin L(M)$
- $T_M(n) = \max\{m \mid \exists w \in \Sigma^*, |w| = n$ such that the running time of $M$ on $w$ is $m\}$
- The *complexity class NP* is the class of languages accepted by a polynomial non-deterministic Turing machine.

## Deciding languages in NP

**Theorem.** If $L \in NP$, then there exists a deterministic Turing machine $M$ and a polynomial $p(n)$ such that

- $M$ decides $L$ and
- $T_M(n) \leq 2^{p(n)}$, for all $n \in \mathbb{N}$.

*Proof:* Suppose $L$ is accepted by a non-deterministic machine $M_{nd}$ whose running time is bounded by the polynomial $q(n)$.

To decide whether $w \in L$, the machine $M$ will

1. determine the length $n$ of $w$ and compute $q(n)$.
2. simulate all executions of $M_{nd}$ of length at most $q(n)$. If the maximum number of choices of $M_{nd}$ in one step is $r$, there are at most $r^{q(n)}$ such executions.

3. if one of the simulated executions accepts $w$, then $M$ accepts $w$, otherwise $M$ rejects $w$.

The overall complexity is bounded by $r^{q(n)} \cdot q'(n) = O(2^{p(n)})$, for some polynomial $p(n)$.

## An alternative characterization of NP

- **Proposition.** $L \in NP$ if there exists $L' \in P$ and a polynomial $p(n)$ such that for all $w \in \Sigma^*$:

$$w \in L \;\Leftrightarrow\; \exists v \in (\Sigma')^* : |v| \leq p(|w|) \text{ and } (w, v) \in L'$$

- Informally, a problem is in $NP$ if it can be solved non-deterministically in the following way:
  1. guess a solution/certificate $v$ of polynomial length,
  2. check in polynomial time whether $v$ has the desired property.

## Propositional satisfiability

- *Satisfiability problem SAT*

  Instance: A formula $F$ in propositional logic with variables $x_1, \dots, x_n$.

  Question: Is $F$ satisfiable, i.e., does there exist an assignment $I : \{x_1, \dots, x_n\} \to \{0, 1\}$ making the formula true ?

- Trying all possible assignments would require exponential time.
- Guessing an assignment $I$ and checking whether it satisfies $F$ can be done in (non-deterministic) polynomial time. Thus:
- **Proposition.** SAT is in $NP$.

## Polynomial reductions

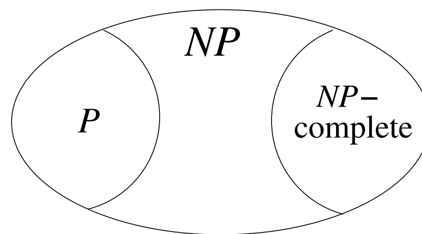
- A *polynomial reduction* of $L_1 \subseteq \Sigma_1^*$ to $L_2 \subseteq \Sigma_2^*$ is a polynomially computable function $f : \Sigma_1^* \to \Sigma_2^*$ with $w \in L_1 \Leftrightarrow f(w) \in L_2$.
- **Proposition.** If $L_1$ is polynomially reducible to $L_2$, then
  1. $L_1 \in P$ if $L_2 \in P$ and $L_1 \in NP$ if $L_2 \in NP$
  2. $L_2 \notin P$ if $L_1 \notin P$ and $L_2 \notin NP$ if $L_1 \notin NP$.
- $L_1$ and $L_2$ are *polynomially equivalent* if they are polynomially reducible to each other.

## NP-complete problems

- A language $L \subseteq \Sigma^*$ is *NP-complete* if
  1. $L \in NP$
  2. Any $L' \in NP$ is polynomially reducible to $L$.
- **Proposition.** If $L$ is $NP$-complete and $L \in P$, then $P = NP$.

- **Corollary.** If $L$ is $NP$-complete and $P \neq NP$, then there exists no polynomial algorithm for $L$.

## Structure of the class NP




**Fundamental open problem:** $P \neq NP$ **?**

## Proving NP-completeness

- **Theorem** (Cook 1971). SAT is *NP*-complete.
- **Proposition.** $L$ is *NP*-complete if
  1. $L \in NP$
  2. there exists an *NP*-complete problem $L'$ that is polynomially reducible to $L$.
- INDEPENDENT SET

  Instance: Graph $G = (V, E)$ and $k \in \mathbb{N}, k \leq |V|$.
  Question: Is there a subset $V' \subseteq V$ such that $|V'| \geq k$ and no two vertices in $V$ are joined by an edge in $E$ ?

## Reducing 3SAT to INDEPENDENT SET

- Let $F$ be a conjunction of $n$ *clauses* of length 3, i.e., a disjunction of 3 propositional variables or their negation.
- Construct a graph $G$ with $3n$ vertices that correspond to the variables in $F$.
- For any clause in $F$, connect by three edges the corresponding vertices in $G$.
- Connect all pairs of vertices corresponding to a variable $x$ and its negation $\neg x$.
- $F$ is satisfiable if and only if $G$ contains an independent set of size $n$.

## Solving numerical constraints

| Satisfiability | over $\mathbb{Q}$ | over $\mathbb{Z}$ | over $\mathbb{N}$ |
|---|---|---|---|
| Linear equations | polynomial | polynomial | *NP*-complete |
| Linear inequalities | polynomial | *NP*-complete | *NP*-complete |

| Satisfiability | over $\mathbb{R}$ | over $\mathbb{Z}$ |
|---|---|---|
| Linear constraints | polynomial | *NP*-complete |
| Nonlinear constraints | decidable | undecidable |

## NP-hard problems

- *Decision problem:* solution is either yes or no
- Example: Traveling salesman decision problem:
  Given a network of cities, distances, and a number B, does there exist a tour with length $\leq$ B?

- *Search problem:* find an object with required properties
- Example: Traveling salesman optimization problem:
  Given a network of cities and distances, find a shortest tour.
- Decision problem *NP*-complete ⇒ search problem *NP*-hard
- *NP-hard problems:* at least as hard as *NP*-complete problems

## Graph theoretical problems

- Shortest path — *polynomial*
- Traveling salesman — *NP-hard*
- Minimum spanning tree — *polynomial*
- Steiner tree — *NP-hard*

## NP-hard problems in bioinformatics

- Multiple sequence alignment — *Wang/Jiang 94*
- Protein folding — *Fraenkel 93*
- Protein threading — *Lathrop 94*
- Protein design — *Pierce/Winfree 02*
- ...

## Approximation algorithms

- Unless $P = NP$, *NP*-hard optimization problems cannot be solved in polynomial time.
- What about polynomial approximation algorithms ?

## NP optimization problems

- An *NP-optimization problem* has the form $\Pi = (\mathcal{I}, \text{Sol}, m, \text{opt})$, where
  - $\mathcal{I}$ is the set of *instances.*
  - $\text{Sol}(I)$, for $I \in \mathcal{I}$, is the set of *feasible* solutions.
  - $m(I,S) \geq 0$ denotes the *value* of $S$.
  - $\text{opt} \in \{\min, \max\}$ is the *type* of optimization problem.
- NPO is the class of *NP*-optimization problems.

## Assumptions

- Any $I \in \mathcal{I}$ can be recognized in time polynomial with $|I|$.
- For every $S \in \text{Sol}(I)$, $|S|$ is polynomial in $|I|$.
- For any $S$ with $|S|$ polynomial in $|I|$, one can decide in polynomial time whether $S \in \text{Sol}(I)$.
- $m$ is polynomially computable.

## Approximation algorithms

- An *approximation algorithm* $A$ for $\Pi$ computes for any instance $I$ a feasible solution $A(I) \in \mathrm{Sol}(I)$.
- If $A$ is polynomial in $|I|$, then $A$ is a *polynomial* approximation algorithm for $\Pi$.
- *Approximation ratio*

$$\rho_A(I) = \frac{m(I, A(I))}{m(I, \mathrm{opt}(I))} \in \begin{cases} [1, \infty), & \text{if opt = min} \\ [0, 1], & \text{if opt = max} \end{cases}$$

## The class APX

- $\Pi$ belongs to the class APX if there is a polynomial approximation algorithm $A$ and $\rho > 0$ such that for any $I \in \mathcal{I}$,

$$\begin{aligned} m(I, A(I)) &\leq \rho \cdot m(I, \mathrm{opt}(I)), && \text{if opt = min} \\ m(I, A(I)) &\geq \rho \cdot m(I, \mathrm{opt}(I)), && \text{if opt = max} \end{aligned}$$

- $A$ is then called a *$\rho$-approximation algorithm.*

## Polynomial time approximation schemes

- $\Pi \in$ PTAS if for any $\varepsilon > 0$ there exists a $\rho$-approximation algorithm $A_\rho$, where $\rho = 1 + \varepsilon$ (for opt = min) resp. $\rho = 1 - \varepsilon$ (for opt = max).
- $\Pi \in$ FPTAS if $\Pi$ belongs to PTAS and if there is polynomial $q(x, y)$ such that the running time of $A_r(I)$ is bounded by $q(|I|, 1/\varepsilon)$.
- PTAS resp. FPTAS stands for *(fully) polynomial time approximation scheme.*
- Assuming $P \neq NP$, one can show

$$\text{FPTAS} \subsetneq \text{PTAS} \subsetneq \text{APX} \subsetneq NPO$$

## 0-1 Knapsack problem

- *KNAPSACK*

  Instance: A finite set $S$ of items, for each $i \in S$ a weight $w_i \in \mathbb{N}$ and a value $v_i \in \mathbb{N}$, a total capacity $c \in \mathbb{N}$, and a value $v \in \mathbb{N}$ (assume $w_i \leq c$).
  Question: Is there a subset $T \subseteq S$ with

$$\sum_{i \in T} w_i \leq c \text{ and } \sum_{i \in T} v_i \geq v$$

- **Theorem.** KNAPSACK is NP-complete
- *Integer programming formulation*

$$\max\{v_1 x_1 + \cdots + v_n x_n \mid w_1 x_1 + \cdots + w_n x_n \leq c, x_i \in \{0, 1\}\}$$

## Dynamic programming algorithm

- $v(i, d)$: maximum value attainable by packing some of the $i$ first items into a knapsack of capacity $d$.

$$v(i, d) = \begin{cases} 0, & \text{if } i = 0 \\ v(i-1, d), & \text{if } i > 0, w_i > d \\ \max\{v(i-1, d), v(i-1, d - w_i) + v_i\}, & \text{if } i > 0, w_i \leq d \end{cases}$$

- $O(nc)$ algorithm for computing $v(n, c)$

## Does it show P = NP ?

- Binary encoding of $c$ requires $k = O(\log_2 c)$ bits, so the running time $O(nc) = O(n2^k)$ is exponential in the input length.
- Using unary encoding, the algorithm runs in polynomial time $\rightsquigarrow$ *pseudo-polynomial algorithm*

## Example

- Instance

| $i$ | 1 | 2 | 3 |
|---|---|---|---|
| $w_i$ | 1 | 2 | 3 |
| $v_i$ | 6 | 10 | 12 |

, capacity $c = 5$

- Dynamic programming table

| $i \setminus d$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 6 | 6 | 6 | 6 | 6 |
| 2 | 0 | 6 | 10 | 16 | 16 | 16 |
| 3 | 0 | 6 | 10 | 16 | 18 | 22 |

- Optimal solution: $x^* = (0, 1, 1)$ of value $v^* = 22$

## Fractional knapsack problem

- Can take fractions of items:

$$\max\{v_1x_1 + \cdots + v_nx_n \mid w_1x_1 + \cdots + w_nx_n \leq c, 0 \leq x_i \leq 1\}$$

- *Greedy algorithm*
  - Sort items by value per weight, i.e. $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$.
  - Pack items in this ordering until no space is left.
  - Solution $(1, \ldots, 1, r, 0 \ldots, 0)$ of value $v_1 + \cdots + v_k + rv_{k+1}$.
- Running time: $O(n \log n)$
- *Example:* $x^* = (1, 1, 2/3)$, $v^* = 24$

## Variant of dynamic programming algorithm

- $w(i, v)$ = *minimum weight* attainable by selecting some among the $i$ first items, so that the value is exactly $v$.

$$w(i, v) = \begin{cases} 0, & \text{if } v = 0 \\ \infty, & \text{if } v > 0, i = 0 \\ w(i-1, v), & \text{if } v_i > v \\ \min\{w(i-1, v), w_i + w(i-1, v-v_i)\}, & \text{otherwise} \end{cases}$$

- Optimal value: $v^* = \max\{v \mid w(n, v) \leq c\}$
- Running time: $O(nv^*) = O(n^2 v_{\max})$

## FPTAS for KNAPSACK

- *Scale and round values:* Define $\hat{v}_i = \left\lfloor \frac{v_i}{b} \right\rfloor$, so $b\hat{v}_i \leq v_i < b\hat{v}_i + b$.
  - $\varepsilon = 1 - \rho$ precision parameter
  - $b = \varepsilon \cdot v_{\max}/n$ scaling factor
- *Intuition*
  - $\hat{v}$ small $\rightsquigarrow$ can get fast solution
  - $b \cdot \hat{v}$ close to $v \rightsquigarrow$ can get near-optimal solution
- *Running time: $O(n^3/\varepsilon)$*
  - Variant of dynamic programming runs in $O(n^2 \hat{v}_{\max})$
  - $\hat{v}_{\max} = \left\lfloor \frac{v_{\max}}{b} \right\rfloor = \left\lfloor \frac{n}{\varepsilon} \right\rfloor$

## Quality of solution

- **Lemma.** If $T_A$ is the solution found by the approximation algorithm and $T^*$ an optimal solution to the original problem, then

$$\begin{aligned} \sum_{i \in T_A} v_i \;\geq\; \sum_{i \in T_A} b\hat{v}_i \;\geq\; \sum_{i \in T^*} b\hat{v}_i \;\geq\; \sum_{i \in T^*} (v_i - b) \;\geq\; \sum_{i \in T^*} v_i - b \cdot n \\ = \; v^* - (\varepsilon v_{\max}/n) \cdot n \;\geq\; (1 - \varepsilon) \cdot v^* \end{aligned}$$

- **Theorem.** KNAPSACK is in FPTAS.

## Traveling salesman problem

- *Hamiltonian circuit*

  Instance: $G = (V, E)$ undirected graph
  Question: Does $G$ contain a simple circuit that contains all vertices in $V$?

- **Theorem.** Hamiltonian circuit is *NP*-complete.
- *Traveling salesman problem (TSP):* Given a complete graph $G = (V, E)$ with integer costs $c(e) \geq 0$, find a Hamiltonian circuit of minimum cost.
- **Theorem.** Unless $P = NP$, there is no $\rho$-approximation algorithm for TSP for any $\rho \geq 1$.

## Proof

- Suppose $A$ is $\rho$-approximation algorithm for TSP.
- Use $A$ to solve an instance $G = (V, E)$ of Hamiltonian circuit.
- Create an instance $I$ of TSP with $|V|$ vertices and weights

$$c(e) = \begin{cases} 1, & \text{if } e \in E \\ \rho \cdot |V| + 1, & \text{if } e \notin E \end{cases}$$

- $C$ Hamiltonian circuit in $G \Leftrightarrow C$ has cost exactly $|V|$ in $I \Leftrightarrow \; m(I, A(I)) \leq \rho|V|$

- $C$ not Hamiltonian circuit in $G \Leftrightarrow C$ has cost at least $\rho \cdot |V| + 1$ in $I \Leftrightarrow \; m(I, A(I)) \geq \rho |V| + 1$

## Further complexity classes

*coNP:* Problems whose complement is in *NP*

*PSPACE:* Problems solvable in polynomial space

*EXPTIME:* Problems solvable in exponential time

⋮

## Literature


- J. E. Hopcroft and J. D. Ullman: Introduction to automata theory, languages and computation. Addison-Wesley, 1979
- M. R. Garey and D. S. Johnson: Computers and intractability. A guide to the theory of NP-completeness. Freeman, 1979
- C. H. Papadimitriou: Computational complexity. Addison-Wesley, 1994