Parameterized Complexity Theory and its Applications to Social Choice

Simon Rey and Ronald de Haan

June Project

Institute for Logic, Language and Computation
University of Amsterdam

Us and You

The project

Four lectures:

1. Today until 15:00: Social choice, standard complexity and fixed parameter tractability
2. Tuesday 11:00-13:00: Treewidth and parameterized complexity on tree-like structures
3. Thursday 9:00-11:00: Hardness theory for parameterized complexity
4. Friday 13:00-15:00: Lower bounds on kernelization

Assessment in two steps:

- Technical presentations during the second week
- Final outcome: a short paper on a topic of your choice

All the details are on the website: http://simonrey.fr/en/teaching/PCT2021.

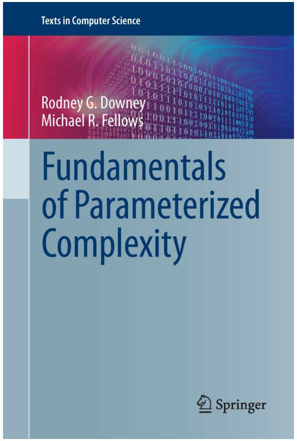
Recap of the Deadlines

All the things you should not forget to do:

- Thursday June 3, before 19:00: preferences for the presentations;
- Tuesday June 15, before 19:00: topic for the paper;
- Tuesday June 15, before 19:00: time slot the paper meeting;
- Friday June 25, before 19:00: final paper.

DO YOU HAVE AN IDEA FOR YOUR PROJECT YET?
NO, I'M WAITING FOR INSPIRATION.
YOU CAN'T JUST TURN ON CREATIVITY LIKE A FAUCET. YOU HAVE TO BE IN THE RIGHT MOOD.
WHAT MOOD IS THAT?
LAST-MINUTE PANIC.

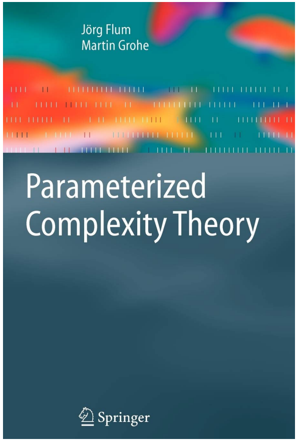
Useful References for the Project


Texts in Computer Science

Rodney G. Downey
Michael R. Fellows

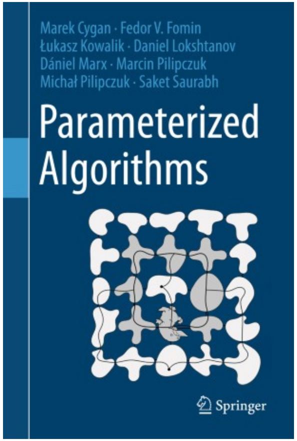
Fundamentals of Parameterized Complexity

Springer


Jörg Flum
Martin Grohe

Parameterized Complexity Theory

Springer


Marek Cygan · Fedor V. Fomin
Łukasz Kowalik · Daniel Lokshtanov
Dániel Marx · Marcin Pilipczuk
Michał Pilipczuk · Saket Saurabh

Parameterized Algorithms

Springer

1. Introduction

Introduction

(Computational) Social Choice

Social Choice Theory

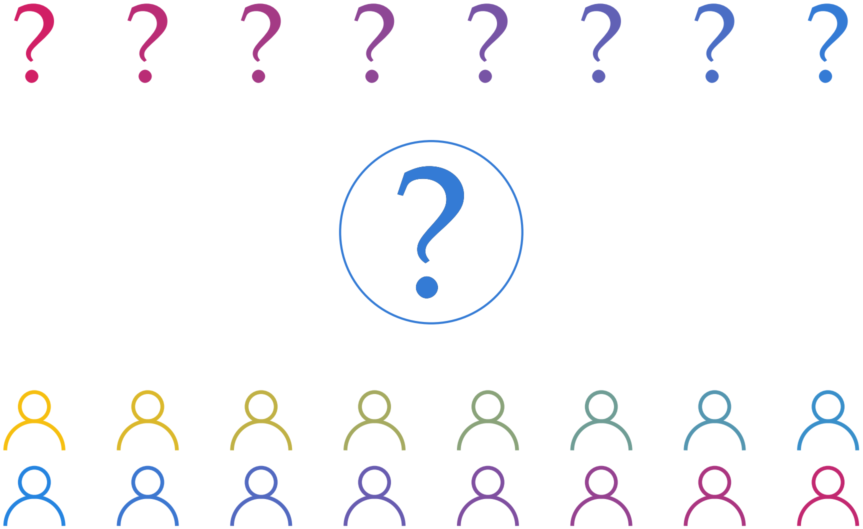
Social choice theory studies problems in which the *outcome* is to be determined by taking into account the *preferences* several agents expressed over the admissible outcomes.

↳ It can also be seen as the study of how to aggregate a set of information into a single outcome.

Social choice theory includes:

- Voting theory
- Judgment aggregation
- Fair division
- Coalition formation
- ...

First Example: Voting



Voting Theory: The Model

The mathematical model we study consists in:

- A set of *candidates*: individuals, parties, projects...
- A set of *admissible outcomes*: single candidates, sets of candidates, sets of projects satisfying the budget constraint...
- A set of *agents*: citizens, residents...
- *Preferences* expressed by the agents : rankings over the candidates, subsets of the candidates...

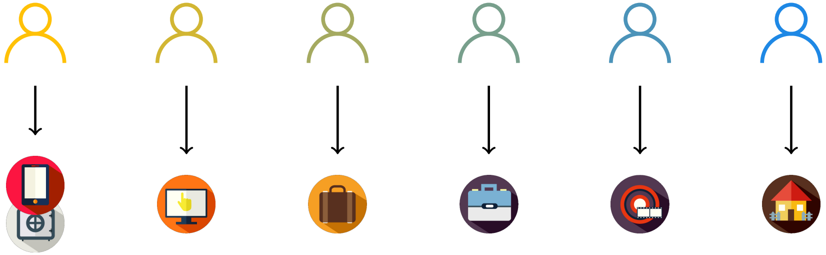
The goal is to select *one or several admissible outcomes* given the preferences submitted by the agents.

↪ We define and study *voting rules*, i.e., mappings taking as input an election and the preferences of the voters and returning one or several admissible outcomes.

Second Example: Fair Division

	👤	👤	👤	👤	👤	👤
	✗	✓	✗	✓	✓	✗
	✓	✓	✗	✗	✓	✓
	✓	✓	✗	✗	✓	✓
	✗	✓	✗	✗	✓	✓
	✓	✓	✓	✓	✓	✗
	✗	✓	✓	✓	✓	✗
	✓	✓	✓	✓	✓	✗

Second Example: Fair Division



Fair Division: The Model

The mathematical model we study consists in:

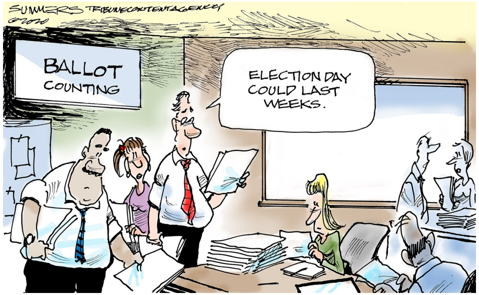
- A set of *items*: objects, tasks, chores...
- A set of *agents*: citizens, employees, friends...
- *Preferences* expressed by the agents : rankings over the items, valuations of the items...

The goal is to *fairly assign the items* to the agents by taking into consideration the preferences they submitted.

↪ We define and study *allocation rules*, i.e., mappings taking as input an allocation instance and the preferences of the agents and returning one or several item allocations.

From Social Choice to Computational Social Choice

The shift from social choice to computational social choice is made by exploring the *computational problems* arising in the study of social choice.


SUMMERS TRIBUNECONTENTAGENCY ©2020
BALLOT COUNTING
ELECTION DAY COULD LAST WEEKS.

- How hard is it to compute the winner of an election for a specific voting rule?
- How hard is it to find an allocation satisfying a given fairness criteria?

↪ These questions can all be answered using *complexity theory*.

Introduction

Complexity Theory

Complexity Theory

Complexity theory studies how hard it is to solve a problem.

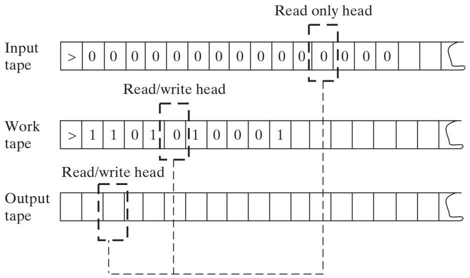
- What do we mean by "solve"?
  ↳ We use an *abstract model of computation*—Turing machine—that gives a precise mathematical definition of what an algorithm is. Solving a problem then means defining a program for this abstract machine which answers the question.
- How do we measure the "hardness" of solving a problem?
  ↳ Different measures have been considered but the one we will use consists on counting *number of elementary steps* needed to run a program solving the problem on a Turing machine.

Turing Machines

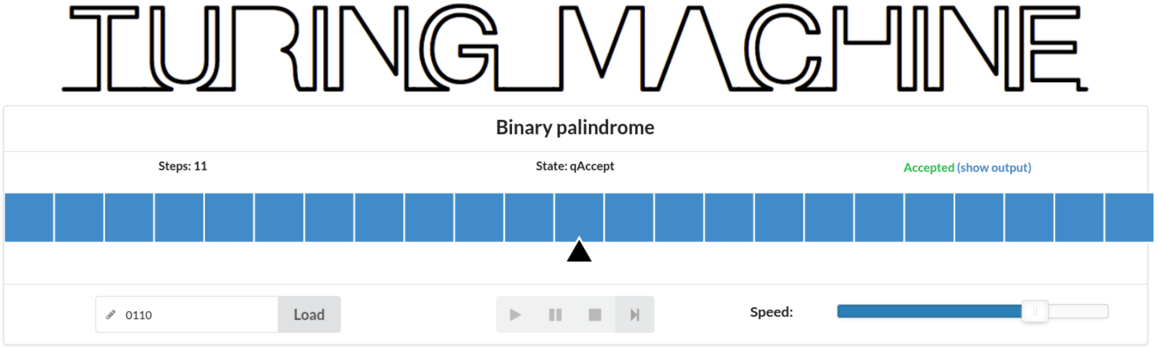
Definition: Turing Machine

A Turing machine is a tuple $\langle \Gamma, Q, \delta \rangle$, where:

- Γ is the *alphabet*, i.e., the set of symbols used by the machine. It includes 0, 1, $\square$ (blank symbol) and $\triangleright$ (start symbol).
- Q is a *finite set of states*, i.e., the set of states the machine can be in at anytime. It includes q_{start} and q_{halt}.
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, R, S\}^k$ is the *transition function* indicating for each state and each symbol read on its k tapes the next configuration of the machine (state, symbols to write and movements of the heads).



Execution of a Turing Machine



Binary palindrome

Steps: 11

State: qAccept

Accepted (show output)

0110

`https://turingmachinesimulator.com`

Decision Problems and Languages

We focus on *decision problems* for which the answer is either Yes or No. They are represented as *languages* $L \subseteq \Sigma^*$ such that the answer for $x \in \Sigma^*$ is yes if and only $x \in L$.

We will say that a Turing machine $\mathbb{M}$ *decides* a language L if for every input $x \in \Sigma^*$, $\mathbb{M}$ halts on an accepting state on x if and only if $x \in L$.

<ins>Example</ins>: The language of all the binary palindromes defined as:

$$L_{pal} = \{x \in \{0,1\}^* \mid x \text{ is a palindrome}\}$$

is decided by the Turing machine $\mathbb{M}$ described on the previous slide. The decision problem given $x \in \{0,1\}^*$, is x a palindrome is thus decided by $\mathbb{M}$.

Complexity Classes

The running time of a Turing machine $\mathbb{M}$ is defined as the *number of steps* it requires to reach an halting state. $\mathbb{M}$ *runs in time* $g : \mathbb{N} \to \mathbb{N}$ if on every input $x \in \Sigma^n$ of length n, $\mathbb{M}$ halts after at most $g(n)$ steps.

We usually study *asymptotic running times* using the big-$\mathcal{O}$ notation. For two functions $f, g : \mathbb{N} \to \mathbb{N}$, we say that f is $\mathcal{O}(g)$ if there exists $c \in \mathbb{N}$ and an $n_0 \in \mathbb{N}$ such that $f(n) \leq c \times g(n)$ for all $n \geq n_0$. In words, f is $\mathcal{O}(g)$ if *g asymptotically upper-bounds f*.

We can then classify languages based on the asymptotic running time required to decide them. For instance, P the class of all languages decidable in *polynomial time*.

Definition: Complexity Class P

P is the class of all the languages $L \subseteq \Sigma^*$ for which there exists a Turing machine $\mathbb{M}$ and a constant $c \in \mathbb{N}$ such that:

- $\mathbb{M}$ decides L;
- $\mathbb{M}$ runs in time $\mathcal{O}(|x|^c)$ on every input $x \in \Sigma^*$.

Problems in P

IS ENVY-FREE

Instance: A set of items $\mathcal{I}$, a set of agents $\mathcal{N} = \{1, \ldots, n\}$, n utility functions $u_i : \mathcal{I} \to \mathbb{N}$, and a allocation $\pi : \mathcal{N} \to 2^{\mathcal{I}}$ such that all items are allocated and there is no overlap between any two agents.

Question: Is the allocation π envy-free, i.e., $\forall i, j \in \mathcal{N}$, $\sum_{o \in \pi(i)} u_i(o) \leq \sum_{o \in \pi(j)} u_i(o)$?

IS MAJORITY-WINNER

Instance: A set of candidates $\mathcal{C}$, a set of agents $\mathcal{N} = \{1, \ldots, n\}$, n ballots $b_i \in \mathcal{C}$, and a candidate $c^\star$

Question: Is $c^\star$ the majority winner of the election, i.e., $\sum_{i \in \mathcal{N}} \mathbb{1}_{b_i = c^\star} \geq {}^{n}/_{2}$?

Problems believed to be Harder than P

Is Pareto-Optimal

Instance: A set of items $\mathcal{I}$, a set of n agents $\mathcal{N}$, n utility functions $u_i : \mathcal{I} \to \mathbb{N}$, and an allocation $\pi : \mathcal{N} \to 2^{\mathcal{I}}$ such that all items are allocated and no item is allocated to several agents (a partition of the items)

Question: Is the allocation π Pareto-optimal, i.e., there is no other allocation π' such that all agents are better off in π' and at least one agent is strictly better off?

Max-Approval Participatory Budgeting

Instance: A set of projects $\mathcal{P}$, a cost function $c : \mathcal{P} \to \mathbb{N}$, a budget limit $B \in \mathbb{N}$, a set of agents $\mathcal{N} = \{1, \ldots, n\}$, n approval ballots $A_i \subseteq \mathcal{P}$ and a parameter $k \in \mathbb{N}$

Question: Is there a budget allocation $\pi \subseteq \mathcal{P}$ with $\sum_{p \in \pi} c(p) \leq B$ and such that $\sum\limits_{i \in \mathcal{N}} |\pi \cap A_i| \geq k$?

What to do with Hard Problems?

We have presented two problems that are *believed* not to be solvable in polynomial time (IS PARETO-OPTIMAL is in coNP and MAX-APPROVAL PARTICIPATORY BUDGETING is in NP). Can we still hope to solve them reasonably well in a reasonable amount of time?

Several direction can be followed to cope with intractability:

- Developing advanced algorithmic techniques to compute *exact solutions*
- Searching for solutions *approximating* the exact solution
- Devising *randomized algorithms* hoping they find the solution

All these approaches would benefit from a more fine-grained analysis of where exactly the complexity is coming from.

$\hookrightarrow$ This is the key idea behind *parameterized complexity*.

2. Parameterized Complexity

Parameterized Computational Problems

Parameterized problems are languages $L \subseteq \Sigma^* \times \mathbb{N}$ of pairs $\langle x, k \rangle$ where $x \in \Sigma^*$ is the main input and $k \in \mathbb{N}$ the parameter (the parameter can equivalently be defined as¨ $\kappa : \Sigma^* \to \mathbb{N}$).

Using only natural numbers as parameters can be seen as a restriction but it is not: Every complex structures can be represented as integers, even lists of parameters.

A parameterized version of MAX-APPROVAL PARTICIPATORY BUDGETING could be:

MAX-APPROVAL PARTICIPATORY BUDGETING

Instance: A set of projects $\mathcal{P}$, a cost function $c : \mathcal{P} \to \mathbb{N}$, a budget limit $B \in \mathbb{N}$, a set of agents $\mathcal{N} = \{1, \ldots, n\}$, n approval ballots $A_i \subseteq \mathcal{P}$ and $k \in \mathbb{N}$

Parameter: k

Question: Is there $\pi \subseteq \mathcal{P}$ with $\sum_{p \in \pi} c(p) \leq B$ and such that $\displaystyle\sum_{i \in \mathcal{N}} |\pi \cap A_i| \geq k$?

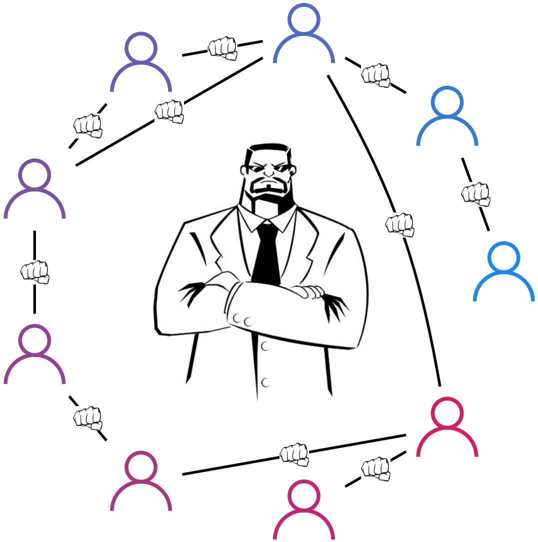
↪ For every problem, *several* interesting parameters can be defined: One could parameterize the problem above by the budget limit B, or the highest cost $\max_{p \in \mathcal{P}} c(p)$.

What to do with the Parameters?

Once we have defined a parameterized problem, we want to analyze its *complexity with respect to the parameter*. For example, given an instance x and a parameter k of the problem, can we solve it in time $\mathcal{O}(2^{k \times |x|})$? In time $\mathcal{O}(|x|^k)$? Or better, in $\mathcal{O}(2^k |x|^{\mathcal{O}(1)})$?

As in the standard complexity theory, we can also define *parameterized complexity classes* and study where the problems belong based on their different parameterizations.

A First Example: the Bar Fight Prevention Problem

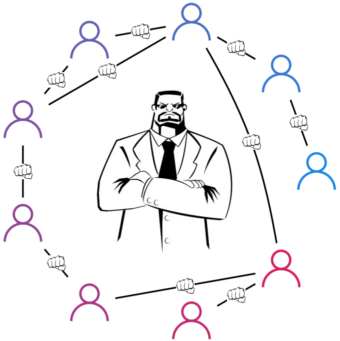


A First Example: the Bar Fight Prevention Problem

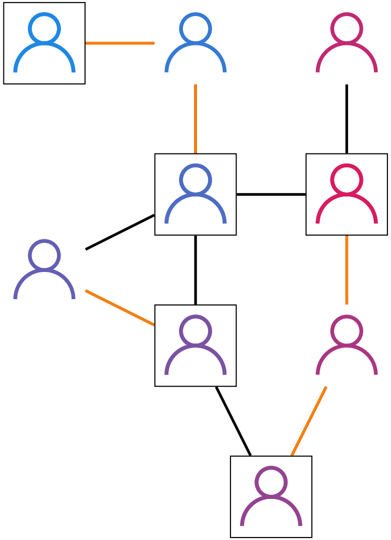
Bar Fight Prevention

Instance:	A set of n regular customers $\mathcal{N}$, a who-fights-who graph $G = \langle \mathcal{N}, E \rangle$ and $k \in \mathbb{I}$
Parameter:	k
Question:	Can we reject k people from the bar to avoid any fights?

↳ How would you solve that? In what running time (based on k and $|\mathcal{N}|$)?



A Clever Algorithm?



Recursive procedure for graph $G = \langle V, E \rangle$, parameter k and current solution S:

- If $\exists (u, v) \in E$ with $\{u, v\} \cap S = \emptyset$:
  - If $k > 0$:
    - Branch on G, $k - 1$ and $S \cup \{u\}$
    - Branch on G, $k - 1$ and $S \cup \{v\}$
  - Otherwise output "no"
- Otherwise output S

↳ What is the running time? $\mathcal{O}(2^k |E|)$!

This shows that the BAR FIGHT PREVENTING problem is solvable in FPT time!

3. Fixed Parameter Tractability

Fixed Parameter Tractability

The previous algorithm has a running time which is polynomial in the input size when the parameter is fixed. All parameterized problems admitting such an algorithm are said to be *fixed parameter tractable* and belong to the complexity class `FPT`.

Definition: Parameterized Complexity Class FPT

A parameterized problem L is in `FPT` if and only if there exists a Turing machine $\mathbb{M}$, a constant $c \in \mathbb{N}$ and a computable function f such that, for all pairs $\langle x, k \rangle \in L$:

- $\mathbb{M}$ runs in time $f(k)|x|^c$ on $\langle x, k \rangle$;
- $\langle x, k \rangle \in L$ if and only if $\mathbb{M}(\langle x, k \rangle) = 1$.

↳ Can you give me some trivial `FPT` problems? Or some parameters for which (almost) everything is `FPT`?

Let's now see some techniques to solve problems in `FPT` time (with interesting parameters)!

Fixed Parameter Tractability

Bounded Search Tree

Bounded Search Tree

The algorithm we designed to solve the BAR FIGHT PREVENTION problem is based on the technique of *bounded search tree*. The general procedure is described below.

DEFINITION: BOUNDED SEARCH TREE

Consider an instance x of a minimization problem, at each step of the process, construct $\ell(x) \in \mathbb{N}$ simpler instances of x called $x_1, \ldots, x_{\ell(x)}$ such that:

- Every solution of x_i corresponds to a solution of x and among all the solutions of x_1 to $x_{\ell(x)}$, one of them corresponds to an optimum solution of x;
- The number $\ell(x)$ is small, e.g., bounded by a function of the parameter;
- For all simpler instance x_i, the value of the parameter is significantly smaller (a constant smaller for instance) than in the original instance.

In the BAR FIGHT PREVENTION problem, we had a straightforward correspondence between a solution for the input and the solutions of the simpler instances; $\ell(x) = 2$ for all instances and the value of the parameter was decrease by 1 at each step: Alles goed!

Fixed Parameter Tractability

Kernelization

Kernelization

What if we could reduce the size of the instance before solving it? That's kernelization!

Definition: Kernel of a Problem

A *data reduction rule* $\varphi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is a function computable in time *polynomial*, mapping an instance $\langle x, k \rangle \in L$ to an *equivalent* instance $\langle x', k' \rangle$, i.e., such that:

$$\langle x, k \rangle \in L \iff \langle x', k' \rangle \in L.$$

A *kernel* of parameterized problem L is a polynomial Turing machine $\mathbb{M}$ that applies a set of data reduction rules on an instance such that:

- Given an instance $\langle x, k \rangle \in L$, $\mathbb{M}$ outputs an *equivalent* instance $\langle x', k' \rangle \in L$;
- There exists a computable function $g : \mathbb{N} \to \mathbb{N}$ depending only on k such that:

$$\sup \{|x'| + k' \mid \langle x', k' \rangle = \mathbb{M}(\langle x, k \rangle), x \in \Sigma^*\} \leq g(k).$$

Kernelization for Efficient and Envy-Free Allocation

ADDITIVE EFFICIENT AND ENVY-FREE ALLOCATION

Instance: A set of items $\mathcal{I}$, a set of n agents $\mathcal{N}$, n utility functions $u_i : \mathcal{I} \to \mathbb{N}$

Parameter: $|\mathcal{I}|$ the number of items

Question: Is there an allocation π (a partition of the items to the agents) that is

1. Pareto-optimal: no one can be made better off with no harm
2. envy-free: $\forall i, j \in \mathcal{N}, \sum_{o \in \pi(i)} u_i(o) \leq \sum_{o \in \pi(j)} u_i(o)$?

Consider the following algorithm:

- If $|\mathcal{N}| \geq |\mathcal{I}|$: return a trivial no instance
- Otherwise return the original instance.

↳ Is this a kernelization? Why?

Kernelization and FPT

PROPOSITION:

A parameterized language $L \subseteq \Sigma^* \times \mathbb{N}$ is in FPT iff it is *decidable* and has a *kernel*.

(Kernel $\Rightarrow$ FPT) Let us describe an FPT Turing machine on $\langle x, k\rangle$: run the kernel (in polynomial time) to obtain $\langle x', k'\rangle$ of size at most $g(k)$. Solve $\langle x', k'\rangle$ (since L is decidable) in time depending on the size of x' which only depends on k.

(FPT $\Rightarrow$ Kernel) Suppose we have an FPT Turing machine $\mathbb{M}$ running in time $f(k) \times |x|^c$. We describe a kernel for L. On input $\langle x, k\rangle$, run $\mathbb{M}$ for at most $|x|^{c+1}$ steps. If it terminates return a trivial yes or no instance based on the output. Otherwise, return $\langle x, k\rangle$. In this case, $f(k) \times |x|^c > |x|^{c+1}$, that is, $f(k) > |x|$ that leads to a kernel of size $f(k) + k$ (which is computable). ■

Fixed Parameter Tractability

Integer Linear Programming

Integer Linear Programming is FPT

Integer Linear Programming (ILP) is a very general framework for solving (optimization) problems. It can be described as follows:

INTEGER LINEAR PROGRAMMING FEASIBILITY

Instance: A matrix $A \in \mathbb{R}^{m,n}$ with m rows and n columns and a vector $b \in \mathbb{R}^m$
Question: Is there a vector $x \in \mathbb{R}^n$ such that $Ax \leq b$?

Interestingly, the problem is in `FPT` when parameterized by the number of variables.

PROPOSITION:

An instance x of the problem INTEGER LINEAR PROGRAMMING FEASIBILITY with n variables can be solved in time $\mathcal{O}(n^{2.5n+o(n)} \times |p|)$.

↳ That is a useful technique to provide `FPT` algorithms!

ILP Formalization for EEF Allocation

Binary Efficient and Envy-Free Allocation	
Instance:	A set of items $\mathcal{I}$, a set of n agents $\mathcal{N}$, n utility functions $u_i : \mathcal{I} \to \{0, 1\}$
Parameter:	n the number of agents
Question:	Is there a Pareto-optimal and envy-free allocation π?

For an item $o \in \mathcal{I}$, let $f_o = (u_i(o))_{i \in \mathcal{N}}$ be its fingerprint. Let $F = \{f_o \mid o \in \mathcal{I}\}$. We describe an allocation through the number x_i^f of items of fingerprint f allocated to agent $i \in \mathcal{N}$.

$$\begin{aligned}
x_i^f &= 0 && \forall f \in F \text{ and } i \in \mathcal{N} \text{ s.t. } f[i] = 0 \\
\sum_{i=1}^{n} x_i^f &= |\{o \in \mathcal{I} \mid f_o = f\}| && \forall f \in F \\
\sum_{f \in F} x_i^f \times f[i] &\geq \sum_{f \in F} x_j^f \times f[i] && \forall i, j \in \mathcal{N}, i \neq j \\
x_i^f &\geq 0 && \forall f \in F, \forall i \in \mathbb{N}
\end{aligned}$$

Since $|F| \leq 2^n$, the number of variable in the program is at most $n2^n$. Binary Efficient and Envy-Free Allocation is thus in `FPT` when parameterized by n.

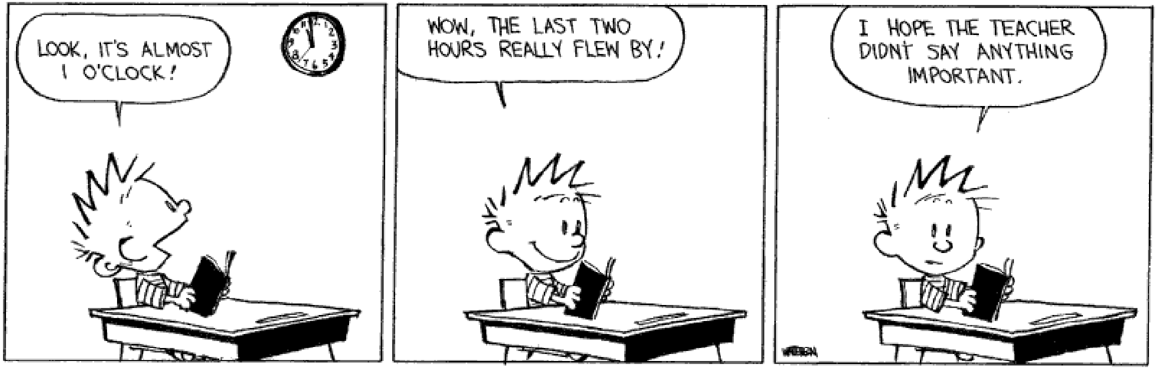
4. Conclusion

Summary and What's Next

Today, we have:

- Introduced the theory of (computational) social choice;
- Gave a brief overview of complexity theory;
- Presented the idea at the core of parameterized complexity theory;
- Seen several techniques to develop fixed-parameter tractable algorithms.

Tomorrow, we'll study other positive algorithmic results, focusing on graphs. Stay tuned!


LOOK, IT'S ALMOST 1 O'CLOCK!
WOW, THE LAST TWO HOURS REALLY FLEW BY!
I HOPE THE TEACHER DIDN'T SAY ANYTHING IMPORTANT.