Arch. Math. Logic (1997) 36:321–337

Archive for
Mathematical
Logic

On the parameterized complexity of short computation and factorization

Liming Cai[1,⋆], **Jianer Chen**[2,⋆⋆], **Rodney G. Downey**[3,⋆⋆⋆], **Michael R. Fellows**[4,⋆⋆⋆⋆]

[1] Department of Mathematics, East Carolina University, Greenville, NC 27858, USA
[2] Department of Computer Science, Texas A & M University, College Station, TX 77843, USA
[3] Department of Mathematics, Victoria University, P.O. Box 600, Wellington, New Zealand
[4] Department of Computer Science, University of Victoria, Victoria, B.C. V8W 3P6, Canada

Received August 1, 1994

Abstract. A completeness theory for parameterized computational complexity has been studied in a series of recent papers, and has been shown to have many applications in diverse problem domains including familiar graph-theoretic problems, VLSI layout, games, computational biology, cryptography, and computational learning [ADF,BDHW,BFH, DEF,DF1-7,FHW,FK]. We here study the parameterized complexity of two kinds of problems: (1) problems concerning parameterized computations of Turing machines, such as determining whether a nondeterministic machine can reach an accept state in k steps (the SHORT TM COMPUTATION PROBLEM), and (2) problems concerning derivations and factorizations, such as determining whether a word x can be derived in a grammar G in k steps, or whether a permutation has a factorization of length k over a given set of generators. We show hardness and completeness for these problems for various levels of the W hierarchy. In particular, we show that SHORT TM COMPUTATION is complete for $W[1]$. This gives a new and useful characterization of the most important of the apparently intractable parameterized complexity classes.

⋆ Research supported by an Engineering Excellence Award from Texas A & M University
⋆⋆ Research supported by NSF grant CCR-9110824
⋆⋆⋆ Research supported by Victoria University IGC and by the United States / New Zealand Cooperative Science Foundation under grant INT 90-20558 and the New Zealand Marsden Fund for Basic Science
⋆⋆⋆⋆ Research supported in part by the National Science and Engineering Research Council of Canada
Mathematics Subject Classification: 68Q05, 68Q15, 68Q25, 68Q50
Correspondence to: R.G. Downey

1 Introduction

The central issues on which the theory of parameterized computational complexity focuses are rooted in the following general observations concerning computational problems.

• Many important natural problems have input that consists of two or more items. For example, the familiar problems BANDWIDTH, MIN CUT LINEAR ARRANGEMENT, VERTEX COVER and DOMINATING SET all take as input a graph G and a positive integer k. So it is natural to consider the relative contributions of the different parts of the input to the complexity of the problem.

• There is a large and growing body of results of the following qualitative sort: there is a constant α such for every fixed "parameter" k, the problem Π is solvable in time $O(n^{\alpha})$, with α independent of k. For example, MIN CUT LINEAR ARRANGEMENT and VERTEX COVER are solvable in time linear in the number of vertices of G for every fixed k. The algorithmic techniques used to prove such *fixed parameter tractability* results are in some cases interesting and distinctive (for example, well-quasiordering).

• The fact that a problem Π taking as input two items x and k is complete for *NP* (or *PSPACE*, or *EXPTIME*, ...) tells us *nothing* about the fixed-parameter tractability or intractability of Π.

• For many natural problems, a small range of parameter values may serve important applications.

• Many natural problems involve a "hidden parameter." Elucidating the contribution of this parameter to the complexity of the problem may illuminate why the problem is "easier to solve in practice" than one might expect from traditional complexity analysis. An example of this phenomena is exhibited by the problem TYPE INFERENCE IN ML [HM,DF6]. Engineering practice may introduce (and fix) a parameter. This can render the complexity of a problem *easier* than one desires (for example, in cryptography [FK]). The familiar practice of "coping with *NP*-completeness" by restricting the input can often be conveniently described by a parameterization (for example, by making the parameter the treewidth of the input for a problem concerning graphs).

• For many familiar parameterized problems the difference between fixed-parameter tractability and apparent intractability is very much akin to the apparent difference we observe between problems which are in P and problems which are *NP*-complete. For example, while we can determine whether a graph has a vertex cover of size k or a cutwidth k layout in linear time for each fixed k, the best known algorithms for determining whether a graph has a dominating set of size k, or a bandwidth k layout, require time $O(n^{k+1})$ and are based essentially on a brute force examination of possible solutions. This is strongly reminiscent of the current situation for many *NP*-complete problems.

In a recent series of papers, a framework has been introduced for exploring, qualitatively, the central phenomena of fixed-parameter tractability for parameterized problems [ADF,CCDF,DF1-6]. We say *qualitative*, noting that this is all

that a complexity theory of this sort (including *NP*-completeness) can do. In the more familiar theory of *NP*-completeness, the central phenomena and the model of computational feasibility, is *polynomial time* computability. No one would seriously argue, however, that a polynomial time complexity of n^{100} represents tractability in any practical sense. A similar statement holds for fixed-parameter tractability.

In traditional complexity theory (such as *NP*-completeness), the input to a computational problem is viewed as a single unstructured object — all that matters is its size. Parameterized computational complexity introduces fundamentally the distinction between "more important" and "less important" parts of problem input, and allows us to investigate how different parts contribute to the overall computational complexity of the problem.

Our main results concern the following two problems about resource-bounded Turing machine computations; informally, these study the power of *k-time* and *k-space* for nondeterministic machines.

SHORT TM COMPUTATION
Input: A nondeterministic Turing machine M, a word x and a positive integer k.
Parameter: k
Question: Is there an computation of M on input x that reaches an accept state in at most k steps?

COMPACT TM COMPUTATION
Input: A nondeterministic Turing machine M, a word x and a positive integer k.
Parameter: k
Question: Is there an accepting computation of M on input x that visits at most k tape squares?

Our main result, Theorem 1, shows that SHORT TM COMPUTATION is complete for $W[1]$, and will be seen to provide a new and useful characterization of this most important of the parameterized complexity classes. ("Most important," because $W[1]$-hardness is presently the minimum available demonstration of likely fixed-parameter intractability.) Concretely, Theorem 1 shows that SHORT TM COMPUTATION is fixed-parameter tractable if and only if the familiar (and apparently resistant) CLIQUE problem is fixed-parameter tractable.

Theorem 2 shows that the k-space analog, COMPACT TM COMPUTATION, is hard for the complexity class $W[P]$.

In the next section we review the basics of parameterized computational complexity. In Sect. 3 we sketch the proofs of the main theorems. In Sect. 4 we address related problems about grammars, Post systems, and square tiling. In Sect. 5 we consider some more distantly related short factorization problems in permutation groups and monoids.

2 Parameterized computational complexity

The formal framework for our study is established as follows.

Definition. A *parameterized problem* is a set $L \subseteq \Sigma^* \times \Sigma^*$ where Σ is a fixed alphabet.

In the interests of readability, and with no effect on the theory, we consider that a parameterized problem L is a subset of $L \subseteq \Sigma^* \times N$. For a parameterized problem L and $k \in N$ we write L_k to denote the associated fixed-parameter problem (k is the parameter) $L_k = \{x | (x,k) \in L\}$.

Definition. We say that a parameterized problem L is (uniformly) *fixed-parameter tractable* if there is a constant α and an algorithm Φ such that Φ decides if $(x,k) \in L$ in time $f(k)|x|^\alpha$ where $f : N \to N$ is an arbitrary function.

Definition. Let A, B be parameterized problems. We say that A is (uniformly many:1) *reducible* to B if there is an algorithm Φ which transforms (x,k) into $(x', g(k))$ in time $f(k)|x|^\alpha$, where $f, g : N \to N$ are arbitrary functions and α is a constant independent of k, so that $(x,k) \in A$ if and only if $(x', g(k)) \in B$.

It is easy to see that if A reduces to B and B is fixed parameter tractable then so too is A. Note that if $P = NP$ then problems such as Minimum Dominating Set are fixed-parameter tractable. Thus a completeness program is reasonable.

The classes of parameterized problems that we define below are intuitively based on the complexity of the circuits required to check a solution, or alternatively the "natural logical depth" of the problem. (See also [CC1-2], [KT] and [PY] for different views of this issue in terms of alternating logarithmically bounded Turing Machines and finite model theory.)

We first define circuits in which some gates have bounded fan-in and some have unrestricted fan-in. It is assumed that fan-out is never restricted.

Definition. A Boolean circuit is of *mixed type* if it consists of circuits having gates of the following kinds.
(1) *Small gates*: *not* gates, *and* gates and *or* gates with bounded fan-in. We will usually assume that the bound on fan-in is 2 for *and* gates and *or* gates, and 1 for *not* gates.
(2) *Large gates*: *And* gates and *Or* gates with unrestricted fan-in.

Definition. The *depth* of a circuit C is defined to be the maximum number of gates (small or large) on an input-output path in C. The *weft* of a circuit C is the maximum number of large gates on an input-output path in C.

Definition. We say that a family of decision circuits F has *bounded depth* if there is a constant h such that every circuit in the family F has depth at most h. We say that F has *bounded weft* if there is constant t such that every circuit in the family F has weft at most t. The *weight* of a boolean vector x is the number of 1's in the vector.

Definition. Let F be a family of decision circuits. We allow that F may have many different circuits with a given number of inputs. To F we associate the parameterized circuit problem $L_F = \{(C, k) : C \text{ accepts an input vector of weight } k\}$.

Definition. A parameterized problem L belongs to $W[t]$ if L reduces to the parameterized circuit problem $L_{F(t,h)}$ for the family $F(t, h)$ of mixed type decision circuits of weft at most t, and depth at most h, for some constant h.

Definition. A parameterized problem L belongs to $W[P]$ is L reduces to the circuit problem L_F, where F is the set of all circuits (no restrictions).

Definition. We designate the class of fixed-parameter tractable problems *FPT*.

The above leads to an interesting hierarchy

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

for which a wide variety of natural problems are now known to be complete or hard for various levels (compendiums can be found in [DF2,4,7]).

It should be emphasized that there is no easy correspondence between membership or hardness in any of these parameterized complexity classes and *NP*- or *PSPACE*-completeness (etc.) for the corresponding "unparameterized" problems. For example, in [ADF] problems concerning k-move games are studied, all of which are *PSPACE*-complete in unparameterized form; some of these turn out to be fixed parameter tractable, and others to be hard for various classes of the W hierarchy. A contrasting example can be found in [DEF] where it is shown that VAPNIK-CHERVONENKIS DIMENSION is complete for $W[1]$; the unparameterized form of this problem is probably of difficulty intermediate between *P* and *NP* (see [PY]). Similarly, TOURNAMENT DOMINATING SET, which can be solved in time $O(n^{\log n})$ and is unlikely to be *NP*-complete [PY], can be shown to be complete (in the natural parameterized form) for $W[2]$ (see [DEF]).

3 k-Resource bounded turing machine computations

Theorem 1. SHORT TM COMPUTATION *is complete for* $W[1]$.

Proof. We give the proof at a high level. The formal details are laborious but straightforward. To show hardness for $W[1]$ we reduce from CLIQUE, which is shown to be hard for $W[1]$ in [DF3]. Let $G = (V, E)$ be a graph for which we wish to determine whether it contains a k-clique. We shown how to construct a nondeterministic Turing machine M that can reach an accept state in $k' = f(k)$ moves if and only if G contains a k-clique. The Turing machine M is designed so that any accepting computation consists of two phases. In the first phase, M nondeterministically writes k symbols representing vertices of G in the first k tape squares. (There are enough symbols so that each vertex of G is represented by a symbol.) The second phase consists of making $\binom{k}{2}$ scans of the k tape

squares, each scan devoted to checking, for a pair of positions i, j, that the vertices represented by the symbols in these positions are adjacent in G. Each such pass can be accomplished by employing $O(|V|)$ states in M dedicated to the ij^{th} scan.

In order to show membership in $W[1]$ it suffices to show how the SHORT COMPUTATION problem for a Turing machine $M = (\Sigma, Q, q_0, \delta, F)$ and positive integer k can be translated into one about whether a circuit C accepts a weight k' input vector, where C has depth bounded by some t (independent of k and the Turing machine M), and has only a single large (output) *and* gate, with all other gates small. We arrange the circuit so that the k' inputs to be chosen to be set to 1 in a weight k' input vector represent the various data: (1) the i^{th} transition of M, for $i = 1, ..., k$, (2) the head position at time i, (3) the state of M at time i, and (4) the symbol in square j at time i for $1 \leq i, j \leq k$. Thus we may take $k' = k^2 + 3k$. In order to force exactly one input to be set equal to 1 among a pool of input variables (for representing one of the above choices), we can add to the circuit, for each such pool of input variables, and for each pair of variables x and y in the pool, a small "not both" circuit representing $(\neg x \vee \neg y)$. It might seem that we must also enforce (e.g. with a large *or* gate) the condition, "at least one variable in each such pool is set true" — but this is actually unnecessary, since in the presence of the "not both" conditions on each pair of input variables in each pool, an accepted weight k' input vector *must have* exactly one variable set true in each of the k' pools. Let n denote the total number of input variables in this construction. We have in any case $n = O(k|\delta| + k^2 + k|Q| + k^2|\Sigma|)$.

The remainder of the circuit encodes various checks on the consistency of the above choices. These consistency checks conjunctively determine whether the choices represent an accepting k-step computation by M, much as in the proof of Cook's theorem. These consistency checks can be implemented so that each involves only a bounded number b of the input variables. For example, we will want to enforce that if five variables are set true indicating particular values of: (1) the tape head position at time $i + 1$, (2) the head position at time i, (3) the state at time i, (4) the scanned symbol at time i and (5) the machine transition executed at time i, then these indicated values are consistent with δ. Thus with $O(n^5)$ small "checking" circuits of bounded depth we can insure this sort of consistency. In general, we will have $O(n^b)$ such "checking" circuits for consistency checks involving b values. All of the small "not both" and "checking" circuits feed into the single large output *and* gate of C. The formal description of all this is straightforward. □

We remark that Theorem 1 depends crucially on there being no bound on alphabet size or on the number of nondeterministic transition possibilities out of a state, in the definition of the problem. If we restrict SHORT TM COMPUTATION to TM's with $|\Sigma|$ bounded by some constant b, then the number of configurations is bounded by $b^k|Q|k$ and the problem becomes fixed parameter tractable. A similar statement holds for TM's having a bound b on the number of nondeterministic transition possibilities for any state.

Theorem 2. COMPACT TM COMPUTATION *is hard for* $W[P]$.

Proof. To show that the problem is hard for $W[P]$ we reduce from the problem MONOTONE WEIGHT k CIRCUIT SATISFIABILITY that has been shown to be complete for $W[P]$ in [DF4]. Let C be a circuit for which we wish to determine whether there is an input vector of weight k accepted by C. We may assume that each logic gate g of C has two inputs. In time polynomial in $|C|$ we can describe a Turing machine M sketched as follows.

M has an alphabet consisting of one letter for each input to C, and the operation of M consists of two phases. In the first phase, M makes k moves nondeterministically, writing down in the first k tape squares k symbols which represent k inputs to C set to 1. In the second phase (and visiting no other tape squares), M checks whether the guess made in the first phase represents a vector accepted by the circuit C.

The key point is that we can structure the transition table of M to accomplish this, with the size of the table polynomial in $|C|$. To do this, we make two states q^l_{up} and q^l_{down} for each connection (or *line*) l of the circuit C. Let g be an *and* gate of C, let l be an output line of g and suppose the input lines to the gate g are l_1 and l_2. We include in the transition table for M transitions from q^l_{up} to $q^{l_1}_{up}$, from $q^{l_1}_{down}$ to $q^{l_2}_{up}$, and from $q^{l_2}_{down}$ to q^l_{down}. The significance of being a state q^l_{down} is that this represents a value of 1 for the line l as computed by C on the input guessed in the first phase. The state q^l_{up} might be viewed as a state of *query* about the value of l for the circuit C on the input guessed in the first phase. Note that the three transitions described above for the *and* gate g thus enforce that q^l_{down} can be reached only if $q^{l_1}_{down}$ *and* $q^{l_2}_{down}$ can be reached. The appropriate transitions for an *or* gate will differ in the obvious way, i.e., we arrange that the state q^l_{down} can be reached if either of $q^{l_1}_{down}$ or $q^{l_2}_{down}$ can be reached.

If l is an input line to the circuit C, then we encode in the state table for M a "check" (involving a scan of the k tape squares) to see if the corresponding input symbol was written during the first phase of computation. The second phase begins in the state $q^{l_{out}}_{up}$ where l_{out} is the output line of C, and the only accept state is $q^{l_{out}}_{down}$. □

4 Grammars, post systems and tiling

In this section we consider the parameterized complexity of the following classical problems about derivations of strings of symbols.

SHORT DERIVATION (FOR UNRESTRICTED GRAMMARS)

Instance: The three pieces of information:

(1) A grammar $G = (N, \Sigma, \Pi, S)$, where N is a finite set of "nonterminal symbols," Σ is a finite set of "terminal symbols," Π is a finite set of "production rules" of the form $(\alpha \rightarrow \beta)$ with $\alpha, \beta \in (N \cup \Sigma)^*$, and $S \in N$ is the "start symbol."

(2) A word $x \in \Sigma^*$.

(3) A positive integer k.
Parameter: k
Question: Is there a G-derivation of x of length k? That is, is there a sequence of words $x_0, ..., x_k$ with $x_i \in (N \cup \Sigma)^*$ for $i = 0, ..., k$, that satisfies the requirements:
(1) $x_0 = S$,
(2) $x_k = x$, and
(3) for each $i = 1, ..., k$ there is a production rule $\pi_i = (\beta \to \gamma) \in \Pi$ such that $x_{i-1} = \alpha\beta\delta$ and $x_i = \alpha\gamma\delta$?

We note in passing that if the above problem is restricted to grammars having production rules in which the left hand side always consists of a single nonterminal symbol (termed a *context-free grammar*), then it is fixed-parameter tractable.

SHORT POST CORRESPONDENCE PROBLEM
Instance: A Post system Π and a positive integer k, where a *Post system* consists of a finite alphabet Σ and two sequences $\alpha = (\alpha_1, ..., \alpha_n)$ and $\beta = (\beta_1, ..., \beta_n)$
Parameter: k
Question: Is there a length k solution for Π? That is, is there a sequence of integers $i_1, ..., i_k$ (not necessarily distinct) such that

$$\alpha_{i_1} \cdots \alpha_{i_k} = \beta_{i_1} \cdots \beta_{i_k} \quad ?$$

Lemma 1. CLIQUE *reduces to* SHORT DERIVATION.

Proof. Let $G = (V, E)$ be a simple graph, and k a positive integer. We describe a grammar $\mathcal{G} = (N, \Sigma, \Gamma, S)$, a word $x \in \Sigma^*$, and a positive integer $k' = f(k)$ such that the start symbol S derives x in k' steps if and only if G has a k-clique.

The terminal symbols of the grammar are

$$\Sigma = \{\#\} \cup \{u : u \in V\} \cup \{l\}$$

The nonterminal symbols are

$$N = N_1 \cup N_2 \cup N_3 \cup N_4$$

where

$$N_1 = \{S, S', M, W, Z\}$$

$$N_2 = \{X_i : 1 \le i \le k\}$$

$$N_3 = \{R[i, j] : 1 \le i \le k,\ 0 \le j \le k - 1\}$$

$$N_4 = \left\{T[i, u, j] : 1 \le i \le k,\ u \in V,\ 1 \le j \le \binom{k}{2}\right\}$$

The start symbol is S.
The target string $x \in \Sigma^*$ is

$$x = \#^{3\binom{k}{2}} l^{k+2}$$

where the exponents indicate symbol repetition.

The set of productions Γ is

$$\Gamma = \Gamma_1 \cup \Gamma_2 \cup \cdots \cup \Gamma_{12}$$

In the formal description of some of these production rule sets (in particular, Γ_6, Γ_7 and Γ_8) we make reference to the following sets of indices. Let σ denote the sequence of symbols in the set $\mathscr{S} = \{\sigma[i,j] : 1 \leq i < j \leq k\}$ in increasing lexicographic order. Thus the length of the sequence σ is $\binom{k}{2}$. Let s_r denote the r^{th} symbol of the sequence σ. For $i = 1, ..., k$ define

$$J_i = \{r : s_r = \sigma[i,j],\ 1 \leq j \leq k\}$$

$$J_i' = \{r : s_r = \sigma[j,i],\ 1 \leq j \leq k\}$$

$$J_i'' = \left\{1, ..., \binom{k}{2}\right\} - (J_i \cup J_i')$$

Thus for $k = 4$ we have $\sigma = \sigma[1,2]\sigma[1,3]\sigma[1,4]\sigma[2,3]\sigma[2,4]\sigma[3,4]$, $J_1 = \{1,2,3\}$, $J_1' = \emptyset$, $J_1'' = \{4,5,6\}$, $J_2 = \{4,5\}$, $J_2' = \{1\}$ and $J_2'' = \{2,3,6\}$.

Intuitive Overview. Let S denote the starting symbol for the grammar. The initial production rule (the only one that can be applied) is

$$S \rightarrow S'R[k,1] \cdots R[1,1]\#X[1]X[2]\#X[1]X[3]\# \cdots \#X[k-1]X[k]Z$$

There are productions by which the pairs of symbols $X[i]X[j]$ may produce a pair of terminal symbols representing an edge of the graph; this represents in some sense a "guess" about a k-clique in G. Note that there are $\binom{k}{2}$ pairs corresponding to the $\binom{k}{2}$ edges of a k-clique. For such a guess to be consistent, it must be verified that each occurrence of the symbol $X[i]$ produces the same guessed vertex (as the endpoint of a guessed edge). The R and T symbols make this consistency check by "commuting across" the intermediate string. The last index of these symbols functions as a counter that keeps track of which edge of the potential clique is being checked. The $R[i,*]$ ("read") symbols commute until the first occurrence of the i^{th} vertex (produced by $X[i]$) of the potential k-clique is encountered. At this point the symbol $R[i,*]$ is commuted past the edge of this first occurrence, but transformed into a T ("test") symbol that records the identity of the i^{th} vertex (as the second component in the indexing of the symbol). As this symbol continues to commute across the portion of the string corresponding to the guessed edges, there are three possibilities that arise with respect to next guessed edge of the commute: (1) the first endpoint of the guessed edge should correspond to the recorded vertex of the symbol, (2) the second endpoint of the guessed edge should correspond to the recorded vertex of the symbol, or (3) neither endpoint should correspond to the recorded vertex of the symbol. The positions in the sequence of $\binom{k}{2}$ guessed edges corresponding to these three possibilities are indexed by J_i, J_i' and J_i'', respectively, and the corresponding sets of production rules are Γ_6, Γ_7, and Γ_8. The "read and test" nonterminal symbols that implement the consistency checking can only be eliminated from

the string by successfully commuting to the Z symbol on the right end of the string. In the final phase of a successful derivation of the target string x the symbol S' must also commute across in order to be eliminated at the right end. As it commutes, it replaces the guessed vertices with the place holding symbol #.

The details of the production rules of the grammar are as follows.

$$
\begin{aligned}
\Gamma_1 &= \{S \to S'R[k,1]\cdots R[1,1]\#X_1X_2\#X_1X_3\# \\
&\qquad \cdots\#X_1X_k\#X_2X_3\#X_2X_4\#\cdots\#X_{k-1}X_kZ\} \\
\Gamma_2 &= \{X_iX_j \to uv : 1 \le i < j \le k,\ uv \in E\} \\
\Gamma_3 &= \{R[1,1]\#uv \to \#uvT[1,u,2] : uv \in E\} \\
\Gamma_4 &= \{R[i,j]\#uv \to \#uvR[i,j+1] : 2 \le i \le k,\ 1 \le j \le i-2,\ uv \in E\} \\
\Gamma_5 &= \{R[i,j]\#uv \to \#uvT[i,v,j+1] : 2 \le i \le k,\ j = i-1,\ uv \in E\} \\
\Gamma_6 &= \{T[i,u,j]\#uv \to \#uvT[i,u,j+1] : 1 \le i \le k,\ j \in J_i,\ uv \in E\} \\
\Gamma_7 &= \{T[i,v,j]\#uv \to \#uvT[i,v,j+1] : 1 \le i \le k,\ j \in J_i',\ uv \in E\} \\
\Gamma_8 &= \{T[i,u,j]\#xy \to \#xyT[i,u,j+1] : 1 \le i \le k,\ j \in J_i''\} \\
\Gamma_9 &= \left\{T\left[i,u,\binom{k}{2}+1\right]Z \to ZM : 1 \le i \le k-1,\ u \in V\right\} \\
\Gamma_{10} &= \left\{T\left[k,u,\binom{k}{2}+1\right]Z \to WM\right\} \\
\Gamma_{11} &= \{S'\#uv \to \#\#\#S' : uv \in E\} \\
\Gamma_{12} &= \{S'WM^k \to l^{k+2}\}
\end{aligned}
$$

It remains only to specify the number of steps for the derivation:

$$k' = 1 + \binom{k}{2} + (k+1)\left[\binom{k}{2}+1\right]$$

Half of the correctness argument for the reduction is straightforward. If G has a k-clique then a derivation of x of length k' can be written down by following the sketch above, noting that all of the necessary means for commuting symbols from left to right are available. For the other half of the argument, it is easy to see that the necessary first step of the derivation (since it is the only one possible) creates a situation where the the R symbols (or the T symbols into which they are transformed) must be commuted across the string. The key point is that this is possible only if the guessed edges are consistent in providing evidence of a k-clique in G. □

Lemma 1 together with the results of [DF3] shows that SHORT DERIVATION is $W[1]$ hard. Recall that a *context-sensitive* grammar is one where the productions $\alpha \to \beta$ satisfy the length restriction $|\alpha| \le |\beta|$ and $\alpha, \beta \ne \epsilon$. Our argument for Lemma 1 actually shows that SHORT DERIVATION remains $W[1]$ hard for this special case. We next establish membership in $W[1]$.

Lemma 2. SHORT DERIVATION *is in* $W[1]$.

Proof. To show that SHORT DERIVATION belongs to $W[1]$ we will use Theorem 1. That is, we will describe (at a high level) how to reduce the problem of determining whether a word x can be generated in k steps from a given (unrestricted) grammar $G = (N, \Sigma, \Pi, S)$, to the problem of determining whether a nondeterministic Turing machine can reach a halting configuration in $f(k)$ steps.

The computation of the Turing machine is organized in two phases. The first phase nondeterministically guesses a description of the k-step derivation of x. (This description can be recorded by $O(k^2)$ symbols written in as many tape squares; we describe how this can be accomplished below.) The second phase consists of a deterministic computation that checks whether the guessed derivation of x is valid.

Suppose the sequence $x_0, ..., x_k$ corresponds to a k-step derivation of x in the grammar G in the sense that: (1) $x_0 = S$ is the start symbol of G, (2) $x_k = x$, and (3) for each i, $i = 1, ..., k$, x_i can be obtained from x_{i-1} by the application of the production rule $\alpha_i \rightarrow \beta_i$. For any word $w \in (N \cup \Sigma)^*$ let $w[j]$ denote the j^{th} symbol of w, and for $s \leq t$ let $w[s, t]$ denote the substring of w consisting of the symbols $w[s] \cdots w[t]$. Thus for the production rule $\alpha_r \rightarrow \beta_r$ we write $\beta_r[s, t]$ to denote the substring of the yield β_r consisting of the s^{th} through t^{th} symbols.

We employ an alphabet with a distinct symbol for each possible substring $\beta_r[s, t]$ of the yield of a production rule. Note that the number of symbols required is bounded by a polynomial in the size of the description of the grammar G. Let Γ denote this alphabet.

The description of the k-step derivation of x consists of, for $i = 1, ..., k$:
(1) A factorization of x_i

$$x_i = \beta_{i_1}[s_{i_1}, t_{i_1}] \cdots \beta_{i_m}[s_{i_m}, t_{i_m}]$$

represented by symbols of Γ.
(2) The production rule $\alpha_i \rightarrow \beta_i$ that yields x_i from x_{i-1}.
(3) The substring of symbols of (1) for x_{i-1} that represents the symbols consumed in the application of the production rule identified in (2).
(4) The substring of symbols of (1) for x_i that represents the symbols yielded by the application of the production rule identified in (2).

We can describe appropriate factorizations for the strings $x_0, ..., x_k$ in more detail as follows. To each symbol of each of the strings $x_0, ..., x_k \in (N \cup \Sigma)^*$ we can associate a *time of production*: the index i of the string x_i in which the symbol first appeared (i.e., was in the yield of the application of the production rule that produced x_i from x_{i-1}), and a *time of consumption*: the index j (if one exists, otherwise say ∞) such that the symbol is consumed by the application of the production rule that yields x_{j+1} from x_j. Say that two symbols in x_i are *equivalent* if: (1) they are adjacent in x_i, and (2) they have the same times of production and consumption. The relevant factorization of x_i is then given by the equivalence classes. Each factor can be expressed by a symbol $\beta_r[s, t] \in \Gamma$.

By an easy induction, the factorization for each x_i has a representation over the symbols of Γ of length at most $1 + 2(k - 1)$.

It follows from the definition of the equivalence relation that if x_i is produced from x_{i-1} by the production rule $\alpha_i \rightarrow \beta_i$, then α_i is represented by a substring of the symbols of Γ that represent the factorization of x_{i-1} and β_i is represented by a substring of the symbols of Γ that represent the factorization of x_i.

The second computational phase for the Turing machine consists of k deterministic checks, with the i^{th} check verifying that the guessed information of the first phase relevant to the derivation of x_i from x_{i-1} is consistent. It is easy to see that each such check can be accomplished by $g(k)$ moves through a state space of polynomial size, for an appropriately chosen $g(k)$. □

As a consequence of Lemmas 1 and 2 and Theorem 1 we have proved:

Theorem 3. SHORT DERIVATION *is complete for* $W[1]$.

We next consider the complexity of SHORT POST CORRESPONDENCE.

Lemma 3. SHORT POST CORRESPONDENCE *reduces to* SHORT TM COMPUTATION.

Proof. Given an instance (Π, k) of the SHORT POST CORRESPONDENCE problem, we can easily express the question in terms of whether a particular Turing machine has a k'-step accepting computation, for k' determined by an appropriate function of k, as follows. Consider a machine M that computes in two phases, over an alphabet that includes one symbol for each pair of strings in the Post system, and one symbol for each of the positive integers $1, ..., m$ where m is a bound on the total number of symbols (in the strings of the Post system) for any solution. (Clearly $m \leq k|\Pi|$.) In the first phase, M writes on $3k$ tape squares indicating (nondeterministically) a *guessed solution* including the information: (1) what Post pair is the i^{th} factor for $i = 1, ..., k$, (2) on what symbol positions the i^{th} factors begin (in the two concatenated strings). In the second phase, M conducts a number of "checks", each consisting of: (1) a scan of the *guess*, recording in the resulting state q, e.g., that the i^{th} factor is (x_j, y_j) and begins in the first component in symbol position r, and that the next factor is due to begin in symbol position s. There is a transition out of q in the transition table for M if and only if the information recorded in the state q is "valid", i.e., $s = r + |x_j|$. The number of states required for this check is $O(|\Pi|^3)$. It is not too hard to see that $2k$ checks of this sort are enough to insure that the guessed starting positions of the factors are consistent. Similarly, it is necessary to make k^2 checks that each guessed (factor + starting position) in the first component is compatible with each (factor + starting position) in the second component. That is, we must insure that this information does not imply any mismatched symbols in the two solution strings for the Post problem. A successful check will make $f(k)$ moves, where f is an appropriately chosen function of k. □

Theorem 4. SHORT POST CORRESPONDENCE *is complete for* $W[1]$.

Proof. Lemma 3 shows that the problem belongs to $W[1]$. To show that it is hard for $W[1]$ we compose the reduction of Lemma 1 (from CLIQUE) with the reduction of Theorem 4.2 of Davis [Da] (attributed to an unpublished manuscript of Floyd). In general, the second reduction of this composition does *not* constitute a parameterized reduction of SHORT DERIVATION to SHORT POST CORRESPONDENCE. For example, the reduction is only defined for context-sensitive grammars. However, the image of the reduction from CLIQUE described by Lemma 1 consists of instances of the SHORT DERIVATION problem where: (1) the grammar is context sensitive, and (2) the word to be derived in k steps has length bounded by a function of k. Under these restrictions, the reductions described by Davis' Theorem 4.2 is indeed a valid parameterized reduction. □

We next consider the complexity of tiling small regions; the problem is defined as follows (see [GJ]).

SQUARE TILING

Instance: Set C of *colors*, collection $T \subseteq C^4$ of *tiles* (where $\langle a, b, c, d\rangle$ denotes a tile whose top, right, bottom and left sides are colored a, b, c, d respectively), and a positive integer k.

Parameter: k

Question: Is there a tiling of a $k \times k$ square using the tiles in T, i.e., an assignment f of a tile $A(i,j) \in T$ to each ordered pair i, j, $1 \leq i \leq k$, $1 \leq j \leq k$, such that (1) if $f(i,j) = \langle a, b, c, d\rangle$ and $f(i+1, j) = \langle a', b', c', d'\rangle$, then $a' = c$, and (2) if $f(i, j+1) = \langle a', b', c', d'\rangle$ then $b = d'$?

Theorem 5. SQUARE TILING *is complete for* $W[1]$.

Proof. Membership in $W[1]$ is straightforward. Given an instance of the tiling problem as defined above we can create a Turing machine M that reaches a halting configuration in k' steps (where k' is an appropriate function of k) as follows. In the first phase of computation, M writes onto k^2 tape squares nondeterministically a choice of tiles. In the second phase of computation M makes $2k(k-1)$ passes, each pass dedicated to checking the compatibility of the tiles chosen for two adjacent positions in the $k \times k$ square. In any given pass, the first recorded tile of the pair is "remembered" as state information.

To show that the problem is hard for $W[1]$ we reduce from the $W[1]$-complete problem CLIQUE in two steps. The most important step is the reduction of CLIQUE to the problem of tiling a $k_1 \times k_2$ rectangular region. The second and easier step is to reduce the rectangular tiling problem to SQUARE TILING. We address the easier step first. Suppose $k_1 \leq k_2$.

Let T denote the set of tiles over the set of colors C. We describe how to construct a set of tiles T' over a set of colors C' such that a $k_1 \times k_2$ region can be tiled from T if and only if a $k' = k_2$ square region can be tiled from T'.

Suppose $z \notin C$. We may take $C' = C_1 \cup C_2$ where

$$C_1 = C \times \{0, ..., k_2\}$$

$$C_2 = \{z\} \times \{0, ..., k_2\}$$

The set of tiles T' is the union $T' = T_1 \cup T_2 \cup T_3$ where

$$\begin{aligned} T_1 &= \{\langle (a,i),(b,j+1),(c,i+1),(d,j)\rangle : 0 \le i \le k_1 - 1, 0 \le j \\ &\quad \le k_2 - 1, \langle a,b,c,d\rangle \in T\} \\ T_2 &= \{\langle (a,k_1),(z,j+1),(z,k_1+1),(z,j)\rangle : 0 \le j \le k_2 - 1, a \in C\} \\ T_3 &= \{\langle (z,i),(z,j+1),(z,i+1),(z,j)\rangle : k_1 + 1 \le i \le k_2 - 1, 0 \le j \le k_2 - 1\} \end{aligned}$$

The basic idea is that the tiles of T_2 and T_3 provide for additional rows of padding to fill out the $k_2 \times k_2$ square. The verification that a $k_1 \times k_2$ rectangular tiling with tiles from T yields a $k_2 \times k_2$ square tiling with tiles from T' is straightforward and left to the reader.

Now suppose there is a $k_2 \times k_2$ square tiling f with tiles from T'. The second components of the C' colors of the tiles in T' forces the tile $A(i,j) = f(i,j)$ to have the form $\langle (x_1, i-1),(x_2,j),(x_3,i),(x_4,j-1)\rangle$, for $1 \le i,j \le k_2$. By the definition of T_1, and forgetting the second components of the colors, f must also describe a tiling of the $k_1 \times k_2$ rectangle in the first k_1 rows with tiles from T.

We next argue that CLIQUE can be reduced to the rectangular tiling problem. Let $G = (V,E)$ and k be an instance of CLIQUE. We will describe a set of colors C and a set of tiles T that can tile a $k_1 \times k_2$ rectangle if and only if G has a k-clique, where $k_1 = k$ and $k_2 = \binom{k}{2}$. We will consider the columns of the $k_1 \times k_2$ rectangle as indexed by J_k in lexicographic order. Let J_k denote the set of $\binom{k}{2}$ ordered pairs (r,s) with $1 \le r < s \le k$. Let J_k^+ denote J_k augmented with the single additional element 0. Consider J_k^+ to be linearly ordered by the lexicographic ordering inherited from J_k together with taking 0 to be the minimal element. For $\alpha \in J_k^+$ let $pre(\alpha)$ denote the immediate predecessor of α in the ordering of J_k^+.

The set of colors is $C = C_1 \cup C_2$ where

$$C_1 = J_k^+ \times V$$

$$C_2 = \{0, ..., k\} \times E$$

The set of tiles is

$$\begin{aligned} T = \{&\langle (i-1, uv),(pre(\alpha), w),(i, uv),(\alpha, w)\rangle : 1 \le i \le k, uv \in E, w \in V, \\ &\alpha = (r,s) \in J_k, \\ &\text{with } (w = u \text{ if } i = r) \text{ and } (w = v \text{ if } i = s)\} \end{aligned}$$

In discussing tiling it is useful to have the notion of the sequence of colors in a row or column. Given a valid tiling f of the $k_1 \times k_2$ rectangle, write

$$f(i,j) = \langle f_1(i,j), f_2(i,j), f_3(i,j), f_4(i,j)\rangle$$

That is, f_1 describes the top color assignment, f_2 describes the right side color assignment, etc. Define the *sequence of colors of the i^{th} row* to be the sequence of length $k_1 + 1$:

$$f_4(i,1), f_2(i,1) = f_4(i,2), f_2(i,2) = f_4(i,3), ..., f_2(i,k_1 - 1) = f_4(i,k_1), f_2(i,k_1)$$

Similarly define the sequence of colors of a column.

Claim 1. If there is a k-clique in G then the $k_1 \times k_2$ rectangle can be tiled from T.

Proof. Let $v_1, ..., v_k$ be the vertices of the clique. We describe a tiling by describing the row and column color sequences. In the i^{th} row the sequence of colors is:

$$(0, v_i), ((1,2), v_i), ((1,3), v_i), ..., ((1,k), v_i), ((2,3), v_i), ..., ((k-1,k), v_i)$$

In the column indexed by $\alpha = (r, s)$ the sequence of colors is:

$$(0, v_r v_s), (1, v_r v_s), (2, v_r v_s), ..., (k, v_r v_s)$$

From this point, it remains only to check that these sequences can be realized by tiles of T, which we leave to the reader.

Claim 2. If the $k_1 \times k_2$ rectangle can be tiled from T then G has a k-clique.

Proof. Let f denote the tiling. By forgetting the first components, we may refer to the colors of C_1 as *representing* a vertex, and we may similarly refer to the colors of C_2 as representing an edge. First note that in any valid tiling from T we must have the same vertex represented in all of the colors of a row sequence, simply because every tile has the same vertex represented on its sides. Similarly, the same edge is represented in the sequence of colors of any column. Furthermore, the first components of the sequence of colors in a column must be increasing and is thus forced to be: $0, 1, 2, ..., k$. A similar statement holds for the sequence of colors in a row. Write v_i to denote the vertex represented in row i. Let $i < j$ and consider the column indexed by (i, j). It is straightforward to check that the definition of T forces the edge represented by this column to be $v_i v_j$, which implies that the v_i are distinct and pairwise adjacent in G. □

5 Short algebraic factorizations

The following problem clearly belongs to $W[P]$.

PERMUTATION GROUP FACTORIZATION

Instance: A set A of permutations $A \subseteq S_n$, and $x \in S_n$.

Parameter: A positive integer k.

Question: Does x have a factorization of length k over A?

Theorem 6. PERMUTATION GROUP FACTORIZATION *is hard for* $W[1]$.

Proof. We reduce from the parameterized problem PERFECT CODE that is shown to be hard for $W[1]$ in [DF3]. Let $G = (V, E)$ be a graph of order n for which we wish to determine whether there is a perfect code of size k, that is, a set of vertices $V' \subseteq V$ of cardinality k with the property that for every vertex $u \in V$, $|V' \cap N[u]| = 1$.

Let $n' = (k+1)n$. We describe how to produce an equivalent instance of PERMUTATION GROUP FACTORIZATION problem for $S_{n'}$.

View n' as divided into n blocks of size $k+1$, with these blocks in 1:1 correspondence with the vertices of G. Let γ denote a cyclic permutation of the elements of a block. Our set A consists of n permutations, one for each vertex of G. For a vertex u and with $N[u]$ denoting the solid neighborhood of u in G, let a_u denote the element of $S_{n'}$ which acts on the blocks corresponding to $v \in N[u]$ according to γ, and which is the identity map on all other blocks.

The permutation x to be factored consists of the permutation γ on each of the n blocks.

The correctness of the reduction is easily seen.

We next consider the related problem of finding short factorizations in the monoid H_n of self-maps on a set of n elements.

MONOID FACTORIZATION
Instance: A set A of self-maps on $[n]$, and a self-map h.
Parameter: A positive integer k.
Question: Is there a factorization of h of length k over A?

Theorem 7. MONOID FACTORIZATION *is hard for* $W[2]$.

Proof. The reduction is from DOMINATING SET. Let $G = (V, E)$ be a graph for which we are to determine whether there is a k-element dominating set. Let n be the order of G. As in Theorem 6, we construct a set A of self-maps on $[n']$ where we view $[n']$ as consisting of n blocks. Here we have $n' = 2n$ (the blocks have size 2). Let α denote the self-map on $\{1, 2\}$ that maps both elements to 2. For each vertex u of G we construct a map a_u that consists of α in each block corresponding to a vertex $v \in N[u]$, and that is the identity map on all other blocks.

The self-map h to be factored consists of α in each block.

Verification that this construction works correctly is straightforward, noting that $\alpha = \alpha^i$ for any number of compositions of the block map α. □

Since the above arguments do not employ any global aspects of the symmetric group or the monoid of self-maps (i.e., everything interesting occurs in the blocks separately) it seems reasonable to ask whether a more intricate construction could be used to improve these results. For example, it might be that these factorization problems are hard for $W[t]$ for any fixed t. Alternatively, it would be interesting if PERMUTATION GROUP FACTORIZATION turned out to belong to $W[1]$ or $W[2]$. This is a nice example of a widespread situation in our present knowledge of the parameterized complexity of concrete problems, namely, large gaps relative to the W hierarchy between the best known membership and hardness results.

References

[ADF] Abrahamson K., Downey R., Fellows M.: Fixed-parameter intractability II. Proc. 10th Symposium on Theoretical Aspects of Computer Science (STACS) (1993). Lecture Notes in Computer Science, vol. 665, pp. 374–385. Berlin Heidelberg New York: Springer 1993

[BDHW] Bodlaender H., Downey R., Hallett M., Wareham H.: Parameterized complexity analysis in computational biology. Comput. Appl. Biosci. **11**, 49–57 (1995)

[BFH] Bodlaender H., Fellows M., Hallett M.: Beyond NP-completeness for problems of bounded width. Proceedings of the ACM Symposium on the Theory of Computing, pp. 449–458 (1994)

[CC1] Cai L., Chen J.: On the amount of nondeterminism and the power of verifying. Proc. International Conference on the Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Computer Science, vol. 711, pp. 311–320. Berlin Heidelberg New York: Springer 1993

[CC2] Cai L., Chen J.: Fixed-parameter tractability and approximability of *NP*-hard optimization problems. Proc. Israeli Conf. on Theory of Computing and Systems (1993), 118–126

[CCDF] Cai L., Chen J., Downey R.G., Fellows M.R.: Advice classes of parameterized tractability. Ann. Pure Appl. Logic (to appear)

[Da] Davis M.: Unsolvable Problems. In: Barwise J. (ed.) Handbook of Mathematical Logic, p. 580. Amsterdam: Elsevier 1977

[DEF] Downey R.G., Evans P.A., Fellows M.R.: Parameterized learning complexity. Proc. Sixth ACM Workshop on Computational Learning Theory (COLT), pp. 51–57. New York: ACM Press 1993

[DF1] Downey R.G., Fellows M.R.: Fixed parameter tractability and completeness. Congr. Num., **87**, 161–187 (1992)

[DF2] Downey R.G., Fellows M.R.: Fixed parameter tractability and completeness I: Basic Results, SIAM J. Comput. **24**, 873–921 (1995)

[DF3] Downey R.G., Fellows M.R.: Fixed parameter tractability and completeness II. On completeness for $W[1]$. Theoretical Computer Science A **141**, 109–131 (1995)

[DF4] Downey R.G., Fellows M.R.: Fixed parameter intractability (extended Abstract). Proceedings of the Seventh Annual IEEE Conference on Structure in Complexity Theory, pp. 36–49 (1992)

[DF5] Downey R.G., Fellows M.R.: Fixed parameter tractability and completeness III. Some structural aspects of the W-hierarchy. In: Ambos-Spies K., Homer S., Schöning U. (eds.) Complexity Theory, pp. 166–191. Cambridge: Cambridge University Press 1993

[DF6] Downey R.G., Fellows M.R.: Parameterized computational feasibility. In: Clote P., Remmel J. (eds.) Feasible Mathematics II, pp. 219–244. Boston: Birkhäuser 1995

[DF7] Downey R.G., Fellows M.R.: Parameterized Complexity. Monograph in preparation

[FHW] Fellows M.R., Hallett M.T., Wareham H.T.: DNA physical mapping: Three ways difficult. Proceedings of the First European Symposium on Algorithms. Lecture Notes in Computer Science, vol. **726**, pp. 157–168. Berlin Heidelberg New York: Springer 1993

[FK] Fellows M.R., Koblitz N.: Fixed-parameter complexity and cryptography. In: in Proceedings of the Tenth International Conference on Algebraic Algorithms and Error-Correcting Codes (AAECC 10). Lecture Notes in Computer Science, vol. **673**, pp. 121–131. Berlin Heidelberg New York: Springer 1993

[GJ] Garey M.R., Johnson D.S.: Computers and intractability: A guide to the theory of *NP*-completeness. San Francisco: Freeman 1979

[HM] Henglein F., Mairson H.G.: The complexity of type inference for higher-order typed Lambda Calculi. In: Proc. Symp. on Principles of Programming Languages (POPL), pp. 119–130 (1991)

[KT] Kolaitis P.G., Thakur M.N.: Approximation properties of *NP* minimization classes. Proc. 6th Structure in Complexity Theory Conference, pp. 353–366 (1991)

[PY] Papadimitriou C.H., Yannakakis M.: On limited nondeterminism and the complexity of the V-C dimension. Proccedings of the Eighth IEEE Conf. on Structure in Complexity Theory, pp. 12–18. New York: IEEE Press 1993