Parameterized Algorithms and Complexity

Hans Bodlaender

Joint work with Gunther Cornelissen, Nils Donselaar, Carla Groenland, Hugo Jacob, Lars Jaffke, Johan Kwisthout, Paloma Lima, Isja Mannens, Jesper Nederlof, Jelle Oostveen, Sukanya Pandey, Marcin Pilipczuk, Michal Pilipczuk, Céline Swennenhuis, Krisztina Szilagyi, Erik Jan van Leeuwen, Marieke van der Wegen

Utrecht University

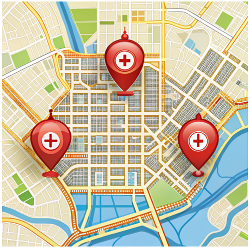
October 10, 2024

This talk

1. An Introduction to Parameterized Algorithms and Parameterized Complexity
   - Parameterized problems
   - FPT, XP, para-NP-complete
   - Kernels
   - The W-hierarchy
2. New developments: the classes XNLP and XALP

Easy problems with small parameters

- Consider facility location problem: place as few as possible fire stations in a city such that each house is < 15 minutes drive from fire station.
- Problem is NP-hard, but ...
- Easy if we have just money for three fire stations: try all possible locations : $O(n^3)$.



Theory of Parameterized Complexity

- Many hard problems become polynomial time solvable when a parameter is small/fixed.
- Early 1990s: Downey and Fellows build theory of parameterized complexity.
- Parameterized problem: subset of $\Sigma^* \times \mathbb{N}$, with Σ a finite alphabet.
  - We call the second argument the parameter: usually denoted by k.
- Compare with 'classic' problem: subset of Σ^*.
- Research questions: *how much time does it cost to solve specific parameterized problems, as function of both the input size (n) and the parameter (k)?*

Parameters

Parameters come in different flavours:

- Target value: is there a set of size at most/at least/exactly k?
- Part of input: given k machines and n jobs, can we schedule ...?
- Structural parameter of input:
  - Graph parameters, like treewidth, pathwidth (and many others).
  - In this talk, we mention treewidth and pathwidth without definition (not really needed to understand arguments today).

Para-NP-completeness

Graph Colouring

Given: Undirected graph $G = (V, E)$, integer k
Question: Is there a proper colouring of G with k colours, i.e, a function $f : V \rightarrow \{1, 2, \ldots, k\}$, such that for all $\{v, w\} \in E$: $f(v) \neq f(w)$?
Parameter: k

Graph Colouring is NP-complete, even when the number of colours $k = 3$. We say:

Graph Colouring is para-NP-complete.

Now, let's look at problems that are polynomial for fixed parameter values...

Different parameterized complexities

FPT (Fixed Parameter Tractable)

There is an algorithm that uses $f(k)n^{O(1)}$ time.

XP (Slice-wise polynomial time)

There is an algorithm that uses $n^{f(k)}$ time.

Example problems

FPT:

- Vertex Cover: $O(2^k(n+m))$ time algorithm.
- Many problems with treewidth as parameter, e.g., Hamiltonian Circuit: $2^{O(tw)}n$ time.
- Integer Linear Programming with p variables can be solved in $O(p^{2.5p+o(p)}L)$ time (with L the number of bits to denote the ILP) (Lenstra, 1983).

XP:

- Dominating Set: $O(n^k)$ time (try all possibilities).
- Bandwidth (def later): $O(n^{k+1})$ time (Gurari, Sudborough, 1984).

Different complexities

☺: **Fixed parameter tractable (FPT)** Has an algorithm with $O(f(k)n^{O(1)})$ time.

😐: **Slice-wise polynomial time (XP)** Has an algorithm with $O(n^{f(k)})$ time.

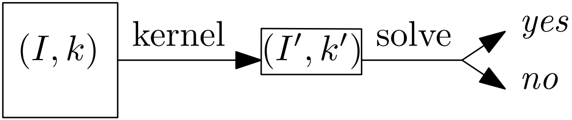
☹: **Para-NP-complete** Problem is NP-complete for fixed value of k.

FPT vs XP

- Complexity of XP is much higher than FPT: $f(k)n^{O(1)}$ vs $n^{f(k)}$.
- Relation with kernelisation(next).
- Downey-Fellows (1990s): Theory to show that problems are unlikely to be in FPT.
  - Central in theory is the W-hierarchy (later); parameterized reductions.
- Proved with diagonalisation: FPT $\subset$ XP.

Kernelisation I

- Before doing a ‘slow’ algorithm, first preprocess the input: build an equivalent, but smaller input.
- Kernelisation: with proof that the resulting equivalent input is *small*: size bounded by function of parameter.



$$Q(I, k) = Q(I', k')$$

Kernelisation II

Kernelisation algorithm

A kernel (or *kernelisation algorithm*) A for a problem Q maps inputs (I, k) of Q to inputs (I', k') such that:

1. A uses polynomial time;
2. $k' \leq g(k)$ and $|I'| \leq g(k)$ for some function g (*the new input has size bounded by a function of the parameter*);
3. $Q(I, k) \Leftrightarrow Q(I', k')$ (*the answer to the problem does not change*).

Some kernelisation theory

Lemma

A decidable problem is in FPT, if and only if it has a kernel.

Proof.

Only $\Leftarrow$ today: build the kernel and then run the decision algorithm. $\square$

Problems with small (polynomial size) kernel, e.g.:

- Vertex Cover: kernel with $\leq 2k$ vertices (Nemhauser, Trotter, 1975, through Linear Programming).
- Maximum Satisfiability: formula in CNF, satisfy at least k clauses: $O(k)$ kernel.

Problems without Polynomial Kernels

Theorem (BDFH+FS/D)

If a parameterized problem is compositional and with parameter in unary NP-hard, then it has no kernel of polynomial size, unless coNP $\subseteq$ NP/poly.

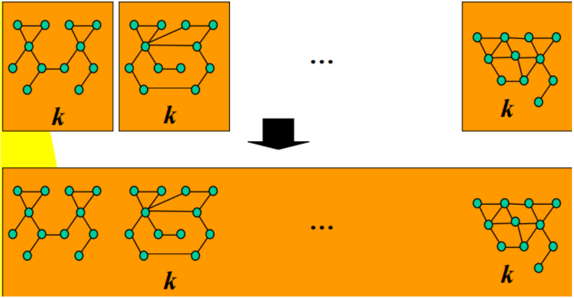


Figure: Composition for Long Path

Distinguishing FPT and XP

- How can we tell that a problem is not in FPT?
- Using complexity classes and reductions.
- Compare to the situation P versus NP — polynomial versus exponential time.
- Downey and Fellows (1990s) introduced:
  - Parameterized reductions.
  - Complexity classes: W[1], W[2], . . . , W[SAT], W[P].

Parameterized reduction

Complete problems are defined in terms of a type of *reductions*.

- A parameterized reduction is a function Φ that maps inputs of parameterized problem A to parameterized problem B:
  - $A(I, k) \Longleftrightarrow B(\Phi(I, k))$;(YES $\Longleftrightarrow$ YES)
  - If $\Phi(I, k) = (I', k')$, then $k' \leq g(k)$ for a computable g (New parameter is also bounded);
  - $\Phi(I, k)$ can be computed in $f(k)n^c$ time.
- Some classes have more restrictions on reductions.

Theorem (Downey,Fellows)

If A has a parameterized reduction to B, and B is in FPT, then A is in FPT.

So, if B is not in FPT, then A is not in FPT. . .

The W-hierarchy

- Classes $W[1]$, $W[2]$, $W[3]$, $\ldots$, $W[SAT]$, $W[P]$ are defined in terms of circuits (definition skipped here); most have equivalent definition with version of Satisfiability (next).
- FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \cdots \subseteq W[SAT] \subseteq W[P]$.
- If $W[1] = FPT$, then the Exponential Time Hypothesis is false — so, we expect that problems that are $W[1]$-hard are not Fixed Parameter Tractable.

W[1]

- A problem belongs to $W[1]$, if and only if it has a parameterized reduction to Weighted 3-Satisfiability.

Weighted 3-Satisfiability

Given: Boolean formula F in Conjunctive Normal Form with three literals per clause, integer k.
Parameter: k.
Question: Can we satisfy F by setting exactly k variables to true and all others to false?

- Also holds also if we replace 3 by any other fixed integer ≥ 2.
- Independent Set and Clique are $W[1]$-complete; many other known $W[1]$-hard and $W[1]$-complete problems.

W[2]

- $W[2]$ has similar characterisation, but clauses can be arbitrary large.
- A problem belongs to $W[2]$, if and only if it has a parameterized reduction to WEIGHTED SATISFIABILITY.

WEIGHTED CNF-SATISFIABILITY

Given: Boolean formula F in Conjunctive Normal Form, integer k
Parameter: k
Question: Can we satisfy F by setting exactly k variables to true and all others to false?

- DOMINATING SET is $W[2]$-complete.

$W[t]$ for $t > 2$

- $W[t]$ is 'roughly' problems of same complexity as deciding if a Boolean formula with t alternations between AND and OR can be satisfied by setting k variables to true.

Weighted t-Normalised Satisfiability

Given: Boolean formula F, integer k, with F of the following form (with t alternations) $\bigvee \bigwedge \bigvee \bigwedge \cdots (\neg) X_i$
Parameter: k
Question: Can we satisfy F by setting exactly k variables to true and all others to false?

$W[SAT]$

- $W[SAT]$: any Boolean formula.
- $W[SAT] \leftrightarrow$ WEIGHTED SATISFIABILITY.

WEIGHTED SATISFIABILITY

Given: Boolean formula F, integer k
Parameter: k
Question: Can we satisfy F by setting exactly k variables to true and all others to false?

$W[P]$

The last class in the W-hierarchy is $W[P] \leftrightarrow$ WEIGHTED CIRCUIT SATISFIABILITY.

WEIGHTED SATISFIABILITY

Given: Boolean circuit C with n input gates and one output gate, integer k
Parameter: k
Question: Can we let C output true by setting exactly k inputs to true and all others to false?

The W-hierarchy: discussion

- Hardness for $W[1]$ implies that it is unlikely that problem is FPT .
- Hardness for classes higher in W-hierarchy implies the same (‘more unlikely’).
- Proving W-hardness: similar to NP-completeness proofs but:
  - parameter must stay bounded;
  - exponential (or more) time in parameter is allowed.
- In W-hierarchy: problems of the form: choose (at least, at most, exactly) k elements from n such that ‘something holds’.

My XNLP-story starts with Bandwidth

- Well studied problem, application for Gaussian elimination. 'Reorganise a matrix such that all non-zero's are in a narrow band around the main diagonal'.

BANDWIDTH

Given: Undirected graph $G = (V, E)$, integer k
Parameter: k
Question: Is there a bijection $f : V \to \{1, 2, \ldots, |V|\}$, such that for each edge $\{v, w\} \in E$: $|(f(v) - f(w)| \leq k$?

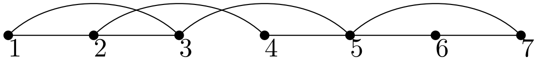


Figure: A layout with bandwidth 2

Some early results on Bandwidth

- 1983, Monien: Bandwidth is NP-complete, for caterpillars with hair length three.
- 1984, Gurari, Sudborough: Bandwidth is in XP: $O(n^{k+1})$ time .
- 1994: Claim by B, Fellows, Hallett that Bandwidth is $W[t]$-hard for all t for trees.
- 1994: Conjecture by Hallett: Bandwidth is not in $W[P]$. Main idea:
  - Problems in $W[P]$ have a certificate with $O(k \log n)$ bits.
  - Bandwidth seems to need $\Omega(n)$ bits for certificate.

More recent results on Bandwidth (parameterized)

- (Dregi, Lokshtanov, 2014): $W[1]$-hard for trees, ETH-based lower bound.
- (B, 2020): BANDWIDTH is $W[t]$-hard for all t for caterpillars .
- "(B, Groenland, Nederlof, Swennenhuis, 2021) BANDWIDTH (for caterpillars) is XNLP-complete.

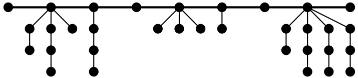
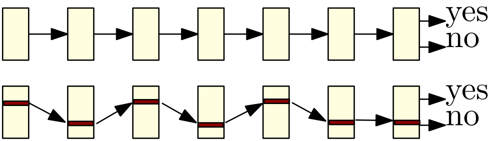


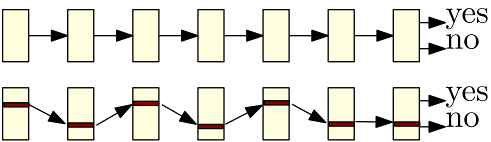
Figure: A caterpillar is a tree with all vertices of degree more than two on one path

From an XP Algorithm to a Non-deterministic Algorithm

- (Gurari, Sudborough, 1984): dynamic programming algorithm. n tables, each of size $O(n^k)$; each table entry has a sequence of k vertices.
- Turn this into a non-deterministic algorithm:
  - Instead of building entire tables, each time non-deterministically guess one element from each table.



A non-deterministic algorithm for BANDWIDTH



Algorithm has k vertices and one counter in $[1, n]$ in memory:

Lemma

BANDWIDTH *can be solved by a non-deterministic Turing Machine in* $O(kn)$ *time with* $O(k \log n)$ *bits additional memory.*

This brings us to a class defined by (Elberfeld et al., 2015).

Birth of XNLP

(Elberfeld, Stockhusen, Tantau, 2015) define parameterized classes with bounded memory and time, including

- $N[f\ poly, \log]$: problems solvable on
  - Non-deterministic Turing Machine;
  - $f(k)n^{O(1)}$ time;
  - $f(k) \log n$ space.
- (EST, 2015): problems complete for $N[f\ poly, \log]$:
  - Non-deterministic Turing machine acceptance with $O(k)$ cells read-write-tape (with polynomial size alphabet) and running time bounded by polynomial in n
  - Timed Non-deterministic Accepting Linear Cellular Automaton
  - Longest Common Subsequence (with variants)

(B, Groenland, Nederlof, Swennenhuis, 2021): renamed $N[f\ poly, \log]$ to XNLP.

Classes with logarithmic space

Classic

- L: deterministic, $O(\log n)$ space
- NL: non-deterministic, $O(\log n)$ space
- L and NL imply polynomial time

Parameterized

- XL: deterministic, $O(f(k) \log n)$ space
- XNL: non-deterministic, $O(f(k) \log n)$ space
- **XNLP: non-deterministic,** $O(f(k) \log n)$ **space and** $O(f(k)poly(n))$ **time**
- $\ldots \subseteq XP$

Relation of XNLP with W-hierarchy

Lemma

For each $t \geq 1$, $W[t] \subseteq XNLP$.

- XNLP-hardness implies $W[1]$-hardness, $W[2]$-hardness, $W[3]$-hardness, ...
- Relation with $W[P]$ and $W[SAT]$ not known; maybe unrelated.

A conjecture

Conjecture ((Michał Pilipczuk and Wrochna, 2018), building upon (Allender et al., 2014))

LONGEST COMMON SUBSEQUENCE *has no algorithm that uses* $n^{f(k)}$ *time and* $g(k)n^{O(1)}$ *space ('XP time and FPT space').*

Equivalent to:

The Slice-wise Polynomial Space Conjecture

If Q is an XNLP-hard problem, then there is no algorithm that solves Q in $n^{f(k)}$ time, and $f(k)n^{O(1)}$ space.

If SPSC holds: XP algorithms for XNLP-hard problems use 'much' space. Indeed, all known algorithms for these use dynamic programming with tables of size $n^{f(k)}$.

Many new XNLP-complete problems

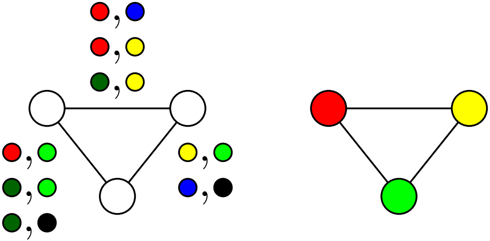
Many new XNLP-complete problems have been found (2021 – now), with results building upon each other, including:

- 'Chained versions': CHAINED INDEPENDENT SET; CHAINED WEIGHTED SATISFIABILITY(BGNS, 2021) — useful for starting reductions
- Many problems with pathwidth as parameter (several papers)
- Problems with other linear width parameters, e.g. linear cliquewidth
- Reconfiguration problems (BGNS, 2021)
- Scheduling problems (BGNS 2021); (Mallem 2024)
- Problems from graph drawing (Blazej et al., 2024)
- Linear graph structure problems, e.g., BANDWIDTH

Binary CSP

Binary CSP

Given: Graph $G = (V, E)$, for each vertex v a set of colours $C(v)$, and for each edge (v, w), a set of pairs of allowed colours $C(v, w) \subseteq C(v) \times C(v)$
Question: Can we assign each vertex v a colour $f(v) \in C(v)$, such that for each edge (v, w), we have $(f(v), f(w) \in C(v, w)$?



Possible ‘starting’ XNLP-hard problem

Theorem

BINARY CSP *is XNLP-complete on* $k \times n$ *grid graphs, with* k *as parameter.*

The hardness proof can be ‘generic’ (in the style of Cook’s proof of the NP-completeness of SATISFIABILITY, using the Turing machine characterisation of the class.

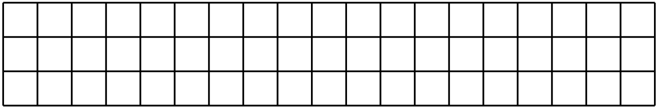


Figure: A $4 \times n$ grid graph

Proof sketch: Membership

Binary CSP for k by n grid graphs is in XNLP:

- For $i = 1$ to n:
  - Guess the colours for the vertices in column i.
  - Have the colours of vertices in columns $i - 1$ (if existing) and i in memory.
  - Check that all adjacent vertices in columns $i - 1$ and i have allowed colour pairs. If not: reject.
- Accept.

We have $2k$ vertex colours and the value of i in memory: $O(k \log n)$ bits.

Proof sketch: Hardness I: the Turing Machine

- Finite alphabet Σ;
- Finite set of states S, with subsets S_A of accepting states and S_R of rejecting states;
- Read-Write Tape of length $f(k) \log n$ + head;
- Input tape of length n + head;
- Collection of transitions: read state, symbol at head on input tape, symbol at head at RW tape — write symbol at head on RW tape, move heads 0 or 1 step left or right, go to new state (non-deterministic).

Proof: Hardness II: Model in the grid

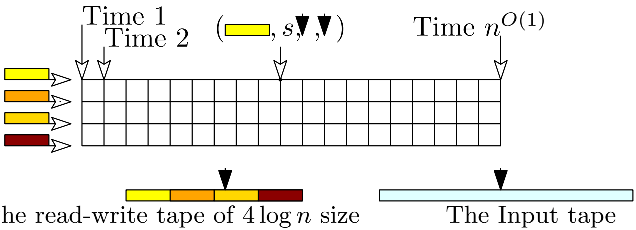
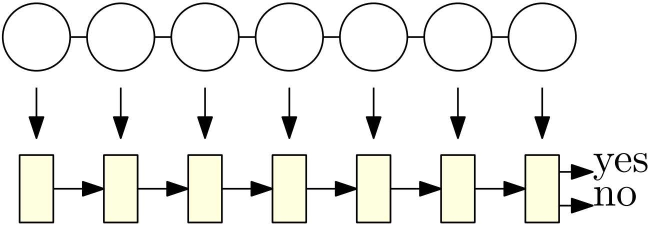


Figure: Partition the RW-tape in $f(k)$ pieces of size $\log n$ each. The colour of the vertex on row i, column t gives the content of the ith piece of RW-tape and state and location of both heads at time t.

First column colours give initial configuration; last column colours must have accepting states. BinCSP can model the proper functioning of TM.

Dynamic Programming on Path Decompositions

- Graphs of small pathwidth have a path decomposition of small width.
- Dynamic programming: compute from left to right a table for each bag.
- Deduce the answer from the last bag.



XNLP-membership Proofs on Path Decompositions

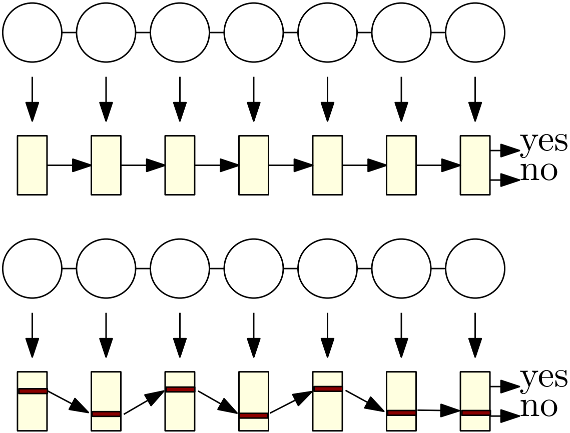


Figure: Turn the DP into XNLP-membership by guessing the element from the next table instead of building it

Example Transformation: List Colouring

List Colouring

Given: Graph $G = (V, E)$, set of colours $\mathcal{C}$, for each vertex $v \in V$, a list of colours $L(v) \subseteq \mathcal{C}$
Question: Is there a colouring $c : V \to \mathcal{C}$, such that for all $v \in V$: $c(v) \in L(v)$, and for all edges $\{v, w\} \in E$: $c(v) \neq c(w)$.

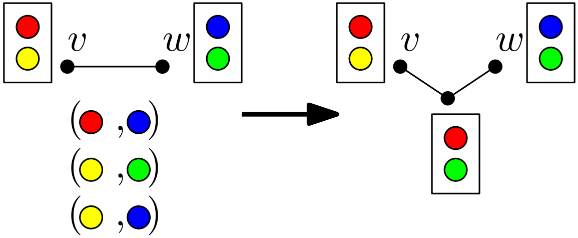
Theorem

List Colouring *with pathwidth as parameter is XNLP-complete.*

Membership with discussed technique (DP by (Jansen, Scheffler, 1997).
Hardness by reduction from Binary CSP for $k \times n$ grids.

Transformation

1. Take input of BINARY CSP for $k \times n$ grids.
2. Change to equivalent instance with each vertex different colour set.
3. For each forbidden pair, add a new vertex with list the forbidden pair.



Transformation Keeps Parameter Small

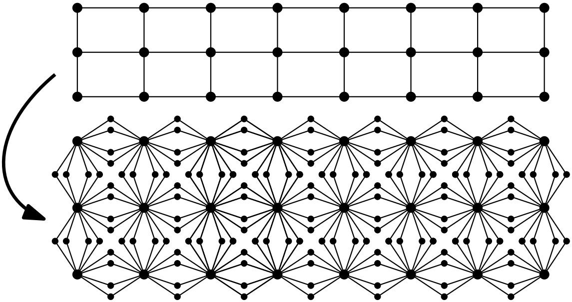


Figure: A $k \times n$ grid graph (pathwidth k) is transformed to a graph with pathwidth $\leq k + 1$

XNLP-hardness proofs for other problems: chains of reductions, each keeping parameter bounded.

Consequences of XNLP-hardness

The Slice-wise Polynomial Space Conjecture

If Q is an XNLP-hard problem, then there is no algorithm that solves Q in $O(n^{f(k)})$ time, and $f(k)n^{O(1)}$ space.

- If SPSC holds, no XP-algorithm for the problem can use FPT space!
- Indeed, the known XP algorithms for XNLP-complete problems use dynamic programming with XP-size tables.
- XNLP-hardness implies $W[t]$-hardness for all $t \in \mathbf{N}$, but with usually much simpler proofs.

From path- to tree-structured graphs

- Several problems on graphs with a linear structure are complete for XNLP.
- When parameterising by treewidth instead of pathwidth, or clique-width instead of linear clique-width, we have XNLP-hardness.
- For what class are these problems complete??

XALP

- Based on (Allender et al. 2014) and (Michał Pilipczuk and Wrochna, 2017) .
- (B, Groenland, Jacob, Pilipczuk, Pilipczuk, 2022) define a class and call it XALP (parameterized variant of class called NLPaux or SAC($O(\log n), n^c$)).
- Where XNLP characterises path-structured dynamic programming, XALP characterises tree-structured dynamic programming.

Definitions of XALP

BGJPP give a number of equivalent definitions of XALP, using Alternating Turing Machines and circuits. An intuitive definition, and easy to work with for membership proofs is:

XALP

Let XALP be the class of parameterized problems accepted by a Non-deterministic Turing Machine that

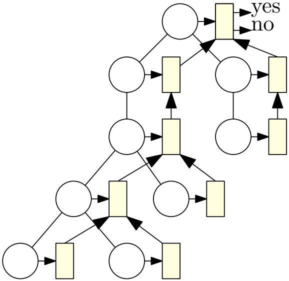
- uses $f(k)n^{O(1)}$ time, for some function f;
- has two types of memory:
  - It has a stack to which it can push symbols, or pop the top symbol;
  - It has a read-write tape of size $f(k) \log n$.

I.e., XNLP plus a stack!

XALP-complete problems I

- About all problems, known to be XNLP-complete with pathwidth as parameter are XALP-complete with treewidth as parameter
- Problems XNLP-complete for linear cliquewidth are usually XALP-complete for cliquewidth, ...
- (B, Szilagyi, 2024): problems on planar graphs with outerplanarity as parameter

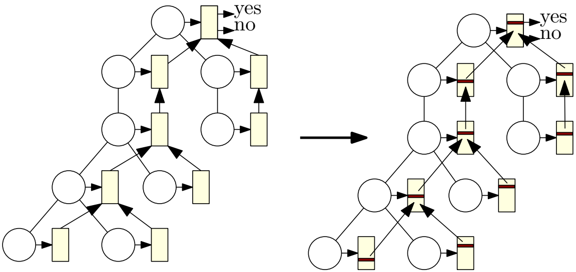
DP on tree decompositions



A dynamic programming algorithm for tree decompositions:

- computes for each bag a table, in post-order (bottom-up);
- deduces the answer from the bag of the root.

DP on tree decompositions and XALP-membership



Turn DP into XALP-membership:
Traverse tree in post-order (bottom-up).

- If bag *i* has $\alpha \in [0, 2]$ children: pop α elements from stack.
- These give 'guessed' table entries of children.
- From these, guess table entry for *i* and push it on stack.

XALP-complete problems II: Structural problems

- Tree Partition Width (introduced as Strong Treewidth by Seese in 1985).
- Domino Treewidth: Is there a tree decomposition of width k (parameter), such that each vertex is in at most two bags.
- Triangulating Coloured Graphs (de Vlas, 2023): Given a graph with a k-colouring of the vertices, is it a subgraph of a properly coloured graph? (Problem with application from phylogeny.)

Other classes

- (Flum, Grohe, 2004): M-hierarchy — small inputs.
- (Abrahamson et al., 1995), (Flum, Grohe, 2001): AW-hierarchy and A-hierarchy: alternations between $\forall$ and $\exists$.
- (Adachi, Iwata, Kasai, 1979, 1984): XP-complete problems (e.g., PEBBLE GAME).
- (B, Groenland, Pilipczuk, 2023): XSLP, captures treedepth.
- (B, Donselaar, Kwisthout, 2022), (Mannens et al., 2024): variants of XNLP and XALP for counting solutions or computing probabilities.

Conclusions

- Rich theory of parameterized complexity; also rich structure of subclasses of XP.
- XNLP ‘captures’ large table sequential dynamic programming.
- XALP ‘captures’ large table tree-structured dynamic programming.
- XNLP-hardness implies (assuming the SPSC) XP-algorithms with ‘much space’.
- Many problems are known/shown to be hard for $W[1]$ or $W[2]$ — interesting to improve this to hardness for larger classes, and aim at completeness.

Additional material

The next slides give additional material:

- More XNLP-hardness proofs: Capacitated Dominating Set and Scheduling with Precedence Constraints
- Slides with overviews
- Discussion on Reconfiguration
- More subclasses of XP

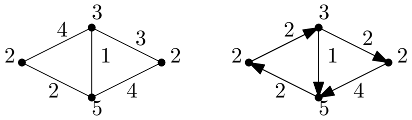
Example Transformation: Target Indegree Orientation to Capacitated Dominating Set

Maximum Target Indegree Orientation (MTIO)

Given: Graph G, weight in unary for each edge $w(e) \in \mathbf{N}$, target in unary for each vertex $t(v) \in \mathbf{N}$.
Question: Can we orient each edge such that each vertex v has total weight of outgoing edges at most $t(v)$?

Theorem (B, Cornelissen, van Wegen, 2022)

MTIO is XNLP-complete with pathwidth as parameter.

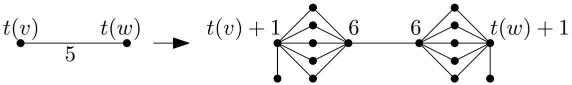


Capacitated Dominating Set

Capacitated Dominating Set

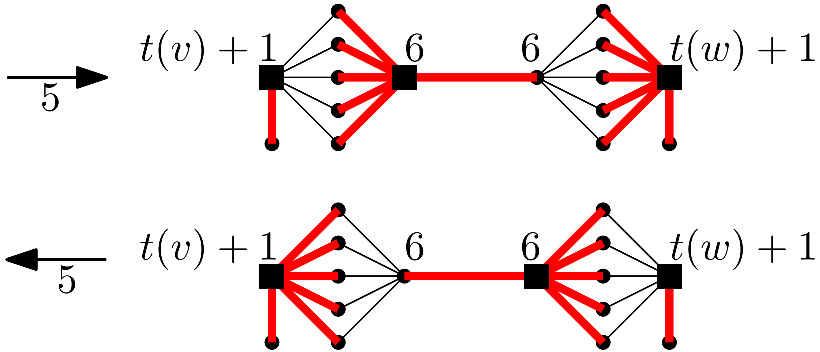
Given: Graph G, capacity $c(v)$ for each v, integer L
Question: is there a set W of size at most L, and a mapping f of vertices in $V \setminus W$ to neighbours in W such that each vertex in W has at most $c(v)$ neighbours mapped to it.

We show that CDS with pathwidth is XNLP-hard: transform each edge as follows:



Pathwidth increases by at most 4.

Equivalence: Max Target Indegree Orientation — Capacitated Dominating Set



The Non-deterministic Non-decreasing Counter Checking Machine

Simple machine model:

- k integer counters, start at 0, never larger than n
- Steps:
  - At each moment, non-deterministically increase a counter
  - Given is a sequence of checks of form $(c_1, \alpha, c_2, \beta)$: if counter c_1 has value α and counter c_2 has value β then halt and reject
  - If all checks successively did not reject, then accept

Accepting NNCCM

Given: Integer k, n, series of checks on k counters.
Parameter: k
Question: does the NNCCM have an accepting run?

The NNCCM theorem

Theorem (BGNS, 2021)

ACCEPTING NNCCM *is XNLP-complete.*

Useful as starting point for reductions (used e.g., for BANDWIDTH and (next:) SCHEDULING WITH PRECEDENCE CONSTRAINTS

Scheduling with Precedence Constraints

$P|prec, p_j = 1|C_{max}$

Given: n tasks, each using unit time, M machines, partial order on tasks (precedence constraints), deadline D.
Question: Can we schedule the tasks (1 task per machine per time step, fulfilling precedences, all before deadline)?

- Well studied problem
- Long standing open problem if problem with three machines is NP-complete or in P (since 1970s)
- (B, Fellows, 1995): problem with parameter M is $W[2]$-hard. The proof also shows that we can take the width of partial order as the second parameter.

XNLP-completeness of Scheduling with Precedence Constraints

Theorem

$P|prec, p_j = 1|C_{max}$ with M and width of partial order as parameters is XNLP-complete.

Membership: take the existing dynamic programming algorithm. Instead of building tables, guess elements.

Sketch of proof 1

From ACCEPTING NNCCM.
Suppose that we have k counters in $[0, n]$, r checks.
Set some 'large enough well-chosen numbers' L, M, D. D is the deadline for all jobs; $D = O(k)$ is number of machines.

We take one *floor gadget*: a chain of D jobs (one at each time step), plus at times $n + 1, 2n + 2, \ldots, L$ additional jobs in parallel.

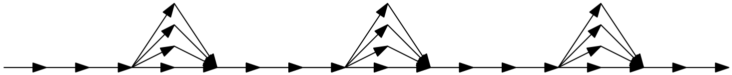


Figure: The floor gadget has length D and forces many jobs at certain time steps (blob)

Sketch of proof 2

For each counter, we have a chain of $D - r$ jobs, with at well chosen points, L additional jobs in parallel (blob).

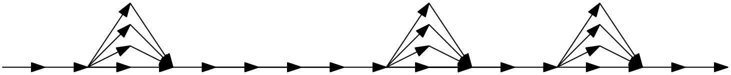


Figure: Quite similar to floor gadgets, but chain has length $D - n$, and 'blobs' at other locations

Intuition

There is space for two blobs at a time step, but not for three. So, when the floor has a blob, at most one counter chain can have a blob.

Sketch of proof 3: Time and counters

Each counter chain can 'skip' a time step at most *n* times (length *D* – *n*, deadline *D*). The *value* of a counter is the number of skips so far.

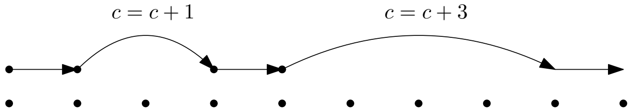


Figure: If we skip α time steps, we increase the counter by α

We choose positions of counter chain blobs such that

NNCCM Accepts ⇔ we never have two chain blobs at the same point as a floor blob.

$\square$

A hierarchy of some classes

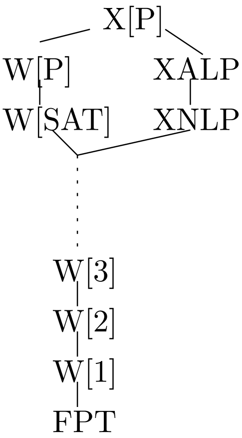


Figure: Relations of some of the classes. In many cases, relations are unknown, and it is not known whether inclusions are proper.

An overview of classes

FPT $f(k)n^c$ time

$W[1]$ Weighted 3-Sat;

$W[2]$ Weighted CNF-Sat;

W-hierarchy 'choose k elements from n'

XNLP Non-deterministic $f(k)n^c$ time, $f(k)\log n$ memory — linear structured Dynamic Programming

XALP XNLP + stack — tree structured dynamic programming

M-hierarchy small input descriptions

A- and AW-hierarchies alternations between $\forall$ and $\exists$ (e.g., short games)

XP $n^{f(k)}$ time; XP-complete game with few pieces

para-NP-complete NP complete for constant parameter

Reconfiguration

- Given: an initial independent set S, a target independent set T, with $|S| = |T|$
- Question: can we change S to T, in a sequences of moves:
  - Each move replaces one vertex from the set by another (*token jumping*)
  - Intermediate sets are still independent

Variant: token sliding: move a vertex to an adjacent vertex
As parameter we take the size of S $(= |T|)$

Independent Set Reconfiguration

INDEPENDENT SET RECONFIGURATION

Given: graph G, two independent sets S and T
Question: Is there a sequence of *moves* that changes S to T? Each move removes a vertex from the set and adds another vertex to the set, while keeping the set to be independent.
Parameter: $|S| = |T|$

- (Kamiński et al., 2012): PSPACE-complete
- (Ito et al., 2014): With $|S| = |T|$ as parameter: $W[1]$-hard
- We also look at variants where we specify maximum number of moves

Overview Independent Set Reconfiguration

Nb of moves	Complexity	Proof idea
parameter	$W[1]$-complete	From INDEPENDENT SET
in unary	XNLP-complete	From CHAINED SATISFIABILITY
in binary	XNL-complete	Simulate TM
unlimited	XL-complete	Simulate Symmetric TM

Results from (Mouawad et al., 2017), (B, Groenland, Nederlof, Swennenhuis, 2021), (BGS, 2022)
Results hold for token jumping and for token sliding

Overview Dominating Set Reconfiguration

Nb of moves	Complexity	Proof idea
parameter	$W[2]$-complete	From DOMINATING SET
in unary	XNLP-complete	From CHAINED SATISFIABILITY
in binary	XNL-complete	Simulate TM
unlimited	XL-complete	Simulate Symmetric TM

Results hold for token jumping and for token sliding

The M-hierarchy: small inputs

- Flum and Grohe introduced the M-hierarchy (2004)
- Capture problems with small input size: instances can be expressed with $k \log n$ bits (definition in terms of circuits, omitted)
- Intersects with W-hierarchy:

$$FPT \subseteq M[1] \subseteq W[1] \subseteq M[2] \subseteq \cdots$$

An $M[1]$-complete problem

With help of the sparsification lemma, the following problem are shown to be $M[1]$-complete (Downey et al. (2003))

Mini Independent Set

Given: Graph $G = (V, E)$ with description length $O(k \log n)$ bits, integer r
Parameter: k
Question: Does G have an independent set of size at least r?

Alternation: the A- and AW-hierarchies

- (Abrahamson et al. 1995): AW-hierarchy
- (Flum and Grohe, 2001): A-hierarchy
- Both capture alternation between $\forall$ and $\exists$ (definitions not given)
- A-hierarchy 'refines' AW-hierarchy
- AW-hierarchy collapses:

$$FPT \subseteq AW[1] = AW[2] = \cdots = AW[t] = \cdots \subseteq AW[SAT] \subseteq AW[P]$$

An $A[2]$-complete problem

p-Clique-Dominating Set

Given: Graph $G = (V, E)$, integers k, ℓ
Parameter: k, ℓ
Question: Does there exist a set $S \subseteq V$ of k vertices, such that for all cliques Q in G with ℓ vertices, Q has a vertex in S or a vertex with a neighbour in S?

- Flum, Grohe: p-Clique-Dominating Set is $A[2]$-complete.
- Notice the alternation: $\exists\forall$

Complete for the AW-hierarchy: Short games

An example of an $AW[*]$-complete problem:

Geography

2-Player game. Given is a graph $G = (V, E)$, a start vertex s. Player 1 starts at s. Players alternatingly choose a neighbour of the last chosen vertex, but cannot choose a vertex that has been chosen. (A simple path is built.) You lose when you are unable to move.

Theorem (Abrahamson et al.)

Deciding if there is a winning strategy for Player 1 that never uses more than k moves (k parameter) is complete for $AW[1] = AW[2] = \cdots$.

XP-complete problems — games with few pieces

- Results from (Adachi, Iwata, Kasai, 1979, 1984) give problems that is complete for XP, e.g. Pebble Game; see (Downey, Fellows, 1997)
- P=AL: Alternating Turing Machines with $O(\log n)$ space = P

Pebble Game

Given: Graph $G = (V, E)$, set of rules $R \subseteq V \times V \times V$, start set $S \subseteq V$, winning vertex $t \in V$.
Parameter: $k = |S|$
Question: Does player 1 have a winning strategy in the following game. Alternatingly, the players move a pebble, following a rule, with $(x, y, z) \in R$ means we can move a pebble from x to z if there are pebbles on x and y but not on z. You win by moving a pebble to t or if your opponent cannot move.

XSLP

(B, Groenland, Pilipczuk, 2023): what if we parameterise by treedepth

- Introduce class XSLP
- Complicated definition
- Several problems complete for XSLP

Counting and XNLP / XALP

- (B, Donselaar, Kwisthout, 2022): variant of XNLP related to PP (probabilistic) — hardness for INFERENCE on probabilistic networks (also called Bayesian Networks)
- (Mannens et al., 2024): #XNLP, #XALP: counting variants of XNLP and XALP