Introduction to solving intractable problems

Serge Gaspers

Contents

1 Algorithms for NP-hard problems

Central question

P vs. NP

NP-hard problems

- no known polynomial time algorithm for any NP-hard problem
- belief: P $\neq$ NP
- What to do when facing an NP-hard problem?

Example problem

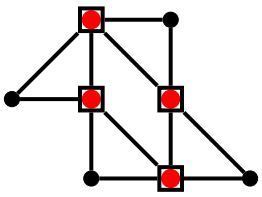
Monitoring a power grid
Tammy is responsible for fault detection on the power grid of an energy company.She has access to k monitoring devices. Each one can be placed on a node of the electrical grid and can monitor the power lines that are connected to this node.Tammy's objective is to place the monitoring devices in such a way that each power line is monitored by at least one monitoring device.

Let us first give an abstraction of this problem and formulate it as a decision problem for graphs.

Example problem: Vertex Cover
A *vertex cover* in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that every edge of G has an endpoint in S.

Vertex Cover	
Input:	Graph G, integer k
Question:	Does G have a vertex cover of size k?

Note: Vertex Cover is NP-complete.



Coping with NP-hardness

- Approximation algorithms
  - There is a polynomial-time algorithm, which, given a graph G, finds a vertex cover of G of size at most $2 \cdot \mathrm{OPT}$, where OPT is the size of a smallest vertex cover of G.
- Exact exponential time algorithms
  - There is an algorithm solving Vertex Cover in time $O(1.1970^n)$, where $n = |V|$ (Xiao and Nagamochi, 2017).
- Fixed parameter algorithms
  - There is an algorithm solving Vertex Cover in time $O(1.2738^k + kn)$ (Chen, Kanj, and Xia, 2010).
- Heuristics
  - The COVER heuristic (COVer Edges Randomly) finds a smaller vertex cover than state-of-the-art heuristics on a suite of hard benchmark instances (Richter, Helmert, and Gretton, 2007).
- Restricting the inputs
  - Vertex Cover can be solved in polynomial time on bipartite graphs, trees, interval graphs, etc. (Golumbic, 2004).
- Quantum algorithms?
  - Not believed to solve NP-hard problems in polynomial time (Aaronson, 2005). Quadratic speedup possible in some cases.

Aims of this course

Design and analyze algorithms for NP-hard problems.

We focus on algorithms that solve NP-hard problems *exactly* and analyze their *worst case running time*.

2 Exponential Time Algorithms

Running times

Worst case running time of an algorithm.

- An algorithm is *polynomial* if $\exists c \in \mathbb{N}$ such that the algorithm solves every instance in time $O(n^c)$, where n is the size of the instance. Also: $n^{O(1)}$ or $\mathrm{poly}(n)$.
- *quasi-polynomial*: $2^{O(\log^c n)}$, $c \in O(1)$
- *sub-exponential*: $2^{o(n)}$
- *exponential*: $2^{\mathrm{poly}(n)}$
- *double-exponential*: $2^{2^{\mathrm{poly}(n)}}$

O^*-notation ignores polynomial factors in the input size:

$$O^*(f(n)) \equiv O(f(n) \cdot \mathsf{poly}(n))$$
$$O^*(f(k)) \equiv O(f(k) \cdot \mathsf{poly}(n))$$

Brute-force algorithms for NP-hard problems

Theorem 1. *Every problem in* NP *can be solved in exponential time.*

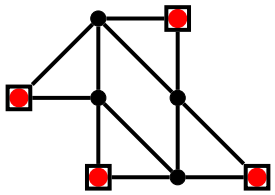
For a proof, see the lecture on NP-completeness.

Three main categories for NP-complete problems

- Subset problems
- Permutation problems
- Partition problems

Subset Problem: Independent Set
An *independent set* in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that the vertices in S are pairwise non-adjacent in G.

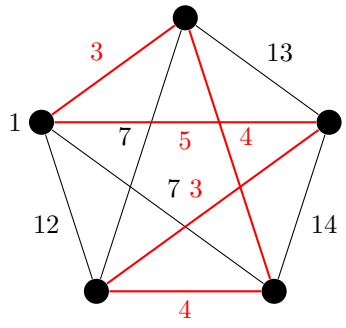
Independent Set	
Input:	Graph G, integer k
Question:	Does G have an independent set of size k?



Brute-force: $O^*(2^n)$, where $n = |V(G)|$

Permutation Problem: Traveling SalesPerson

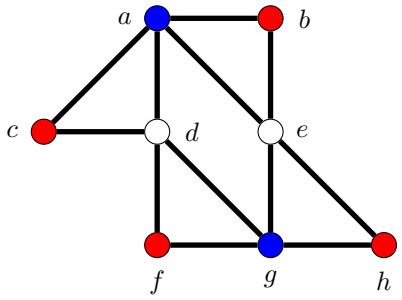
Traveling SalesPerson (TSP)	
Input:	a set of n cities, the distance $d(i, j) \in \mathbb{N}$ between every two cities i and j, integer k
Question:	Is there a permutation of the cities (a *tour*) such that the total distance when traveling from city to city in the specified order, and returning back to the origin, is at most k?



Brute-force: $O^*(n!) \subseteq 2^{O(n \log n)}$

Partition Problem: Coloring
A *k-coloring* of a graph $G = (V, E)$ is a function $f : V \to \{1, 2, \ldots, k\}$ assigning colors to V such that no two adjacent vertices receive the same color.

Coloring	
Input:	Graph G, integer k
Question:	Does G have a k-coloring?



Brute-force: $O^*(k^n)$, where $n = |V(G)|$

Exponential Time Algorithms

- natural question in Algorithms: design faster (worst-case analysis) algorithms for problems

- might lead to practical algorithms
  - for small instances
    - you don't want to design software where your client/boss can find with better solutions *by hand* than your software
  - subroutines for
    - (sub)exponential time approximation algorithms
    - randomized algorithms with expected polynomial run time

Solve an NP-hard problem

- exhaustive search
  - trivial method
  - try all candidate solutions (certificates) for a ground set on n elements
  - running times for problems in NP
    - Subset Problems: $O^*(2^n)$
    - Permutation Problems: $O^*(n!)$
    - Partition Problems: $O^*(c^{n \log n})$
- faster exact algorithms
  - for some problems, it is possible to obtain provably faster algorithms
  - running times $O(1.0836^n), O(1.4689^n), O(1.9977^n)$

Exponential Time Algorithms in Practice

- How large are the instances one can solve in practice?

Available time nb. of operations	1 s 2^{38}	1 min $\sim 2^{44}$	1 hour $\sim 2^{50}$	3 days $\sim 2^{56}$	6 months $\sim 2^{62}$
n^5	194	446	1,024	2,352	5,404
n^{10}	14	21	32	49	74
1.05^n	540	625	711	796	881
1.1^n	276	320	364	407	451
1.5^n	65	75	85	96	106
2^n	38	44	50	56	62
5^n	16	19	22	24	27
$n!$	14	16	17	19	20

Note: Intel Core i7-8086K executes $\sim 2^{38}$ instructions per second at 5 GHz.

> "For every polynomial-time algorithm you have, there is an exponential algorithm that I would rather run."
>
> – Alan Perlis (1922-1990, programming languages, 1st recipient of Turing Award)

Hardware vs. Algorithms

- Suppose a 2^n algorithm enables us to solve instances up to size x
- Faster processors
  - processor speed doubles after 18–24 months (Moore's law)
  - can solve instances up to size $x + 1$
- Faster algorithm
  - design an $O^*(2^{n/2}) \subseteq O(1.4143^n)$ time algorithm
  - can solve instances up to size $2 \cdot x$

3 Parameterized Complexity

A story
A computer scientist meets a biologist ... The biologist has performed n experiments. Unfortunately, the data obtained from these experiments has some conflicts. He suspects that a small number k of experiments have gone wrong, and he would like to detect whether removing k experiments can solve all the conflicts.

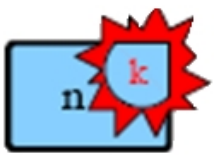
Eliminating conflicts from experiments
$n = 1000$ experiments, $k = 20$ experiments failed

Running Time		
Theoretical	Number of Instructions	Real
2^n	$1.07 \cdot 10^{301}$	$4.941 \cdot 10^{282}$ years
n^k	10^{60}	$4.611 \cdot 10^{41}$ years
$2^k \cdot n$	$1.05 \cdot 10^9$	0.01526 seconds

Notes

- We assume that 2^{36} instructions are carried out per second.
- The Big Bang happened roughly $13.5 \cdot 10^9$ years ago.

Goal of Parameterized Complexity
Confine the combinatorial explosion to a parameter k.



For which problem–parameter combinations can we find algorithms with running times of the form

$$f(k) \cdot n^{O(1)},$$

where the f is a computable function independent of the input size n?

Examples of Parameters

A Parameterized Problem	
Input:	an instance of the problem
Parameter:	a parameter k
Question:	a YES/NO question about the instance and the parameter

- A parameter can be
  - input size (trivial parameterization)
  - solution size
  - related to the structure of the input (maximum degree, treewidth, branchwidth, genus, ...)
  - etc.

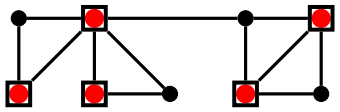
Main Complexity Classes
P: class of problems that can be solved in time $n^{O(1)}$
FPT: class of problems that can be solved in time $f(k) \cdot n^{O(1)}$
W[·]: parameterized intractability classes
XP: class of problems that can be solved in time $f(k) \cdot n^{g(k)}$

$$\text{P} \subseteq \text{FPT} \subseteq \text{W[1]} \subseteq \text{W[2]} \cdots \subseteq \text{W}[P] \subseteq \text{XP}$$

Known: If FPT = W[1], then the Exponential Time Hypothesis fails, i.e. 3-SAT can be solved in time $2^{o(n)}$.

3.1 FPT Algorithm for Vertex Cover

VERTEX COVER (VC)	
Input:	A graph $G = (V, E)$ on n vertices, an integer k
Parameter:	k
Question:	Is there a set of vertices $C \subseteq V$ of size at most k such that every edge has at least one endpoint in C?



3.2 Algorithms for Vertex Cover

Brute Force Algorithms

- $2^n \cdot n^{O(1)}$ not FPT
- $n^k \cdot n^{O(1)}$ not FPT

An FPT Algorithm

Algorithm vc1(G, k);

```
1 if E = ∅ then                                   // all edges are covered
2     return Yes
3 else if k ≤ 0 then                              // we cannot select any vertex
4     return No
5 else
6     Select an edge uv ∈ E;
7     return vc1(G − u, k − 1) ∨ vc1(G − v, k − 1)
```

Running Time Analysis

- Let us look at an arbitrary execution of the algorithm.
- Recursive calls form a *search tree* T
  - with depth $\leq k$
  - where each node has ≤ 2 children
- $\Rightarrow T$ has $\leq 2^k$ leaves and $\leq 2^k - 1$ internal nodes
- at each node the algorithm spends time $n^{O(1)}$
- The running time is $O^*(2^k)$

A faster FPT Algorithm

Algorithm vc2(G, k);

```
1 if E = ∅ then                                          // all edges are covered
2 |  return Yes
3 else if k ≤ 0 then                                     // we used too many vertices
4 |  return No
5 else if Δ(G) ≤ 2 then                                  // G has maximum degree ≤ 2
6 |  Solve the problem in polynomial time;
7 else
8 |  Select a vertex v of maximum degree;
9 |  return vc2(G − v, k − 1) ∨ vc2(G − N[v], k − d(v))
```

Running time analysis of vc2

- Number of leaves of the search tree:

$$\begin{aligned} T(k) &\leq T(k-1) + T(k-3) \\ x^k &\leq x^{k-1} + x^{k-3} \\ x^3 - x^2 - 1 &\leq 0 \end{aligned}$$

- The equation $x^3 - x^2 - 1 = 0$ has a unique positive real solution: $x \approx 1.4655\ldots$
- Running time: $1.4656^k \cdot n^{O(1)}$

4 Further Reading

- Exponential-time algorithms
  - Chapter 1, *Introduction*, in (Fomin and Kratsch, 2010).
  - Survey on exponential-time algorithms (Woeginger, 2001).
  - Chapter 1, *Introduction*, in (Gaspers, 2010).
- Parameterized Complexity
  - Chapter 1, *Introduction*, in (Cygan et al., 2015)
  - Chapter 2, *The Basic Definitions*, in (Downey and Fellows, 2013)
  - Chapter I, *Foundations*, in (Niedermeier, 2006)
  - *Preface* in (Flum and Grohe, 2006)

References

Scott Aaronson (2005). "Guest Column: NP-complete problems and physical reality". In: *SIGACT News* 36.1, pp. 30–52. DOI: 10.1145/1052796.1052804.

Jianer Chen, Iyad A. Kanj, and Ge Xia (2010). "Improved upper bounds for vertex cover". In: *Theoretical Computer Science* 411.40-42, pp. 3736–3756. DOI: 10.1016/j.tcs.2010.06.026.

Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh (2015). *Parameterized Algorithms*. Springer. DOI: 10.1007/978-3-319-21275-3.

Rodney G. Downey and Michael R. Fellows (2013). *Fundamentals of Parameterized Complexity*. Springer. DOI: 10.1007/978-1-4471-5559-1.

Jörg Flum and Martin Grohe (2006). *Parameterized Complexity Theory*. Springer. DOI: 10.1007/3-540-29953-X.

Fedor V. Fomin and Dieter Kratsch (2010). *Exact Exponential Algorithms*. Springer. DOI: 10.1007/978-3-642-16533-7.

Serge Gaspers (2010). *Exponential Time Algorithms: Structures, Measures, and Bounds*. VDM Verlag Dr. Mueller.

Martin Charles Golumbic (2004). *Algorithmic Graph Theory and Perfect Graphs*. Elsevier.

Rolf Niedermeier (2006). *Invitation to Fixed Parameter Algorithms*. Oxford University Press. DOI: 10.1093/ACPROF:OSO/9780198566076.001.0001.

Silvia Richter, Malte Helmert, and Charles Gretton (2007). "A Stochastic Local Search Approach to Vertex Cover". In: *Proceedings of the 30th Annual German Conference on Artificial Intelligence (KI 2007)*. Vol. 4667. Lecture Notes in Computer Science. Springer, pp. 412–426. DOI: 10.1007/978-3-540-74565-5_31.

Gerhard J. Woeginger (2001). "Exact Algorithms for NP-Hard Problems: A Survey". In: *Combinatorial Optimization*, pp. 185–208. DOI: 10.1007/3-540-36478-1_17.

Mingyu Xiao and Hiroshi Nagamochi (2017). "Exact algorithms for maximum independent set". In: *Information and Computation* 255, pp. 126–146. DOI: 10.1016/j.ic.2017.06.001.

NP-completeness

Serge Gaspers

Contents

1 Overview

Polynomial-time algorithm
Polynomial-time algorithm: There exists a constant $c \in \mathbb{N}$ such that the algorithm has (worst-case) running-time $O(n^c)$, where n is the size of the input.

Example
Polynomial: n; $n^2 \log_2 n$; n^3; n^{20} Super-polynomial: $n^{\log_2 n}$; $2^{\sqrt{n}}$; 1.001^n; 2^n; $n!$

Central Question
Which computational problems have polynomial-time algorithms?

Million-dollar question
Intriguing class of problems: NP-complete problems.

NP-complete problems
It is unknown whether NP-complete problems have polynomial-time algorithms.

- A polynomial-time algorithm for one NP-complete problem would imply polynomial-time algorithms for all problems in NP.

Gerhard Woeginger's P vs NP page: `http://www.win.tue.nl/~gwoegi/P-versus-NP.htm`

Polynomial vs. NP-complete
Polynomial

- Shortest Path: Given a graph G, two vertices a and b of G, and an integer k, does G have a simple a–b-path of length at most k?
- Euler Tour: Given a graph G, does G have a cycle that traverses each edge of G exactly once?
- 2-CNF SAT: Given a propositional formula F in 2-CNF, is F satisfiable?*A k-CNF formula is a conjunction (AND) of clauses, and each clause is a disjunction (OR) of at most k literals, which are negated or unnegated Boolean variables.*

NP-complete

- LONGEST PATH: Given a graph G and an integer k, does G have a simple path of length at least k?
- HAMILTONIAN CYCLE: Given a graph G, does G have a simple cycle that visits each vertex of G?
- 3-CNF SAT: Given a propositional formula F in 3-CNF, is F satisfiable? *Example:* $(x \vee \neg y \vee z) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z)$.

Overview

What's next?

- Formally define P, NP, and NP-complete (NPC)
- (New) skill: show that a problem is NP-complete

2 Turing Machines, P, and NP

Decision problems and Encodings

<Name of Decision Problem> Input: <What constitutes an instance> Question: <*Yes/No* question>

We want to know which decision problems can be solved in polynomial time – polynomial in the *size of the input* n.

- Assume a "reasonable" encoding of the input
- Many encodings are polynomial-time equivalent; i.e., one encoding can be computed from another in polynomial time.
- Important exception: unary versus binary encoding of integers.
  - An integer x takes $\lceil \log_2 x \rceil$ bits in binary and $x = 2^{\log_2 x}$ bits in unary.

Formal-language framework

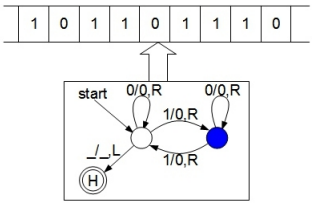
We can view decision problems as languages.

- Alphabet Σ: finite set of symbols. W.l.o.g., $\Sigma = \{0, 1\}$
- Language L over Σ: set of strings made with symbols from Σ: $L \subseteq \Sigma^*$
- Fix an encoding of instances of a decision problem Π into Σ
- Define the language $L_\Pi \subseteq \Sigma^*$ such that

$$x \in L_\Pi \Leftrightarrow x \text{ is a Yes-instance for } \Pi$$

Non-deterministic Turing Machine (NTM)

- *input word* $x \in \Sigma^*$ placed on an *infinite tape* (memory)
- read-write head initially placed on the first symbol of x
- computation step: if the machine is in state s and reads a, it can move into state s', writing b, and moving the head into direction $D \in \{L, R\}$ if $((s, a), (s', b, D)) \in \delta$.



- Q: finite, non-empty set of states
- Γ: finite, non-empty set of tape symbols
- $\_ \in \Gamma$: blank symbol (the only symbol allowed to occur on the tape infinitely often)
- $\Sigma \subseteq \Gamma \setminus \{b\}$: set of input symbols
- $q_0 \in Q$: start state
- $A \subseteq Q$: set of accepting (final) states
- $\delta \subseteq (Q \setminus A \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$: transition relation, where L stands for a move to the left and R for a move to the right.

Accepted Language

Definition 1. A NTM *accepts* a word $x \in \Sigma^*$ if there exists a sequence of computation steps starting in the start state and ending in an accept state.

Definition 2. The language *accepted* by an NTM is the set of words it accepts.

Video

The LEGO Turing Machine `https://www.youtube.com/watch?v=cYw2ewoO6c4`

Acceptance in polynomial time

Definition 3. A language L is *accepted in polynomial time* by an NTM M if

- L is accepted by M, and
- there is a constant k such that for any word $x \in L$, the NTM M accepts x in $O(|x|^k)$ computation steps.

Deterministic Turing Machine

Definition 4. A *Deterministic Turing Machine (DTM)* is a Non-deterministic Turing Machine where the transition relation contains at most one tuple $((s, a), (\cdot, \cdot, \cdot))$ for each $s \in Q \setminus A$ and $a \in \Gamma$.

The transition relation δ can be viewed as a function $\delta : Q \setminus A \times \Gamma \to Q \times \Gamma \times \{L, R\}$. $\Rightarrow$ For a given input word $x \in \Sigma^*$, there is exactly one sequence of computation steps starting in the start state.

DTM equivalents

Many computational models are polynomial-time equivalent to DTMs:

- Random Access Machine (RAM, used for algorithms in the textbook)
- variants of Turing machines (multiple tapes, infinite only in one direction, ...)
- ...

P and NP

Definition 5 (P)**.** $\mathrm{P} = \{L \subseteq \Sigma^* :$ there is a DTM accepting L in polynomial time$\}$

Definition 6 (NP)**.** $\mathrm{NP} = \{L \subseteq \Sigma^* :$ there is a NTM accepting L in polynomial time$\}$

Definition 7 (coNP)**.** $\mathrm{coNP} = \{L \subseteq \Sigma^* : \Sigma^* \setminus L \in \mathrm{NP}\}$

coP?

Theorem 8. *If $L \in$ P, then there is a polynomial-time DTM that halts in an accepting state on every word in L and it halts in a non-accepting state on every word not in L.*

Proof sketch. Suppose $L \in$ P. By the definition of P, there is a DTM M that accepts L in polynomial time. Idea: design a DTM M' that simulates M for $c \cdot n^k$ steps, where $c \cdot n^k$ is the running time of M and transitions to a non-accepting state if M does not halt in an accepting state. (Note that this proof is nonconstructive: we might not know the running time of M.) □

NP and certificates

Non-deterministic choices
A NTM for an NP-language L makes a polynomial number of non-deterministic choices on input $x \in L$. We can encode these non-deterministic choices into a *certificate* c, which is a polynomial-length word. Now, there exists a DTM, which, given x and c, verifies that $x \in L$ in polynomial time.

Thus, $L \in$ NP iff there is a DTM V and for each $x \in L$ there exists a polynomial-length certificate c such that $V(x, c) = 1$, but $V(y, \cdot) = 0$ for each $y \notin L$.

CNF-SAT is in NP

- A *CNF formula* is a propositional formula in conjunctive normal form: a conjunction (AND) of clauses; each clause is a disjunction (OR) of literals; each literal is a negated or unnegated Boolean variable.
- An assignment $\alpha : \mathsf{var}(F) \to \{0, 1\}$ satisfies a clause C if it sets a literal of C to true, and it satisfies F if it satisfies all clauses in F.

CNF-SAT	
Input:	CNF formula F
Question:	Does F have a satisfying assignment?

Example: $(x \vee \neg y \vee z) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z)$.

Lemma 9. *CNF-SAT* $\in$ NP.

Proof. Certificate: assignment α to the variables. Given a certificate, it can be checked in polynomial time whether all clauses are satisfied. □

Brute-force algorithms for problems in NP

Theorem 10. *Every problem in* NP *can be solved in exponential time.*

Proof. Let Π be an arbitrary problem in NP. [Use certificate-based definition of NP] We know that $\exists$ a polynomial p and a polynomial-time verification algorithm V such that:

- for every $x \in \Pi$ (i.e., every YES-instance for Π) $\exists$ string $c \in \{0, 1\}^*$, $|c| \leq p(|x|)$, such that $V(x, c) = 1$, and
- for every $x \notin \Pi$ (i.e., every NO-instance for Π) and every string $c \in \{0, 1\}^*$, $V(x, c) = 0$.

Now, we can prove that there exists an exponential-time algorithm for Π with input x:

- For each string $c \in \{0, 1\}^*$ with $|c| \leq p(|x|)$, evaluate $V(x, c)$ and return YES if $V(x, c) = 1$.
- Return NO.

Running time: $2^{p(|x|)} \cdot n^{O(1)} \subseteq 2^{O(2 \cdot p(|x|))} = 2^{O(p(|x|))}$, but non-constructive. □

3 Reductions and NP-completeness

Polynomial-time reduction

Definition 11. A language L_1 is *polynomial-time reducible* to a language L_2, written $L_1 \leq_P L_2$, if there exists a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$,

$$x \in L_1 \Leftrightarrow f(x) \in L_2.$$

A polynomial time algorithm computing f is a *reduction algorithm.*

New polynomial-time algorithms via reductions

Lemma 12. *If $L_1, L_2 \in \Sigma^*$ are languages such that $L_1 \leq_P L_2$, then $L_2 \in$ P implies $L_1 \in$ P.*

NP-completeness

Definition 13 (NP-hard)**.** A language $L \subseteq \Sigma^*$ is NP-*hard* if

$$L' \leq_P L \text{ for every } L' \in \text{NP}.$$

Definition 14 (NP-complete)**.** A language $L \subseteq \Sigma^*$ is NP-*complete* (in NPC) if

1. $L \in$ NP, and
2. L is NP-hard.

A first NP-complete problem

Theorem 15. *CNF-SAT is* NP*-complete.*

Proved by encoding NTMs into SAT (Cook, 1971; Levin, 1973) and then CNF-SAT (Karp, 1972).

Proving NP-completeness

Lemma 16. *If L is a language such that $L' \leq_P L$ for some $L' \in$ NPC, then L is* NP*-hard. If, in addition, $L \in$ NP, then $L \in$ NPC.*

Proof. For all $L'' \in$ NP, we have $L'' \leq_P L' \leq_P L$. By transitivity, we have $L'' \leq_P L$. Thus, L is NP-hard. □

Proving NP-completeness (2)

Method to prove that a language L is NP-complete:

1. Prove $L \in$ NP
2. Prove L is NP-hard.
   - Select a known NP-complete language L'.
   - Describe an algorithm that computes a function f mapping every instance $x \in \Sigma^*$ of L' to an instance $f(x)$ of L.
   - Prove that $x \in L' \Leftrightarrow f(x) \in L$ for all $x \in \Sigma^*$.
   - Prove that the algorithm computing f runs in polynomial time.

4 NP-complete problems

3-CNF SAT is NP-hard

Theorem 17. *3-CNF SAT is* NP*-complete.*

Proof. 3-CNF SAT is in NP, since it is a special case of CNF-SAT. To show that 3-CNF SAT is NP-hard, we give a polynomial reduction from CNF-SAT. Let F be a CNF formula. The reduction algorithm constructs a 3-CNF formula F' as follows. For each clause C in F:

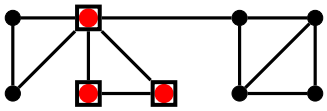
- If C has at most 3 literals, then copy C into F'.
- Otherwise, denote $C = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k)$. Create $k-3$ new variables $y_1, \ldots, y_{k-3}$, and add the clauses $(\ell_1 \vee \ell_2 \vee y_1), (\neg y_1 \vee \ell_3 \vee y_2), (\neg y_2 \vee \ell_4 \vee y_3), \ldots, (\neg y_{k-3} \vee \ell_{k-1} \vee \ell_k)$.

Show that F is satisfiable $\Leftrightarrow$ F' is satisfiable. Show that F' can be computed in polynomial time (trivial; use a RAM). □

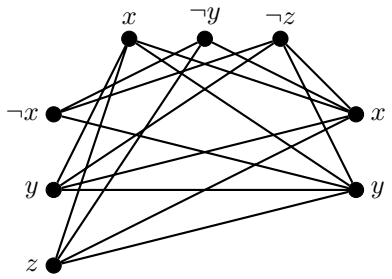
Clique

A *clique* in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that every two vertices of S are adjacent in G.

CLIQUE	
Input:	Graph G, integer k
Question:	Does G have a clique of size k?



Theorem 18. CLIQUE *is* NP*-complete.*



$$(\neg x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (x \vee y)$$

- CLIQUE is in NP
- Let $F = C_1 \wedge C_2 \wedge \ldots C_k$ be a 3-CNF formula
- Construct a graph G that has a clique of size k iff F is satisfiable
- For each clause $C_r = (\ell_1^r \vee \cdots \vee \ell_w^r)$, $1 \le r \le k$, create w new vertices $v_1^r, \ldots, v_w^r$
- Add an edge between v_i^r and v_j^s if

$$r \neq s \qquad \text{and}$$
$$\ell_i^r \neq \neg \ell_j^s \qquad \text{where } \neg\neg x = x.$$

- Check correctness and polynomial running time
- Correctness: F has a satisfying assignment iff G has a clique of size k.

- ($\Rightarrow$): Let α be a sat. assignment for F. For each clause C_r, choose a literal ℓ_i^r with $\alpha(\ell_i^r) = 1$, and denote by s^r the corresponding vertex in G. Now, $\{s^r : 1 \leq r \leq k\}$ is a clique of size k in G since $\alpha(x) \neq \alpha(\neg x)$.
- ($\Leftarrow$): Let S be a clique of size k in G. Then, S contains exactly one vertex $s_r \in \{v_1^r, \ldots, v_w^r\}$ for each $r \in \{1, \ldots, k\}$. Denote by l^r the corresponding literal. Now, for any r, r', it is not the case that $l_r = \neg l_{r'}$. Therefore, there is an assignment α to $\mathsf{var}(F)$ such that $\alpha(l_r) = 1$ for each $r \in \{1, \ldots, k\}$ and α satisfies F.

Vertex Cover

A *vertex cover* in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that every edge of G has an endpoint in S.

Vertex Cover	
Input:	Graph G, integer k
Question:	Does G have a vertex cover of size k?

Theorem 19. Vertex Cover *is* NP*-complete.*

The proof is left as an exercise.

Hamiltonian Cycle

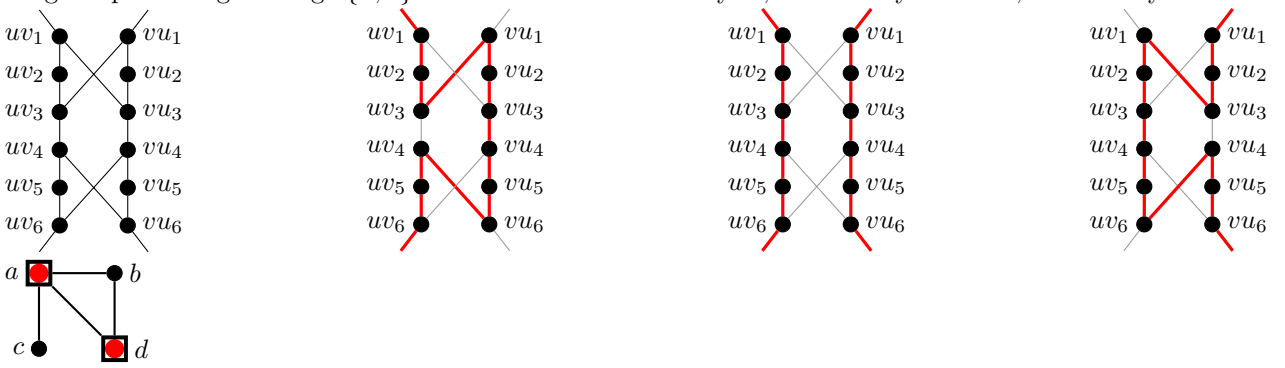
A *Hamiltonian Cycle* in a graph $G = (V, E)$ is a cycle visiting each vertex exactly once. (Alternatively, a permutation of V such that every two consecutive vertices are adjacent and the first and last vertex in the permutation are adjacent.)

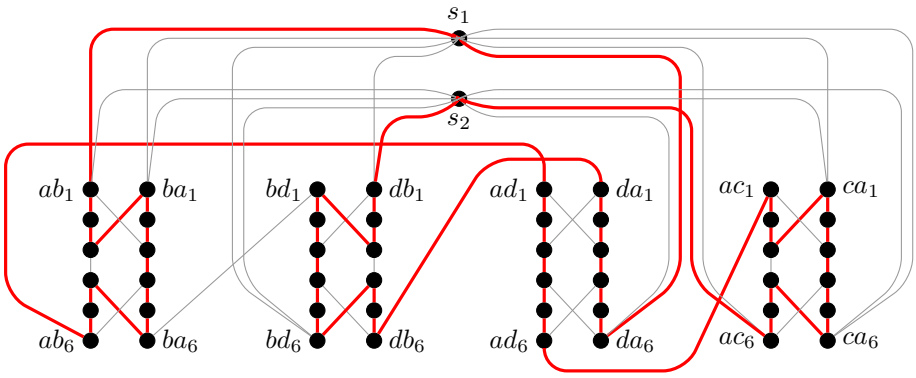
Hamiltonian Cycle	
Input:	Graph G
Question:	Does G have a Hamiltonian Cycle?

Theorem 20. Hamiltonian Cycle *is* NP*-complete.*

Proof sketch.
- Hamiltonian Cycle is in NP: the certificate is a Hamiltonian Cycle of G.
- Let us show: Vertex Cover $\leq_P$ Hamiltonian Cycle
- Let $(G = (V, E), k)$ be an instance for Vertex Cover (VC).
- We will construct an equivalent instance G' for Hamiltonian Cycle (HC).
- Intuition: Non-deterministic choices
  - for VC: which vertices to select in the vertex cover
  - for HC: which route the cycle takes
- Add k vertices $s_1, \ldots, s_k$ to G' (*selector vertices*)
- Each edge of G will be represented by a gadget (subgraph) of G'
- s.t. the set of edges covered by a vertex x in G corresponds to a partial cycle going through all gadgets of G' representing these edges.
- Attention: we need to allow for an edge to be covered by both endpoints

Gadget representing the edge $\{u, v\} \in E$ Its states: 'covered by u', 'covered by u and v', 'covered by v'





5 Further Reading

- Chapter 34, **NP-Completeness**, in (Cormen et al., 2009)
- Garey and Johnson's influential reference book (Garey and Johnson, 1979)

References

Stephen A. Cook (1971). "The Complexity of Theorem-Proving Procedures". In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971)*, pp. 151–158.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2009). *Introduction to Algorithms*. 3rd ed. The MIT Press.

Michael R. Garey and David S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.

Richard M. Karp (1972). "Reducibility among combinatorial problems". In: *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*. New York: Plenum, pp. 85–103.

Leonid Levin (1973). "Universal sequential search problems". In: *Problems of Information Transmission* 9.3, pp. 265–266.

Kernelization

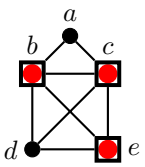
Serge Gaspers

Contents

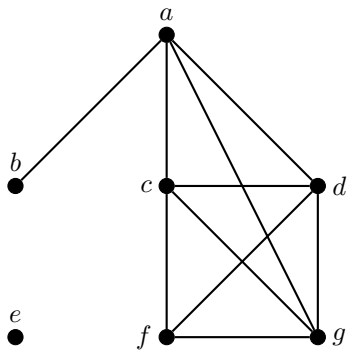
1 Vertex Cover

A *vertex cover* of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that for each edge $\{u, v\} \in E$, we have $u \in S$ or $v \in S$.

Vertex Cover	
Input:	A graph $G = (V, E)$ and an integer k
Parameter:	k
Question:	Does G have a vertex cover of size at most k?



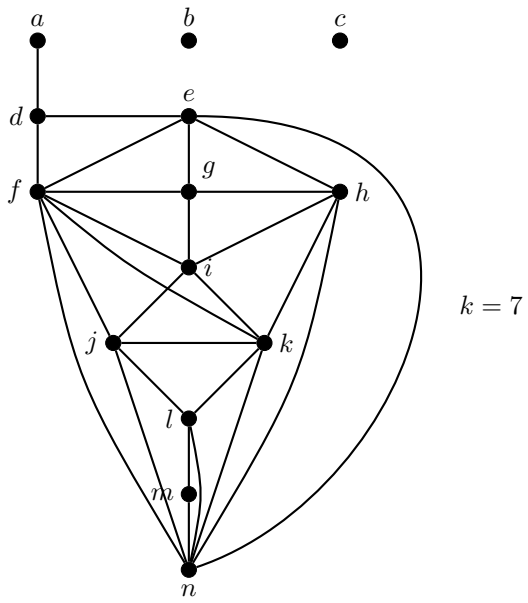
Exercise 1



$k = 4$

Is this a Yes-instance for Vertex Cover? (Is there $S \subseteq V$ with $|S| \leq 4$, such that $\forall\ uv \in E$, $u \in S$ or $v \in S$?)

Exercise 2



1.1 Simplification rules

(Degree-0)
If $\exists v \in V$ such that $d_G(v) = 0$, then set $G \leftarrow G - v$.

Proving correctness. A simplification rule is *sound* if for every instance, it produces an equivalent instance. Two instances I, I' are *equivalent* if they are both YES-instances or they are both NO-instances.

Lemma 1. *(Degree-0) is sound.*

Proof. First, suppose $(G - v, k)$ is a YES-instance. Let S be a vertex cover for $G - v$ of size at most k. Then, S is also a vertex cover for G since no edge of G is incident to v. Thus, (G, k) is a YES-instance.

Now, suppose $(G - v, k)$ is a NO-instance. For the sake of contradiction, assume (G, k) is a YES-instance. Let S be a vertex cover for G of size at most k. But then, $S \setminus \{v\}$ is a vertex cover of size at most k for $G - v$; a contradiction. □

(Degree-1)
If $\exists v \in V$ such that $d_G(v) = 1$, then set $G \leftarrow G - N_G[v]$ and $k \leftarrow k - 1$.

Lemma 2. *(Degree-1) is sound.*

Proof. Let u be the neighbor of v in G. Thus, $N_G[v] = \{u, v\}$.

If S is a vertex cover of G of size at most k, then $S \setminus \{u, v\}$ is a vertex cover of $G - N_G[v]$ of size at most $k - 1$, because $u \in S$ or $v \in S$. If S' is a vertex cover of $G - N_G[v]$ of size at most $k - 1$, then $S' \cup \{u\}$ is a vertex cover of G of size at most k, since all edges that are in G but not in $G - N_G[v]$ are incident to u. □

(Large Degree)
If $\exists v \in V$ such that $d_G(v) > k$, then set $G \leftarrow G - v$ and $k \leftarrow k - 1$.

Lemma 3. *(Large Degree) is sound.*

Proof. Let S be a vertex cover of G of size at most k. If $v \notin S$, then $N_G(v) \subseteq S$, contradicting that $|S| \leq k$. □

(Number of Edges)
If $d_G(v) \leq k$ for each $v \in V$ and $|E| > k^2$ then return NO

Lemma 4. *(Number of Edges) is sound.*

Proof. Assume $d_G(v) \leq k$ for each $v \in V$ and $|E| > k^2$. Suppose $S \subseteq V$, $|S| \leq k$, is a vertex cover of G. We have that S covers at most k^2 edges. However, $|E| \geq k^2 + 1$. Thus, S is not a vertex cover of G. □

1.2 Preprocessing algorithm

VC-preprocess
Input: A graph G and an integer k.
Output: A graph G' and an integer k' such that G has a vertex cover of size at most k if and only if G' has a vertex cover of size at most k'.

$G' \leftarrow G$
$k' \leftarrow k$
repeat
| Execute simplification rules (Degree-0), (Degree-1), (Large Degree), and (Number of Edges) for (G', k')
until *no simplification rule applies*
return (G', k')

Effectiveness of preprocessing algorithms

- How effective is VC-preprocess?
- We would like to study preprocessing algorithms mathematically and quantify their effectiveness.

First try

- Say that a preprocessing algorithm for a problem Π is *nice* if it runs in polynomial time and for each instance for Π, it returns an instance for Π that is strictly smaller.
- $\rightarrow$ executing it a linear number of times reduces the instance to a single bit
- $\rightarrow$ such an algorithm would solve Π in polynomial time
- For NP-hard problems this is not possible unless P = NP
- We need a different measure of effectiveness

Measuring the effectiveness of preprocessing algorithms

- We will measure the effectiveness in terms of the *parameter*
- How large is the resulting instance in terms of the parameter?

Effectiveness of VC-preprocess

Lemma 5. *For any instance (G, k) for* Vertex Cover*, VC-preprocess produces an equivalent instance (G', k') of size $O(k^2)$.*

Proof. Since all simplification rules are sound, $(G = (V, E), k)$ and $(G' = (V', E'), k')$ are equivalent. By (Number of Edges), $|E'| \leq (k')^2 \leq k^2$. By (Degree-0) and (Degree-1), each vertex in V' has degree at least 2 in G'. Since $\sum_{v \in V'} d_{G'}(v) = 2|E'| \leq 2k^2$, this implies that $|V'| \leq k^2$. Thus, $|V'| + |E'| \subseteq O(k^2)$. □

2 Kernelization algorithms

Kernelization: definition

Definition 6. A *kernelization* for a parameterized problem Π is a **polynomial time** algorithm, which, for any instance I of Π with parameter k, produces an **equivalent** instance I' of Π with parameter k' such that $|I'| \leq f(k)$ and $k' \leq f(k)$ for a computable function f. We refer to the function f as the *size* of the kernel.

Note: We do not formally require that $k' \leq k$, but this will be the case for many kernelizations.

VC-preprocess is a quadratic kernelization

Theorem 7. *VC-preprocess is a $O(k^2)$ kernelization for* Vertex Cover.

3 Kernel for Hamiltonian Cycle

A *Hamiltonian cycle* of G is a subgraph of G that is a cycle on $|V(G)|$ vertices.

vc-Hamiltonian Cycle	
Input:	A graph $G = (V, E)$.
Parameter:	$k = vc(G)$, the size of a smallest vertex cover of G.
Question:	Does G have a Hamiltonian cycle?

Thought experiment: Imagine a very large instance where the parameter is tiny. How can you simplify such an instance?
Issue: We do not actually know a vertex cover of size k.We do not even know the value of k (it is not part of the input).

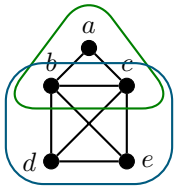
- Obtain a vertex cover using an approximation algorithm. We will use a 2-approximation algorithm, producing a vertex cover of size $\leq 2k$ in polynomial time.
- If C is a vertex cover of size $\leq 2k$, then $I = V \setminus C$ is an independent set of size $\geq |V| - 2k$.
- No two consecutive vertices in the Hamiltonian Cycle can be in I.
- A kernel with $\leq 4k$ vertices can now be obtained with the following simplification rule.

(Too-large)
Compute a vertex cover C of size $\leq 2k$ in polynomial time. If $2|C| < |V|$, then return No

4 Kernel for Edge Clique Cover

Definition 8. An *edge clique cover* of a graph $G = (V, E)$ is a set of cliques in G covering all its edges. In other words, if $\mathcal{C} \subseteq 2^V$ is an edge clique cover then each $S \in \mathcal{C}$ is a clique in G and for each $\{u, v\} \in E$ there exists an $S \in \mathcal{C}$ such that $u, v \in S$.

Example: $\{\{a, b, c\}, \{b, c, d, e\}\}$ is an edge clique cover for this graph.



Edge Clique Cover	
Input:	A graph $G = (V, E)$ and an integer k
Parameter:	k
Question:	Does G have an edge clique cover of size at most k?

The *size* of an edge clique cover $\mathcal{C}$ is the number of cliques contained in $\mathcal{C}$ and is denoted $|\mathcal{C}|$.

Helpful properties

Definition 9. A clique S in a graph G is a *maximal* clique if there is no other clique S' in G with $S \subset S'$.

Lemma 10. *A graph G has an edge clique cover $\mathcal{C}$ of size at most k if and only if G has an edge clique cover $\mathcal{C}'$ of size at most k such that each $S \in \mathcal{C}'$ is a maximal clique.*

Proof sketch. ($\Rightarrow$): Replace each clique $S \in \mathcal{C}$ by a maximal clique S' with $S \subseteq S'$.
($\Leftarrow$): Trivial, since $\mathcal{C}'$ is an edge clique cover of size at most k. □

Simplification rules for Edge Clique Cover
Thought experiment: Imagine a very large instance where the parameter is tiny. How can you simplify such an instance?
The instance could have many degree-0 vertices.

(Isolated)
If there exists a vertex $v \in V$ with $d_G(v) = 0$, then set $G \leftarrow G - v$.

Lemma 11. *(Isolated) is sound.*

Proof sketch. Since no edge is incident to v, a smallest edge clique cover for $G - v$ is a smallest edge clique cover for G, and vice-versa. □

(Isolated-Edge)
If $\exists uv \in E$ such that $d_G(u) = d_G(v) = 1$, then set $G \leftarrow G - \{u, v\}$ and $k \leftarrow k - 1$.

(Twins)
If $\exists u, v \in V$, $u \neq v$, such that $N_G[u] = N_G[v]$, then set $G \leftarrow G - v$.

Lemma 12. *(Twins) is sound.*

Proof. We need to show that G has an edge clique cover of size at most k if and only if $G - v$ has an edge clique cover of size at most k.

($\Rightarrow$): If $\mathcal{C}$ is an edge clique cover of G of size at most k, then $\{S \setminus \{v\} : S \in \mathcal{C}\}$ is an edge clique cover of $G - v$ of size at most k.

($\Leftarrow$): Let $\mathcal{C}'$ be an edge clique cover of $G - v$ of size at most k. Partition $\mathcal{C}'$ into $\mathcal{C}'_u = \{S \in \mathcal{C}' : u \in S\}$ and $\mathcal{C}'_{\neg u} = \mathcal{C}' \setminus \mathcal{C}'_u$. Note that each set in $\mathcal{C}_u = \{S \cup \{v\} : S \in \mathcal{C}'_u\}$ is a clique in G since $N_G[u] = N_G[v]$ and that each edge incident to v is contained in at least one of these cliques. Now, $\mathcal{C}_u \cup \mathcal{C}'_{\neg u}$ is an edge clique cover of G of size at most k. □

(Size-V)
If the previous simplification rules do not apply and $|V| > 2^k$, then return No.

Lemma 13. *(Size-V) is sound.*

Proof. For the sake of contradiction, assume neither (Isolated) nor (Twins) are applicable, $|V| > 2^k$, and G has an edge clique cover $\mathcal{C}$ of size at most k. Since $2^{\mathcal{C}}$ (the set of all subsets of $\mathcal{C}$) has size at most 2^k, and every vertex belongs to at least one clique in $\mathcal{C}$ by (Isolated), we have that there exists two vertices $u, v \in V$ such that $\{S \in \mathcal{C} : u \in S\} = \{S \in \mathcal{C} : v \in S\}$. But then, $N_G[u] = \bigcup_{S \in \mathcal{C}: u \in S} S = \bigcup_{S \in \mathcal{C}: v \in S} S = N_G[v]$, contradicting that (Twin) is not applicable. □

Kernel for Edge Clique Cover

Theorem 14 ((Gramm et al., 2008))**.** Edge Clique Cover *has a kernel with $O(2^k)$ vertices and $O(4^k)$ edges.*

Corollary 15. Edge Clique Cover *is* FPT.

5 Kernels and Fixed-parameter tractability

Theorem 16. *Let* Π *be a decidable parameterized problem.* Π *has a kernelization algorithm* $\Leftrightarrow$ Π *is* FPT.

Proof. ($\Rightarrow$): An FPT algorithm is obtained by first running the kernelization, and then any brute-force algorithm on the resulting instance.

($\Leftarrow$): Let A be an FPT algorithm for Π with running time $O(f(k)n^c)$. If $f(k) < n$, then A has running time $O(n^{c+1})$. In this case, the kernelization algorithm runs A and returns a trivial Yes- or No-instance depending on the answer of A. Otherwise, $f(k) \geq n$. In this case, the kernelization algorithm outputs the input instance. □

6 Further Reading

- Chapter 2, *Kernelization* in (Cygan et al., 2015)
- Chapter 4, *Kernelization* in (Downey and Fellows, 2013)
- Chapter 7, *Data Reduction and Problem Kernels* in (Niedermeier, 2006)
- Chapter 9, *Kernelization and Linear Programming Techniques* in (Flum and Grohe, 2006)
- the kernelization book (Fomin et al., 2019)

References

Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh (2015). *Parameterized Algorithms*. Springer. DOI: 10.1007/978-3-319-21275-3.

Rodney G. Downey and Michael R. Fellows (2013). *Fundamentals of Parameterized Complexity*. Springer. DOI: 10.1007/978-1-4471-5559-1.

Jörg Flum and Martin Grohe (2006). *Parameterized Complexity Theory*. Springer. DOI: 10.1007/3-540-29953-X.

Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi (2019). *Kernelization. Theory of Parameterized Preprocessing*. Cambridge University Press.

Jens Gramm, Jiong Guo, Falk Huffner, and Rolf Niedermeier (2008). "Data reduction and exact algorithms for clique cover". In: *ACM J. Exp. Algorithmics* 13. DOI: 10.1145/1412228.1412236.

Rolf Niedermeier (2006). *Invitation to Fixed Parameter Algorithms*. Oxford University Press. DOI: 10.1093/ACPROF:OSO/9780198566076.001.0001.