Languages and Algorithms for Artificial Intelligence (Third Module)

Polynomial Time Computable Problems

Ugo Dal Lago

University of Bologna, Academic Year 2019/2020

Complexity Classes

- A **complexity class** is a set of *tasks* which can be computed within some prescribed resource bounds.
  - It is *not* a set of TMs, although it is defined based on TMs.
  - Typically, the task we are interested at are decision problems, or equivalently languages (i.e. subsets of $\{0, 1\}^*$).

Complexity Classes

- A **complexity class** is a set of *tasks* which can be computed within some prescribed resource bounds.
  - It is *not* a set of TMs, although it is defined based on TMs.
  - Typically, the task we are interested at are decision problems, or equivalently languages (i.e. subsets of $\{0,1\}^*$).
- Let $T : \mathbb{N} \to \mathbb{N}$. A language $\mathcal{L}$ is in the class $\mathbf{DTIME}(T(n))$ iff *there is* a TM deciding $\mathcal{L}$ and running in time $n \mapsto c \cdot T(n)$ for some constant c.

Complexity Classes

- A **complexity class** is a set of *tasks* which can be computed within some prescribed resource bounds.
  - It is *not* a set of TMs, although it is defined based on TMs.
  - Typically, the task we are interested at are decision problems, or equivalently languages (i.e. subsets of $\{0,1\}^*$).
- Let $T : \mathbb{N} \to \mathbb{N}$. A language $\mathcal{L}$ is in the class $\mathbf{DTIME}(T(n))$ iff *there is* a TM deciding $\mathcal{L}$ and running in time $n \mapsto c \cdot T(n)$ for some constant c.
- The letter "D" in $\mathbf{DTIME}(\cdot)$ refers to *determinism*: the machines on which the class is based work deterministically.
- Should we study efficiently solvable tasks by way of classes in the form $\mathbf{DTIME}(T(n))$?
  - The answer is bound to be negative, because these classes are not **robust**, they depends too much on
  - We need a larger class.

The Class $\mathbf{P}$

- The class $\mathbf{P}$ is defined as follows:

$$\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(n^c).$$

- In other words, the class $\mathbf{P}$ includes all those languages $\mathcal{L}$:
    1. which can be decided by a TM;
    2. working in time P;
    3. where P is a any polynomial.

The Class $\mathbf{P}$

- The class $\mathbf{P}$ is defined as follows:

$$\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(n^c).$$

- In other words, the class $\mathbf{P}$ includes all those languages $\mathcal{L}$:
  1. which can be decided by a TM;
  2. working in time P;
  3. where P is a any polynomial.
- Indeed, for any any polynomial P there are $c, d > 0$ such that $P(n) \geq c \cdot n^d$ for sufficiently large n.

The Class $\mathbf{P}$

- The class $\mathbf{P}$ is defined as follows:

$$\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(n^c).$$

- In other words, the class $\mathbf{P}$ includes all those languages $\mathcal{L}$:
  1. which can be decided by a TM;
  2. working in time P;
  3. where P is a any polynomial.
- Indeed, for any any polynomial P there are $c, d > 0$ such that $P(n) \geq c \cdot n^d$ for sufficiently large n.
- Please observe that c and d can be arbitrarily large, so a TM deciding $\mathcal{L}$ and working in time $10^{20} \cdot n^{10^{30}}$ is a witness of $\mathcal{L}$ being in $\mathbf{P}$.

The Class $\mathbf{P}$

- The class $\mathbf{P}$ is defined as follows:

$$\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(n^c).$$

- In other words, the class $\mathbf{P}$ includes all those languages $\mathcal{L}$:
  1. which can be decided by a TM;
  2. working in time P;
  3. where P is a any polynomial.
- Indeed, for any any polynomial P there are $c, d > 0$ such that $P(n) \geq c \cdot n^d$ for sufficiently large n.
- Please observe that c and d can be arbitrarily large, so a TM deciding $\mathcal{L}$ and working in time $10^{20} \cdot n^{10^{30}}$ is a witness of $\mathcal{L}$ being in $\mathbf{P}$.
- $\mathbf{P}$ is generally considered as *the* class of efficiently decidable languages.

The (Strong) Church-Turing Thesis

- But, again, why basing complexity theory on TMs? They are a rather simplicistic model!

The (Strong) Church-Turing Thesis

- But, again, why basing complexity theory on TMs? They are a rather simplicistic model!
- **The Church-Turing Thesis**
  - Every physically realizable computer can be simulated by a TM with a (possibly *very large*) overhead in time.
  - The class of computable tasks *would not be larger* (actually, equal!) if formalized in a realistic way, but differently.
  - Most scientists believe in it.

The (Strong) Church-Turing Thesis

- But, again, why basing complexity theory on TMs? They are a rather simplicistic model!
- **The Church-Turing Thesis**
  - Every physically realizable computer can be simulated by a TM with a (possibly *very large*) overhead in time.
  - The class of computable tasks *would not be larger* (actually, equal!) if formalized in a realistic way, but differently.
  - Most scientists believe in it.
- **The Strong Church-Turing Thesis**
  - Every physically realizable computer can be simulated by a TM with a *polynomial* overhead in time. (n steps on the computer requires n^c on TMs, where c only depends on the computer), and viceversa.
  - The class **P** would be *the same* if defined based on other realistic models of computation.
  - This is more controversial (due to, e.g., quantum computation).

Why Polynomials?

- **P is Robust**
  - As already mentioned, polynomials seem to be the smallest class of bounds which make **P** a robust class.

Why Polynomials?

- **P is Robust**
  - As already mentioned, polynomials seem to be the smallest class of bounds which make **P** a robust class.
- **Exponents are Often Small**
  - In principle, the exponent c bounding the time of any machine deciding $\mathcal{L} \in \mathbf{P}$ can be huge.
  - For many problems of interest and in **P**, there are TMs working within quadratic or cubit bounds.

Why Polynomials?

- **P is Robust**
  - As already mentioned, polynomials seem to be the smallest class of bounds which make **P** a robust class.
- **Exponents are Often Small**
  - In principle, the exponent c bounding the time of any machine deciding $\mathcal{L} \in \mathbf{P}$ can be huge.
  - For many problems of interest and in **P**, there are TMs working within quadratic or cubit bounds.
- **Nice Closure Properties**
  - The class is closed various operations on programs, e.g. composition and bounded loops (with some restrictions!).
  - As a consequence, it is relatively easy to prove that a given problem/task is *in* the class: it suffices to give an algorithm solving the problem and working in polynomial time, without constructing the TM explicitly.

Some Criticisms on **P**

- **Worst-Case is Not Realistic**
  - The definition of **P** is intrinsically based on worst-case complexity: there must be *a* polynomial and *a* TM such that *for every input...*
  - It is good enough is our problem takes little time *on the types of inputs* which arise in practice, and not on *all* of them.
  - Solutions: Average-case Complexity, Approximation Algorithms

Some Criticisms on **P**

- **Worst-Case is Not Realistic**
  - The definition of **P** is intrinsically based on worst-case complexity: there must be *a* polynomial and *a* TM such that *for every input*. . .
  - It is good enough is our problem takes little time *on the types of inputs* which arise in practice, and not on *all* of them.
  - Solutions: Average-case Complexity, Approximation Algorithms
- **Alternative Computational Models**
  - Feasibility can be also be defined for classes dealine with arbitrary precision computation, with randomized computation, or with quantum computation.
  - Solutions: the class **P** can be spelled out with other computational models in mind, giving rise to other classes (e.g. **BPP** or **BQP**).
- **Why Just Decision Problems?**
  - As alreday pointed out, not all tasks can be modeled this way.

Thank You!

Questions?