Languages and Algorithms for Artificial Intelligence (Third Module)

Between the Feasible and the Unfeasible

Ugo Dal Lago

University of Bologna, Academic Year 2019/2020

The Border Between the Tractable and the Intractable

- Up to now, we have encountered (essentially speaking) two complexity classes, namely:
  - **P**, which contains those problems which can be solved in polynomial time, so the **tractable** ones.
  - **EXP**, which contains the whole of **P**, but also some problems which *cannot* be solved in polynomial time, intrinsically requiring exponential time (and as such, **intractable**).

The Border Between the Tractable and the Intractable

- Up to now, we have encountered (essentially speaking) two complexity classes, namely:
  - **P**, which contains those problems which can be solved in polynomial time, so the **tractable** ones.
  - **EXP**, which contains the whole of **P**, but also some problems which *cannot* be solved in polynomial time, intrinsically requiring exponential time (and as such, **intractable**).
- It's now time to study the "border" between tractability and intractability:
  - Between **P** and **EXP**, one can define *many other* classes. i.e. there are many ways of defining a class **A** such that

$$\mathbf{P} \subseteq \mathbf{A} \subseteq \mathbf{EXP}$$

  - This is a formidable tool to classify those problems in **EXP** for which *we do not know* whether they are in **P** (and there are *so many* of them).

Creating vs. Verifying

- Very often, the language we would like to classify can be written as follows:

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)}.(x,y) \in \mathcal{A}\}$$

where $p : \mathbb{N} \to \mathbb{N}$ and $\mathcal{A}$ is a set of pairs of strings.

Creating vs. Verifying

- Very often, the language we would like to classify can be written as follows:

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)}.(x,y) \in \mathcal{A}\}$$

  where $p : \mathbb{N} \to \mathbb{N}$ and $\mathcal{A}$ is a set of pairs of strings.
- In other words, the elments of $\mathcal{L}$ are those strings for which we can find a *certificate* y (of polynomial length) such that the pair (x, y) passes the *test* $\mathcal{A} \subseteq \{0,1\}^* \times \{0,1\}^*$.

Creating vs. Verifying

- Very often, the language we would like to classify can be written as follows:

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)}.(x,y) \in \mathcal{A}\}$$

  where $p : \mathbb{N} \to \mathbb{N}$ and $\mathcal{A}$ is a set of pairs of strings.
- In other words, the elments of $\mathcal{L}$ are those strings for which we can find a *certificate* y (of polynomial length) such that the pair (x, y) passes the *test* $\mathcal{A} \subseteq \{0,1\}^* \times \{0,1\}^*$.
- What if $\mathcal{A}$ is itself decidable in polynomial time? Does this imply that $\mathcal{L}$ is itself decidable in polynomial time?

Creating vs. Verifying

- Very often, the language we would like to classify can be written as follows:

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)}.(x,y) \in \mathcal{A}\}$$

  where $p : \mathbb{N} \to \mathbb{N}$ and $\mathcal{A}$ is a set of pairs of strings.
- In other words, the elments of $\mathcal{L}$ are those strings for which we can find a *certificate* y (of polynomial length) such that the pair (x, y) passes the *test* $\mathcal{A} \subseteq \{0,1\}^* \times \{0,1\}^*$.
- What if $\mathcal{A}$ is itself decidable in polynomial time? Does this imply that $\mathcal{L}$ is itself decidable in polynomial time?
  - *Not necessarily*: given x, we can check whether $x \in \mathcal{L}$ by $(x, y) \in \mathcal{A}$ for all possible possible y such that $|y| \leq p(|x|)$, of which however there are *exponentially* many.
  - Of course this does *not* rule out other strategies to decide $\mathcal{L}$.

Creating vs. Verifying

- Very often, the language we would like to classify can be written as follows:

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)}.(x,y) \in \mathcal{A}\}$$

  where $p : \mathbb{N} \to \mathbb{N}$ and $\mathcal{A}$ is a set of pairs of strings.
- In other words, the elments of $\mathcal{L}$ are those strings for which we can find a *certificate* y (of polynomial length) such that the pair (x, y) passes the *test* $\mathcal{A} \subseteq \{0,1\}^* \times \{0,1\}^*$.
- What if $\mathcal{A}$ is itself decidable in polynomial time? Does this imply that $\mathcal{L}$ is itself decidable in polynomial time?
  - *Not necessarily*: given x, we can check whether $x \in \mathcal{L}$ by $(x, y) \in \mathcal{A}$ for all possible possible y such that $|y| \leq p(|x|)$, of which however there are *exponentially* many.
  - Of course this does *not* rule out other strategies to decide $\mathcal{L}$.
- The *take-away message* is thus the following: **crafting** a solution for the problem x (i.e., finding y) can potentially be more difficult than just **checking** y to be a solution to x.

The Complexity Class **NP**

- A language $\mathcal{L} \subseteq \{0,1\}^*$ is in the class **NP** iff there exist a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time TM $\mathcal{M}$ such that

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(\llcorner x, y \lrcorner) = 1\}$$

The Complexity Class **NP**

- A language $\mathcal{L} \subseteq \{0,1\}^*$ is in the class **NP** iff there exist a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time TM $\mathcal{M}$ such that

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(\llcorner x, y \lrcorner) = 1\}$$

- With the hypotheses above:
  - M is said to be the **verifier** for $\mathcal{L}$.
  - Any $y \in \{0,1\}^{p(|x|)}$ such that $\mathcal{M}(\llcorner (x,y) \lrcorner) = 1$ is said to be a **certificate** for x.

The Complexity Class **NP**

- A language $\mathcal{L} \subseteq \{0,1\}^*$ is in the class **NP** iff there exist a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time TM $\mathcal{M}$ such that

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(\llcorner x, y \lrcorner) = 1\}$$

- With the hypotheses above:
  - M is said to be the **verifier** for $\mathcal{L}$.
  - Any $y \in \{0,1\}^{p(|x|)}$ such that $\mathcal{M}(\llcorner (x,y) \lrcorner) = 1$ is said to be a **certificate** for x.
- Differently from **P** and **EXP**, the class **NP** does not have a natural counterpart as a class of *functions*.

The Complexity Class **NP**

- A language $\mathcal{L} \subseteq \{0,1\}^*$ is in the class **NP** iff there exist a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time TM $\mathcal{M}$ such that

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(\llcorner x, y \lrcorner) = 1\}$$

- With the hypotheses above:
  - M is said to be the **verifier** for $\mathcal{L}$.
  - Any $y \in \{0,1\}^{p(|x|)}$ such that $\mathcal{M}(\llcorner (x,y) \lrcorner) = 1$ is said to be a **certificate** for x.
- Differently from **P** and **EXP**, the class **NP** does not have a natural counterpart as a class of *functions*.

Theorem

P $\subseteq$ NP $\subseteq$ EXP

Examples Problems in **NP**

1. **Maximum Independent Set**
   - In its decision form, it asks whether a pair $(\mathbb{G}, k)$ of an undirected graph and a natural number $k \in \mathbb{N}$ is such that $\mathbb{G} = (V, E)$ admits an independent set $W \subseteq V$ of cardinality at least k.

Examples Problems in **NP**

1. **Maximum Independent Set**
   - In its decision form, it asks whether a pair $(\mathbb{G}, k)$ of an undirected graph and a natural number $k \in \mathbb{N}$ is such that $\mathbb{G} = (V, E)$ admits an independent set $W \subseteq V$ of cardinality at least k.
   - Certificate: the certificate here is just W. Indeed, checking whether W is independent can easily be done in polynomial time.

Examples Problems in **NP**

1. **Maximum Independent Set**
   - In its decision form, it asks whether a pair $(\mathbb{G}, k)$ of an undirected graph and a natural number $k \in \mathbb{N}$ is such that $\mathbb{G} = (V, E)$ admits an independent set $W \subseteq V$ of cardinality at least k.
   - Certificate: the certificate here is just W. Indeed, checking whether W is independent can easily be done in polynomial time.
2. **Subset Sum**
   - It asks whether, given a sequence of natural numbers $n_1, \ldots, n_m$ and a number k, there exists a subset I of $\{1, \ldots, m\}$ such that $\sum_{i \in I} n_i = k$.

Examples Problems in **NP**

1. **Maximum Independent Set**
   - In its decision form, it asks whether a pair $(\mathbb{G}, k)$ of an undirected graph and a natural number $k \in \mathbb{N}$ is such that $\mathbb{G} = (V, E)$ admits an independent set $W \subseteq V$ of cardinality at least k.
   - Certificate: the certificate here is just W. Indeed, checking whether W is independent can easily be done in polynomial time.
2. **Subset Sum**
   - It asks whether, given a sequence of natural numbers $n_1, \ldots, n_m$ and a number k, there exists a subset I of $\{1, \ldots, m\}$ such that $\sum_{i \in I} n_i = k$.
   - Certificate: again, the certificate here is just I: checking whether $\sum_{i \in I} n_i = k$ just amounts to some additions and comparisons.

Examples Problems in **NP**

3. **Composite Numbers**
   - Given a number $n \in \mathbb{N}$, determine whether n is composite (i.e., not prime).

Examples Problems in **NP**

3. **Composite Numbers**
   - Given a number $n \in \mathbb{N}$, determine whether n is composite (i.e., not prime).
   - Certificate: it is the factorization of n, a pair (m, l) of natural numbers (greater than 2) such that $n = m \cdot l$.

Examples Problems in **NP**

3. **Composite Numbers**
   - Given a number $n \in \mathbb{N}$, determine whether n is composite (i.e., not prime).
   - Certificate: it is the factorization of n, a pair (m, l) of natural numbers (greater than 2) such that $n = m \cdot l$.
4. **Factoring**
   - Given three numbers n, m, l, it asks whether n has a prime factor in the interval $[m, l]$.

Examples Problems in **NP**

3. **Composite Numbers**
   - Given a number $n \in \mathbb{N}$, determine whether n is composite (i.e., not prime).
   - Certificate: it is the factorization of n, a pair (m, l) of natural numbers (greater than 2) such that $n = m \cdot l$.
4. **Factoring**
   - Given three numbers n, m, l, it asks whether n has a prime factor in the interval $[m, l]$.
   - Certificate: it can be taken to be the prime number p: checking that $p \in [m, l]$ and that p divides n is easy. Instead, checking that p is prime requires a lot of work, but can indeed be done in polynomial time.

Examples Problems in **NP**

3. **Decisional Linear Programming**
   - Given a sequence of m linear inequalities with rational coefficients over n variables (i.e. inequalities of the form $\sum_{i=1}^{n} a_i x_i \leq b$, where the coefficients $a_1, \ldots, a_n, b$ are in $\mathbb{Q}$), decide whether there is a rational assignment to the variables $x_1, \ldots, x_n$ which makes all the inequalities true.

Examples Problems in **NP**

3. **Decisional Linear Programming**
   - Given a sequence of m linear inequalities with rational coefficients over n variables (i.e. inequalities of the form $\sum_{i=1}^{n} a_i x_i \leq b$, where the coefficients $a_1, \ldots, a_n, b$ are in $\mathbb{Q}$), decide whether there is a rational assignment to the variables $x_1, \ldots, x_n$ which makes all the inequalities true.
   - Certificate: the assignment, which can be easily checked for correctness.

Examples Problems in **NP**

3. **Decisional Linear Programming**
   - Given a sequence of m linear inequalities with rational coefficients over n variables (i.e. inequalities of the form $\sum_{i=1}^{n} a_i x_i \leq b$, where the coefficients $a_1, \ldots, a_n, b$ are in $\mathbb{Q}$), decide whether there is a rational assignment to the variables $x_1, \ldots, x_n$ which makes all the inequalities true.
   - Certificate: the assignment, which can be easily checked for correctness.
4. **Decisional 0/1 Linear Programming**
   - Given a sequence of linear inequalities as above, decide whether there is an assignment *of zeros and ones* to the variables $x_1, \ldots, x_n$ rendering all the inequalities true.

Examples Problems in **NP**

3. **Decisional Linear Programming**
   - Given a sequence of m linear inequalities with rational coefficients over n variables (i.e. inequalities of the form $\sum_{i=1}^{n} a_i x_i \leq b$, where the coefficients $a_1, \ldots, a_n, b$ are in $\mathbb{Q}$), decide whether there is a rational assignment to the variables $x_1, \ldots, x_n$ which makes all the inequalities true.
   - Certificate: the assignment, which can be easily checked for correctness.
4. **Decisional 0/1 Linear Programming**
   - Given a sequence of linear inequalities as above, decide whether there is an assignment *of zeros and ones* to the variables $x_1, \ldots, x_n$ rendering all the inequalities true.
   - Certificate: again, the assignments suffices.

Examples Problems in **NP**

- Some of the aforementioned problems are also in **P**:
  - *Decisional linear programming* can be proved to be in **P** thanks to, e.g., the Ellipsoid algorithm.
  - The *composite numbers* problem can be proved itself to be in **P**, thanks to a breakthrough recent result, namely the so-called AKS algorithm.

Examples Problems in **NP**

- Some of the aforementioned problems are also in **P**:
  - *Decisional linear programming* can be proved to be in **P** thanks to, e.g., the Ellipsoid algorithm.
  - The *composite numbers* problem can be proved itself to be in **P**, thanks to a breakthrough recent result, namely the so-called AKS algorithm.
- All the other problems are currently **not known** to be in **P**.
  - Are they all equivalent in terms of their inherent computational difficulty?
  - Is there any way isolate those problems in **NP** whose difficult is maximal, i.e. they are at least as hard as all other problems in **NP**?

Nondeterministic Turing Machines

- The class **NP** can also be defined using a variant of Turing machines, called the *nondeterministic* Turing machines (NDTM for short).
  - This is the original definition by Hartmanis and Stearns, the founding fathers of computational complexity.
  - This is also the reason for the letter **N** in **NP**.

Nondeterministic Turing Machines

- The class **NP** can also be defined using a variant of Turing machines, called the *nondeterministic* Turing machines (NDTM for short).
  - This is the original definition by Hartmanis and Stearns, the founding fathers of computational complexity.
  - This is also the reason for the letter **N** in **NP**.
- The only differences between a NDTM and an ordinary TM is that the former has:
  - Two transition functions δ_0 and δ_1 rather than just one. *At every step*, the machine chooses nondeterministically one between the two transition functions and proceed according to it
  - A special state $q_{\texttt{accept}}$.

Nondeterministic Turing Machines

- The class **NP** can also be defined using a variant of Turing machines, called the *nondeterministic* Turing machines (NDTM for short).
  - This is the original definition by Hartmanis and Stearns, the founding fathers of computational complexity.
  - This is also the reason for the letter **N** in **NP**.
- The only differences between a NDTM and an ordinary TM is that the former has:
  - Two transition functions δ_0 and δ_1 rather than just one. *At every step*, the machine chooses nondeterministically one between the two transition functions and proceed according to it
  - A special state $q_{\texttt{accept}}$.
- We say that a NDTM M:
  - **Accepts** the input $x \in \{0,1\}^*$ iff *there exists* one among the many possible evolutions of the machine M when fed with x which makes it reaching $q_{\texttt{accept}}$.
  - **Rejects** the input $x \in \{0,1\}^*$ iff *none* of the aformentioned evolutions leads to $q_{\texttt{accept}}$.

Nondeterministic Turing Machines

- The class **NP** can also be defined using a variant of Turing machines, called the *nondeterministic* Turing machines (NDTM for short).
  - This is the original definition by Hartmanis and Stearns, the founding fathers of computational complexity.
  - This is also the reason for the letter **N** in **NP**.
- The only differences between a NDTM and an ordinary TM is that the former has:
  - Two transition functions δ_0 and δ_1 rather than just one. *At every step*, the machine chooses nondeterministically one between the two transition functions and proceed according to it
  - A special state $q_{\texttt{accept}}$.
- We say that a NDTM M:
  - **Accepts** the input $x \in \{0,1\}^*$ iff *there exists* one among the many possible evolutions of the machine M when fed with x which makes it reaching $q_{\texttt{accept}}$.
  - **Rejects** the input $x \in \{0,1\}^*$ iff *none* of the aformentioned evolutions leads to $q_{\texttt{accept}}$.

Time-Bounded NDTMs

- We say that a NDTM M *runs in time* $T : \mathbb{N} \to \mathbb{N}$ iff for every $x \in \{0,1\}^*$ *and for every possible nondeterministic evolution*, M reaches either the halting state or $q_{\texttt{accept}}$ within $c \cdot T(|x|)$ steps, where $c > 0$.

Time-Bounded NDTMs

- We say that a NDTM M *runs in time* $T : \mathbb{N} \to \mathbb{N}$ iff for every $x \in \{0,1\}^*$ *and for every possible nondeterministic evolution*, M reaches either the halting state or $q_{\texttt{accept}}$ within $c \cdot T(|x|)$ steps, where $c > 0$.
- For every function $T : \mathbb{N} \to \mathbb{N}$ and $\mathcal{L} \subseteq \{0,1\}^*$, we say that $\mathcal{L} \in \mathbf{NDTIME}(T(n))$ iff there is a NDTM M working in time T and such that $M(x) = 1$ iff $x \in \mathcal{L}$.

Time-Bounded NDTMs

- We say that a NDTM M *runs in time* $T : \mathbb{N} \to \mathbb{N}$ iff for every $x \in \{0,1\}^*$ *and for every possible nondeterministic evolution*, M reaches either the halting state or $q_{\texttt{accept}}$ within $c \cdot T(|x|)$ steps, where $c > 0$.
- For every function $T : \mathbb{N} \to \mathbb{N}$ and $\mathcal{L} \subseteq \{0,1\}^*$, we say that $\mathcal{L} \in \mathbf{NDTIME}(T(n))$ iff there is a NDTM M working in time T and such that $M(x) = 1$ iff $x \in \mathcal{L}$.

Theorem

$\mathbf{NP} = \cup_{c \in \mathbb{N}} \mathbf{NDTIME}(n^c)$

Time-Bounded NDTMs

- We say that a NDTM M *runs in time* $T : \mathbb{N} \to \mathbb{N}$ iff for every $x \in \{0,1\}^*$ *and for every possible nondeterministic evolution*, M reaches either the halting state or $q_{\texttt{accept}}$ within $c \cdot T(|x|)$ steps, where $c > 0$.
- For every function $T : \mathbb{N} \to \mathbb{N}$ and $\mathcal{L} \subseteq \{0,1\}^*$, we say that $\mathcal{L} \in \mathbf{NDTIME}(T(n))$ iff there is a NDTM M working in time T and such that $M(x) = 1$ iff $x \in \mathcal{L}$.

Theorem

$\mathbf{NP} = \cup_{c \in \mathbb{N}} \mathbf{NDTIME}(n^c)$

- All this being said, NDTMs, contrarily to TMs, are *not* meant to model any form of physically realisable machine.

Thank You!

Questions?