Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 14: Other Approaches to Intractable Problems

Aalto University
School of Science
Department of Computer Science

Spring 2019

Agenda

- Case studies: MinVC and MaxIS
- Parameterisation
- Exact exponential algorithms
- Other approaches

Solving Hard Problems: Parameterisation

- **There are intractable problems that we don't know how to solve in polynomial time**
  - How to deal with such problems in practice?
- **Today we look at various approaches to this question:**
  - *Parameterised algorithms*
  - *Faster exact exponential algorithms*
  - *Restricted subproblems*
  - *Heuristics*

Case 1: **MinVC on Trees**

Minimum Vertex Cover (MinVC)

- **Instance:** Graph $G = (V, E)$, an integer $k \geq 1$.
- **Question:** Is there a set of vertices C such that $|C| \leq k$ and for all $\{u, v\} \in E$, either $v \in C$ or $u \in C$ (or both)?

- **Trivial algorithm for finding minimum vertex cover:**
  - Try all possible sets $C \subseteq V$
  - Check if C is a vertex cover
  - Running time: $2^n \operatorname{poly}(n)$

- **Consider finding minimum vertex cover on *trees***
  - A graph $G = (V, E)$ is a tree if G does not contain cycles
  - Arbitrarily choose one vertex as the *root*

Case 1: **MinVC on Trees**

- **Greedy algorithm finds an optimal vertex cover on trees:**
  - Any parent u of a leaf v can always be selected to be in an optimal vertex cover
    - Edge $\{u, v\}$ needs to be covered, so either u or v is in any optimal cover
    - v does not cover other edges, so we can always replace it with u
  - We can thus greedily select all parents of the leaves, and remove covered edges
- **Running time is polynomial in the size of input**
- ***Minimum vertex cover on trees*** **is in polynomial time**

Case 2: **Parameterised MinVC**

- **Another perspective on VC: how does the complexity depend on parameter k?**
  - the NP-completeness proof roughly says that the problem is difficult if $k \approx 6|V|/7$
  - What if e.g. $k = O(\log|V|)$?
- **Trivial algorithm for a small minimum vertex cover:**
  - Try all possible sets $C \subseteq V$ with $|C| \leq k$
  - Check if C is a vertex cover
  - Running time: roughly $O(n^k)$

Case 2: **Parameterised MinVC**

Decision algorithm for vertex cover

Input: graph $G = (V, E)$, k

- If $k = 0$ and G has an edge, reject. If $k = 0$ and G has no edges, accept.
- Select an arbitrary edge $e = \{u, v\}$ from the graph.
- Try adding one of the endpoints of e to the vertex cover and recursively call the algorithm to determine if either of the cases can be completed to a vertex cover of size k:
  - Call this algorithm recursively on $(G \setminus v, k-1)$
  - Call this algorithm recursively on $(G \setminus u, k-1)$
- Accept if one of the recursive calls accepts; otherwise reject.

Case 2: **Parameterised MinVC**

- **Algorithm finds a vertex cover of size k if one exists:**
  - Since any vertex cover contains at least one endpoint of each edge, the recursion will have a branch corresponding to any vertex cover of size k
- **Algorithm runs in time $2^k \operatorname{poly}(n)$:**
  - Since the parameter k decreases by one each time the algorithm is called, the depth of the recursion tree is at most k
  - Total size of the recursion tree is thus at most 2^k

Case 3: **Exact Algorithm for MaxIS**

Maximum Independent Set (MaxIS)

- **Instance:** Graph $G = (V, E)$, an integer $k \geq 1$.
- **Question:** Is there a set of vertices I such that $|I| \geq k$ and for all $u, v \in I$, we have that $\{u, v\} \notin E$?

- **Trivial algorithm for finding a MaxIS:**
  - Try all possible sets $C \subseteq V$
  - Check if C is an independent set
  - Running time: $2^n \operatorname{poly}(n)$
  - *Can we do better?*

Case 3: **Exact Algorithm for MaxIS**

- **In the following, $N[v]$ denotes the closed neighbourhood of v, that is,**

$$N[v] = \{v\} \cup \{u \in V \colon \{u, v\} \in E\}$$

An algorithm for independent set

Input: graph $G = (V, E)$, **Output:** size of maximum IS

- If $|V| = 0$, return 0.
- Select the vertex $v \in V$ with smallest degree.
  - Recursively compute the size s_u of the maximum independent set for $G \setminus N[u]$ for all $u \in N[v]$
  - Return $1 + \min_{u \in N[v]} s_u$

Case 3: **Exact Algorithm for MaxIS**

- **Algorithm finds the size of the maximum IS:**
  - Recursion tries all possible choices
  - For any vertex v, at least one vertex in $N[v]$ is in any maximum independent set

- **Complexity analysis:**
  - Size of the recursion tree is given by the recurrence

$$T(n) \leq T(n - \deg(v) - 1) + \sum_{u \in N[v]} T(n - \deg(u) - 1)\,,$$

where v is the vertex chosen by the algorithm

Case 3: **Exact Algorithm for MaxIS**

- **Analysis of the recurrence:**
  - Since the algorithm picks the vertex with smallest degree, we have $\deg(v) \leq \deg(u)$ for all $u \in N[v]$
  - Thus, we have $T(n) \leq (\deg(v)+1)T(n-\deg(v)-1)$
  - Writing $s = \deg(v)+1$, we have

$$\begin{aligned} T(n) \leq sT(n-s) &\leq 1+s+s^2+\cdots+s^{n/s} \\ &\leq \frac{1-s^{n/s+1}}{1-s} = s^{n/s}\operatorname{poly}(n,s) \end{aligned}$$

  - $s^{n/s}$ is maximised by $s = 3$ (for integers)
- **MaxIS can be solved in time** $3^{n/3}\operatorname{poly}(n) \approx 1.44^n \operatorname{poly}(n)$

Parameterised Problems

Definition

Parameterisation and parameterised problems

- A *parameterisation* is a polynomial-time computable function $k\colon \{0,1\}^* \to \mathbb{N}$.
- A *parameterised problem* is a pair (L,k), where $L \subseteq \{0,1\}^*$ is a language and k is a parameterisation.

- **Parameter can describe any aspect of the instance**
  - Simple examples: number of vertices, number of edges
  - Define parameter to be e.g. 0 for non-valid instances
- **Basic approach of parameterised complexity:** *study complexity in terms of different parameters*

Parameterised Problems

- **Natural parameter for optimisation-style problems:**
  - *size of the solution*

Parameterised Vertex Cover

- **Instance:** Graph $G = (V, E)$, an integer $k \geq 1$.
- **Parameter:** k.
- **Question:** Is there a set of vertices C such that $|C| \leq k$ and for all $\{u, v\} \in E$, either $v \in C$ or $u \in C$ (or both)?

Fixed-parameter Tractability

Definition

A parameterised problems (L, k) is *fixed-parameter tractable (FPT)* if there is a computable function $f : \mathbb{N} \to \mathbb{N}$, polynomial p and a Turing machine M such that M decides L and runs in time

$$f(k(x)) \cdot p(|x|)$$

for all $x \in \{0,1\}^*$

- **Fixed-parameter algorithm isolates the non-polynomial behaviour to the parameter**
  - For constant parameter, the problem is polynomial-time solvable

Fixed-parameter Tractability

- **Some fixed-parameter tractable problems**
  - *Vertex cover* parameterised by the solution size
  - *k-path* parameterised by k
  - *CNF-SAT* parameterised by the number of variables
- **Not FPT unless** $\mathrm{P} = \mathrm{NP}$
  - *Colouring* parameterised by the number of colours
- **What about *independent set* (parameterised by solution size)?**

FPT Reductions

Definition

An *FPT reduction* from a parameterised problem (L, k) to a parameterised problem (L', k') is a mapping $R\colon \{0,1\}^* \to \{0,1\}^*$ such that

- $x \in L$ if and only if $R(x) \in L'$,
- R is computable in time $f(k(x))\,\mathrm{poly}(n)$, and
- there is a computable function $g\colon \mathbb{N} \to \mathbb{N}$ such that $k'(R(x)) \leq g(k(x))$ for all $x \in \{0,1\}^*$.

- **FPT reductions preserve fixed-parameter tractability**
  - Theory of fixed-parameter intractability is based on FPT reductions

W[1] and W[2]

- **There is a hierarchy of classes $W[1], W[2], \ldots$ of parameterised problems believed not to be FPT**

- **Exact definition of class $W[t]$ is somewhat technical**
- **Complete problems for $W[1]$ under FPT reductions:**
  - *Independent set* parameterised by the solution size
  - Deciding if a nondeterministic *single-tape* Turing machine accepts the empty string in k steps, parameterised by k
- **Complete problems for $W[2]$ under FPT reductions:**
  - *Dominating set* parameterised by the solution size
  - Deciding if a nondeterministic *multi-tape* Turing machine accepts the empty string in k steps, parameterised by k

XP

Definition

A parameterised problems (L, k) is in *class XP* if there is a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant c and a Turing machine M such that M decides L and runs in time $c \cdot |x|^{f(k(x))}$ for all $x \in \{0, 1\}^*$

- **Problems in XP have polynomial-time solutions for constant parameter**
  - The *degree* of the polynomial can grow very quickly
- FPT $\subseteq W[1] \subset W[2] \subseteq \cdots \subseteq$ XP

Exact Exponential Algorithm

- **Assuming** P $\neq$ NP**, we cannot solve certain problems in polynomial time**
  - Can we still solve them fast enough?
  - 1.0001^n is better than n^{100} in practice
  - *Warning:* many ‘fast’ exact algorithms are not really practical
- **Exact exponential algorithmics studies *less bad* exponential algorithms**

Exact Exponential Algorithm

- **Examples of exact exponential algorithms:**
  - *Maximum independent set* can be solved in time $O(1.1996^n)$
  - *Undirected Hamiltonian cycle* can be solved in time $O(1.657^n)$
  - *TSP* can be solved in time $O(2^n n^2)$
- **Typical questions:**
  - What is the best δ such that we can solve a given problem in time $O(\delta^n)$?
  - Can we solve a given problem in subexponential time? (taken to mean $2^{o(1)}$ in this context)

Exponential Time Hypotheses

- **For CNF-SAT, the best algorithm has complexity about $2^n \operatorname{poly}(n,m)$ (n variables, m clauses)**
  - Is there an $O\big((2-\varepsilon)^n\big)$ algorithm?
  - Is there a subexponential algorithm?

- **This gives raise to two hypotheses:**
  - *Exponential time hypothesis (ETH):*
    no $2^{o(n)}$ algorithm for CNF-SAT
  - *Strong exponential time hypothesis (SETH):*
    no $O\big((2-\varepsilon)^n\big)$ algorithm for CNF-SAT for any $\varepsilon > 0$
  - Not necessarily widely believed
  - Can still be used to prove lower bounds for other problems via fine-grained reductions

Restricted Subproblems

- **For understanding NP-hard problem, a common solution is to look at *restricted subproblems***
  - **Example:** TSP, Metric TSP, Euclidean TSP
  - Subproblems may be easier than the problem itself

- **For graph problems, this often means considering restricted input graphs:**
  - *Trees:* many common problems are polynomial-time solvable on trees, but not everything
  - *Planar graphs:* graphs that can be drawn on a plane without edges crossing
  - *Bounded treewidth graphs:* generalisation of trees, important in fixed-parameter complexity

Heuristics

- ***Heuristics*** **are algorithmic techniques without theoretical guarantees**
  - Common outside theoretical computer science
  - However, often useful in practice
- **Heuristics can fail in many ways:**
  - No running time guarantees
  - May not find an optimal solution, just a feasible one
  - No approximation guarantees
  - May fail to find a solution when one exists, or fail to detect that solution does not exist

Heuristics

- **Heuristics are not necessarily incompatible with theory**
  - Ideally: *theoretical guarantees* + *heuristics*
- **Case: *SAT solvers* for CNF-SAT**
  - Modern SAT solvers use heuristic algorithms to find a solution quickly if one exists
  - Since a solutions can be verified, yes-instances can often be solved very quickly
  - Does not necessarily help with no-instances
- **This makes SAT solvers a powerful tool in practical algorithmics when a reduction to CNF-SAT is feasible**

Lecture 14: Summary

- Fixed-parameter tractability
- W[1]-hard and W[2]-hard problems
- Exact Exponential Algorithms
- Restricted subproblems
- Heuristics