The Computational Complexity of Factored Graphs

Shreya Gupta
University of California San Diego, La Jolla, CA, USA

Boyang Huang
University of California San Diego, La Jolla, CA, USA

Russell Impagliazzo
University of California San Diego, La Jolla, CA, USA

Stanley Woo
University of California San Diego, La Jolla, CA, USA

Christopher Ye
University of California San Diego, La Jolla, CA, USA

Abstract

While graphs and abstract data structures can be large and complex, practical instances are often regular or highly structured. If the instance has sufficient structure, we might hope to compress the object into a more succinct representation. An efficient algorithm (with respect to the compressed input size) could then lead to more efficient computations than algorithms taking the explicit, uncompressed object as input. This leads to a natural question: when does knowing the input instance has a more succinct representation make computation easier?

We initiate the study of the computational complexity of problems on factored graphs: graphs that are given as a formula of products and unions on smaller graphs. For any graph problem, we define a parameterized version that takes factored graphs as input, parameterized by the number of (smaller) ordinary graphs used to construct the factored graph. In this setting, we characterize the parameterized complexity of several natural graph problems, exhibiting a variety of complexities. We show that a decision version of lexicographically first maximal independent set is **XP**-complete, and therefore *unconditionally* not fixed-parameter tractable (**FPT**). On the other hand, we show that clique counting is **FPT**. Finally, we show that reachability is **XNL**-complete. Moreover, **XNL** is contained in **FPT** if and only if **NL** is contained in some fixed polynomial time.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Complexity classes; Theory of computation → Problems, reductions and completeness

Keywords and phrases Parameterized Complexity, Fine-grained complexity, Fixed-parameter tractability, Graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ITCS.2025.58

Related Version *Full Version*: `https://arxiv.org/abs/2407.19102` [37]

Funding *Russell Impagliazzo*: Supported by NSF Award AF: Medium 2212136.
Christopher Ye: Supported by NSF Award AF: Medium 2212136, NSF grants 1652303, 1909046, 2112533, and HDR TRIPODS Phase II grant 2217058.

Acknowledgements We would like to thank Antonina Kolokolova, Anthony Ostuni, and anonymous reviewers for their many helpful comments and suggestions.

1 Introduction

Traditionally, algorithm design and computational complexity both measure computational time as a function of the input size. Thus, the complexity of computational problems is crucially sensitive to the way the instances are represented as bit sequences. While graphs and abstract data structures can be complex and expressive in the worst case, practical instances

are often highly structured or regular. For example, road networks are often organized into repetitive grid patterns. Similarly, in databases, relations frequently inherit underlying structures from previous relations through operations like joins. In molecular geometry, compounds such as graphite are composed of layers of graphene, with each layer forming a regular honeycomb structure. If the input instance has sufficiently regular structure, we might hope to compress the object into a more succinct representation.

An algorithm that is efficient with respect to the size of the succinct encodings would then be able to compute the desired result more efficiently than an algorithm that takes a naive representation of the input. This raises the question: when does knowing that the input instance was created in a uniform way (or has a succinct representation) make computation easier?

To address this question, various formulations of "succinct representation" have been explored. For example, one such formulation is given by a Boolean circuit that can produce any particular input bit [35, 61]. Frequently, the complexity of problems given in this type of succinct format is exponentially more difficult than when the input is given explicitly (see e.g. [35, 31, 49, 48]). However, it's important to note that the difficulty only increases as a function of the smaller size parameter, and the actual problem has not necessarily become more difficult.

Another type of succinct representation, factored instances, was introduced by [21]. While they give a number of specific problems rather than introduce the concept abstractly, we can generalize by thinking of a fixed set of operations that take pairs of instances to possibly larger instances. For example, one natural operation might be set sum, taking two sets of integers A, B to the set of all sums $A + B = \{a + b \mid a \in A, b \in B\}$. Instead of being described directly, the input to a factored problem is given as this operation applied to a pair of possibly smaller instances. If these operations can be computed in polynomial time, the size of the output might be polynomially larger than the input, so this representation might be considerably smaller than the original. For each such set of operations and problem on instances, we can define a parameterized version of the problem on factored instances, where the input is represented as a formula in these (binary) operations over smaller instances, with the parameter k being the number of smaller instances. If the underlying problem is polynomial time solvable, then for fixed k, the problem remains polynomial time solvable, but with an exponent that potentially grows with k. How the compressed representation affects the difficulty of the problems can be formalized in terms of the complexity of such factored problems from the point of view of parameterized complexity. In this paper, we consider such parameterized factored problems for graphs, using standard graph products and union as our operations.

To make this precise, we first review some standard concepts from parameterized complexity. The gold standard for tractability in parameterized complexity is membership in the class of fixed-parameter tractable (**FPT**) problems [25]. Roughly speaking, a factored problem will be fixed-parameter tractable (i.e. in **FPT**) if on an instance composed using k smaller instances of size at most n, there is an algorithm computing the function in $O(f(k)n^C)$ time for some fixed function f and absolute constant C independent of k. In these settings, we think of n as large and k as small, so that any dependence on k alone is preferable over an exponent of n that grows with k. A natural approach to solving problems on factored graphs is to explicitly compute the graph, requiring time and space $n^{O(k)}$, and solving the problem on the explicit graph. Such an algorithm has an exponent growing with k. Thus such factored problems will always be in **XP**[1], the class of problems that are polynomial for

[1] assuming the problem can be solved in polynomial time on the explicit graph

any particular k, but not necessarily in **FPT**. Even for simple problems with linear time algorithms (on explicit graphs), computing an explicit representation already requires $n^{O(k)}$ time and so this approach does not put the problem in **FPT**.

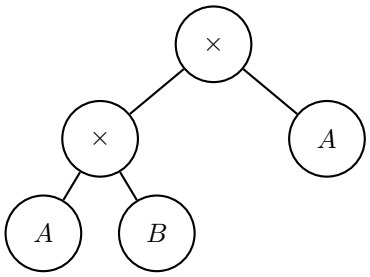
In this work, we initiate the study of the complexity of computational problems on factored graphs. In particular, we consider the question: for a given computational problem, is there an algorithm substantially better than first explicitly computing the factored graph G? This can be formalized as: is the problem in **FPT**, and, if not, how does the best exponent of n possible depend on k?

1.1 Definition of Factored Graphs

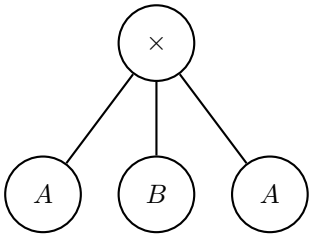
We consider a highly natural class of factored problems where the instances are graphs, and the operations are well-studied graph products and unions. Our factored graphs consist of arbitrary combinations of graphs under the following (binary) operations. Unless otherwise stated, all graphs are directed and (a, b) denotes an edge from a to b.

1. **Cartesian Product.** Given A, B, the Cartesian product $A \square B$ has vertices (a, b) for $a \in A, b \in B$ and there is an edge from (a, b) to (c, d) if either 1) $a = c$ and $(b, d) \in E(B)$, or 2) $(a, c) \in E(A)$ and $b = d$.
2. **Tensor Product.** Given A, B, the tensor product $A \times B$ has vertices (a, b) for $a \in A, b \in B$ and there is an edge from (a, b) to (c, d) if $(a, c) \in E(A)$ and $(b, d) \in E(B)$.
3. **Union.** Given A, B, the union $A \cup B$ has vertices $x \in V(A) \cup V(B)$ and there is an edge from x to y if $(x, y) \in E(A) \cup E(B)$.

A factored graph G can be defined by an ordered tree $T(G)$, where internal nodes are labelled by one of the above three operations $\square, \times, \cup$ and leaves are labelled with (possibly identical) input graphs. Since each operation is associative, we in fact allow the internal nodes to have arbitrary degree.[2] We say that a factored graph $G = f(G_1, \ldots, G_k)$ is *of complexity* (n, k) if the tree has k leaves and each input graph has at most n vertices. Here, we stress that the we parameterize by the number of leaves in $T(G)$, regardless of the fan-in of specific internal nodes or if a certain input graph appears in multiple leaf nodes. As an example, both $(A \times B) \times A$ and $A \times B \times A$ are factored graphs of complexity $(n, 3)$, where $n = \max(|V(A)|, |V(B)|)$. We illustrate their tree structures in Figure 1.



Tree structure of $(A \times B) \times A$.



Tree structure of $A \times B \times A$.

Figure 1 Tree structures of factored graphs $(A \times B) \times A$ and $A \times B \times A$. Both factored graphs have complexity $(n, 3)$ where $n = \max(|V(A)|, |V(B)|)$. Note that in this example the two factored graphs are in fact isomorphic.

[2] We require each internal node to have degree greater than 1, as otherwise the operation is the identity.

As another example, any graph G can be represented by a factored graph of complexity $(2, |E(G)|)$ by taking a union over all edges of the graph.

We define a *factored component* of G, denoted G_F, as follows. G_F is defined by an ordered tree $T(G_F)$, which is obtained from $T(G)$ by deleting every internal node v labeled by $\cup$ and attaching exactly one child of v to its parent. Another way to understand this is that, since the union and product operations obey the distributive law, we can recursively apply this law to convert a factored graph formula into a union of products. Then, each of these product terms is a factored component. The vertices of G_F are k_F-tuples (note $k_F \leq k$, where k_F is the number of leaves of $T(G_F)$ and k is the number of leaves of $T(G)$). We emphasize that the vertices are *flattened* tuples and do not preserve the topology of $T(G_F)$ beyond the order of the leaves. We call k_F the *dimension* of the factored component G_F. The edges of G_F are determined by $T(G_F)$ according to the definition of the graph operations $\square$ and $\times$.

The vertex and edge sets of G are then given by the union of the vertex and edge sets of the factored components. While a vertex may belong to multiple factored components, they must all have the same dimension, allowing us to define the dimension of a vertex (Definition 14). We also observe that G has at most 2^k factored components. We give a more formal definition of factored graphs as well as simple examples in Section 2.

1.2 Our Contributions

Our first result provides an *unconditional* hardness for a natural parameterized version of the well-studied Lexicographically First Maximal Independent Set (LFMIS) problem. The standard decision version of the LFMIS problem takes as input a graph $G = (V(G), E(G))$, where the vertices are indexed as $V(G) = \{0, 1, \ldots, |V(G)|-1\}$, and a special vertex $s \in V(G)$. The problem asks whether s belongs to the LFMIS of G. In the parameterized version, the input is given as a factored graph $G = f(G_1, \ldots, G_k)$ and the vertex indices are provided only for each graph factor $V(G_i) = \{0, 1, \ldots, |V(G_i)| - 1\}$. To recover the indices for $V(G)$, recall that each vertex in the factored graph G is a flattened tuple of numbers $(v_1, \ldots, v_{k'})$ for some $k' \leq k$, and therefore, we define the vertex indices to be given according to the standard lexicographic order of these tuples, with the index 0 given to the vertex with the lowest lexicographic order. In this work, we show that the LFMIS problem on factored graphs is **XP**-complete under **FPT**-reductions[3] and therefore *unconditionally* not in **FPT**, providing a lower bound on the best possible exponent as a function of k. LFMIS is a natural **P**-complete problem [18], studied by [58, 59, 60] including in the parallel setting [20, 50, 12, 8]. We use the **P**-completeness of this problem as an intuition for why the factored version might be difficult (**XP**-complete), but we know of no direct connection between **P**-completeness and hardness of the factored version.

▶ **Theorem 1** (**XP**-completeness of LFMIS). *The LFMIS problem on factored graphs is* **XP***-complete under* **FPT***-reductions and not fixed-parameter tractable. In particular, the LFMIS problem on a factored graph $G = f(G_1, \ldots, G_k)$ of complexity (n, k) requires $n^{\Omega(\sqrt{k})}$ time.*

While many works in parameterized complexity gave *conditional* lower bounds against problems in **FPT** [25], for example completeness for the **W**-hierarchy [25, 26, 28] or the **A**-hierarchy [32], *unconditional* lower bounds against natural problems in **FPT** are relatively rare [27]. Moreover, although it is known that **FPT** $\subsetneq$ **XP** via a diagonalization argument,

[3] See Definition 9.3 [27]. Also formally stated in Definition 20 in the context of factored graph problems.

the literature on **XP**-complete problems remains sparse. The "pebble game" problem was first introduced by Kasai, Adachi, and Iwata [45]. Its parameterized version, the "k-pebble game", was one of the earliest natural combinatorial problems shown to require $\Omega(n^k)$ time and be **XP**-complete [1]. As an application of the k-pebble game, some other game problems, such as the "cat and mouse game" [13] and the "k-peg game", have also been proven to be **XP**-complete via reductions from the k-pebble game problem [1]. More recently, Berkholz established an unconditional lower bound of $O(n^{(k-3)/12})$-time for the existential k-pebble game [6]. Our result provides yet another natural example of an **XP**-complete problem with an explicit lower bound of $n^{\Omega(\sqrt{k})}$.

In contrast to the LFMIS problem on factored graphs, we show that the well-studied problem of clique counting [51] is in **FPT**, solvable in a fixed polynomial of the input graph sizes (where the exponent possibly depends on the size of the clique of interest) times some function on the number of input graphs. Subgraph counting (in particular clique counting) has been studied by a long line of works [51, 16, 11, 42] in a variety of computational models [21, 24, 2, 56, 39]. On graphs with n vertices, s-cliques can be counted in $O(n^s)$ time (and $n^{\Omega(s)}$ time is necessary under standard hardness assumptions). We show that counting s-cliques on factored graphs of complexity (n, k) is in $O(g(s, k)n^s)$ time for some fixed (large) function g. Note that we do not show clique counting is in **FPT** with respect to the clique size parameter s, but only with respect to the complexity of the factored graph k.

▶ **Theorem 2** (Counting Clique Subgraphs is in **FPT**). *Let H be a clique on s vertices. Then, computing the number of exact copies of H in G, denoted $\#\mathbf{H}(G)$, is fixed-parameter tractable. In particular, there is an algorithm computing $\#\mathbf{H}(G)$ on factored graphs G of complexity (n, k) in $O\left(g(s, k)n^s\right)$ time for some function $g(s, k)$.*

Finally, we turn to the problem of reachability, one of the most fundamental computational problems on graphs. On an explicit graph, algorithms such as depth-first search (DFS) and breadth-first search (BFS) compute reachability in linear time. This raises the question: can reachability on factored graphs be computed efficiently? Moreover, reachability is closely related to classic space complexity classes. The general reachability problem is an important **NL**-complete problem [57, 43]; meanwhile, Cook and Mckenzie showed that reachability on a directed acyclic graph of outdegree at most one is **L**-complete [19]. This naturally gives rise to another question: does the completeness of reachability for a classic complexity class extend to its parameterized counterpart when the inputs are given as factored graphs, similar to what we have seen in the case of LFMIS in Theorem 1?

Indeed, we show that the parameterized version of reachability on factored graphs is **XNL**-complete under **FPT**-reductions, where **XNL** is the class of parameterized problems solvable in nondeterministic logarithmic space for any fixed parameter k. On the other hand, while we cannot provide a definitive answer to whether reachability on factored graphs is in **FPT**, we show that answering this question would exactly resolve a major open problem in classical complexity theory. Specifically, we establish a parameterized analog of the well-known open problem that asks whether $\mathbf{NL} \subseteq \mathbf{DTIME}(n^C)$ for some fixed constant C.

▶ **Theorem 3** (**XNL**-completeness of Reachability). *Reachability on factored graphs is* **XNL***-complete under* **FPT***-reductions. Furthermore, the following are equivalent:*

1. *There is a constant C such that $\mathbf{NL} \subseteq \mathbf{DTIME}(n^C)$.*
2. $\mathbf{XNL} \subseteq \mathbf{FPT}$.

This result also has several further implications. First, we observe that if reachability is not in **FPT**, $\mathbf{NSPACE}(h(n)) \not\subseteq \mathbf{P}$ for any space-constructible $h(n) = \omega(\log n)$. On the other hand, if reachability is in **FPT** then the Exponential Time Hypothesis (ETH) is false.

In particular, we claim that since k-SUM (determining if in a set of n numbers of $O(k \log n)$ bits there is a subset of k numbers summing to zero) is in **NL**, then k-SUM can be solved in $O\left(n^C\right) = n^{o(k)}$ time, and therefore ETH is false [55].[4] Finally, by combining the above implications, one can also conclude that if ETH is true, then $\mathbf{NSPACE}(\omega(\log n)) \subsetneq \mathbf{P}$, a result that, to the best of our knowledge, has not been previously established.

Parameterized space complexity classes, including **XL** and **XNL**, have also been studied extensively in the literature [10, 17, 33, 29]. Chen, Flum, and Grohe introduced the first complete problems for **XL** and **XNL** under parameterized-logspace-reductions[5] [17]. In a related direction, [9] identified a wide variety of problems complete for the class **XNLP**, which is the class of parameterized problems solvable in nondeterministic logarithmic space *and* polynomial time for each parameter. Most relevant to our result, Elberfeld, Stockhusen, and Tantau also showed that a parameterized version of reachability with multi-colored edges is complete for the class parameterized-**NL**-cert under **PL**-reductions [29].

1.3 Related Work

A long line of work, initiated by [25, 26, 28, 32], has studied the complexity of parameterized problems as a function of their input size and a parameter. Within parameterized complexity, a common theme is to study the complexity of problems given succinct representations of their input. For example, several previous works have investigated the complexity of computing Nash equilibrium of *succinct* games (represented implicitly) [31, 23, 22, 34, 53, 36]. As another example, [62] considers the complexity of combinatorial problems with succinct representation. Similarly, [52, 63, 40, 41, 49, 48, 14, 15] study the complexity of various computational problems on periodic structures, i.e., travel schedules on a periodic timetable.

Most relevant to this paper, several works have investigated computational problems on graphs with succinct representations such as small circuits [35], distributed graphs described by low complexity agents [4], and factored problems [21]. However, none of these works consider the complexity of factored graphs formed under graph products.

On the subject of succinct representations, researchers have also studied how to represent graphs as efficiently as possible [44, 38, 5, 30, 7, 47, 54].

2 Preliminaries

Unless otherwise noted, we work with directed graphs, with edges from a to b denoted by (a, b). For a graph G, we denote its vertices by $V(G)$ and edges by $E(G)$. For any subset of vertices $S \subset V(G)$, let $G[S]$ denote the subgraph induced by S. For a set X, we define $\mathcal{P}(X) = \{Y : Y \subseteq X\}$ to be the power set of X.

2.1 Graph Products

We begin with the definitions of the relevant graph operations. Let G and H be two directed graphs.

▶ **Definition 4.** *The* Cartesian product $G \square H$ *of* G *and* H *has vertex set* $V(G) \times V(H)$ *and directed edges* $((v_1, u_1), (v_2, u_2))$ *if and only if either*

- $v_1 = v_2$ *and* $(u_1, u_2) \in E(H)$*, or*
- $u_1 = u_2$ *and* $(v_1, v_2) \in E(G)$

[4] We need $O(k^2 \log n)$ bits to verify that k numbers of $O(k \log n)$ bits sum to zero.
[5] Also known as **PL**-reductions, a more restrictive notion compared to **FPT**-reductions

As a simple example, note that the Cartesian product of two paths is a grid.

▶ **Definition 5.** *The* tensor product $G \times H$ *of* G *and* H *has vertex set* $V(G) \times V(H)$ *and directed edges* $((v_1, u_1), (v_2, u_2))$ *if and only if* $(v_1, v_2) \in E(G)$ *and* $(u_1, u_2) \in E(H)$.

▶ **Definition 6.** *The* union $G \cup H$ *of* G *and* H *has vertex set* $V(G) \cup V(H)$ *and edge set* $E(G) \cup E(H)$.

While there are many other graph products and operations to consider, such as the or product, or graph negation, we will primarily focus our study of factored graphs under the above three operations.

We note that the above three operations are associative, and observe that the products of higher arity are given as follows.

▶ **Definition 7.** *The* Cartesian product $\square_{i=1}^{k} G_i$ *has vertex set* $\prod_{i=1}^{k} V(G_i)$ *and directed edges* $((v_1, \ldots, v_k), (u_1, \ldots, u_k))$ *if and only if there is some index* $j \in [k]$ *such that* $(v_j, u_j) \in E(G_j)$ *and* $v_i = u_i$ *for all* $i \neq j$.

▶ **Definition 8.** *The* tensor product $\times_{i=1}^{k} G_i$ *has vertex set* $\prod_{i=1}^{k} V(G_i)$ *and directed edges* $((v_1, \ldots, v_k), (u_1, \ldots, u_k))$ *if and only if* $(v_i, u_i) \in E(G_i)$ *for all* $i \in [k]$.

▶ **Definition 9.** *The* union $\bigcup_{i=1}^{k} G_i$ *of* G *and* H *has vertex set* $\bigcup_{i=1}^{k} V(G_i)$ *and edge set* $\bigcup_{i=1}^{k} E(G_i)$.

2.2 Factored Graph Construction

We now describe how the above operations construct factored graphs. Formally, we define a factored graph by describing how the above graph operations combine the input graphs into a single graph. This motivates the definition of a factored graph according to a tree which specifies the order of operations, or the tree structure of a factored graph.

▶ **Definition 10** (Factored Graph Tree Structure)**.** *The* factored graph tree structure *with* k *leaves, denoted* $f(G_1, \ldots, G_k)$*, is an ordered tree with* k *leaves where internal nodes are labelled by an operation (one of* $\square, \times$*, or* $\cup$*) and the leaves are labelled by graphs* $G_1, \ldots, G_k$*. Note that* G_i *are arbitrary and not necessarily distinct. We require that each internal node has degree at least* 2.

Let $G = f(G_1, \ldots, G_k)$ *denote that* G *is the* factored graph *given by the factored graph tree structure* $f(G_1, \ldots, G_k)$*. For convenience, let* $T(G)$ *denote the factored graph tree structure. We say that the factored graph tree structure* $f(G_1, \ldots, G_k)$*, or simply* G*, is* of complexity (n, k) *if* $T(G)$ *has at most* k *leaves and each leaf is labelled by a graph with at most* n *vertices.*

To illustrate our definitions, we will use the following example of a factored graph $G = ((A \times B) \cup C) \square (D \times (E \cup F))$. Note that there is a natural correspondence between the tree structures of factored graphs and the formulas of the above form, by using parentheses to delineate subtrees of the tree structure. The tree structure of G is given in Figure 2.

We now describe how the vertex and edge sets of a factored graph are obtained from its tree structure. First, we define factored components of factored graphs.

▶ **Definition 11** (Factored Component)**.** *Let* G *be a factored graph. A* factored component G_F *of* G *is the factored graph whose tree structure* $T(G_F)$ *is obtained from the tree structure of* G *by recursively replacing each internal node labelled by* $\cup$ *with the subtree rooted at one of its children. Note that the internal nodes of* $T(G_F)$ *are labelled either* $\square$ *or* $\times$.

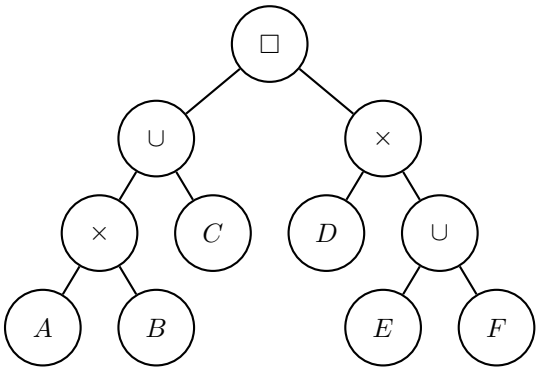
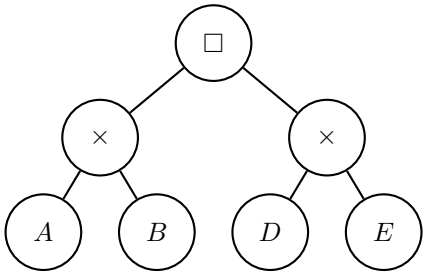


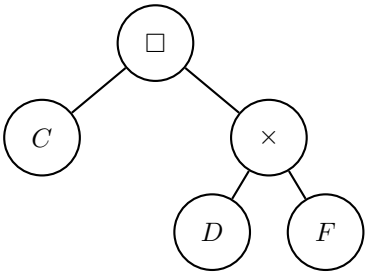
Figure 2 Tree structure of the factored graph $G = ((A \times B) \cup C) \,\square\, (D \times (E \cup F))$.

We say G_F has dimension ℓ *if $T(G_F)$ has ℓ leaves. Suppose the leaves are labelled $G_1, \ldots, G_\ell$. Then, $V(G_F) = \prod_{i=1}^{\ell} V(G_i)$. We say every vertex $v \in V(G_F)$ has dimension ℓ. The edge set $E(G_F)$ is determined by $T(G_F)$ and $E(G_1), \ldots, E(G_\ell)$ following the rules of Cartesian and tensor products.*

In our example, the factored components are $(A \times B) \,\square\, (D \times E), (A \times B) \,\square\, (D \times F), C \,\square\, (D \times E)$, and $C \,\square\, (D \times F)$, with dimensions $4, 4, 3, 3$, respectively. Some example tree structures of factored components are given in Figure 3.



Tree structure of $(A \times B) \,\square\, (D \times E)$.



Tree structure of $C \,\square\, (D \times F)$.

Figure 3 Examples of the Tree Structures of Factored Components.

Here, we emphasize that the vertex sets of the factored components are *flattened* products of the vertex sets of the graphs labeling the leaves. For example, the vertex of the factored component $(A \times B) \,\square\, (D \times E)$ has the form (a, b, d, e) not $((a, b), (d, e))$. In particular, the vertex sets do not depend on the topology of the tree beyond the order of the leaves. Finally, the vertex and edge set of G is simply the union of the factored components.

▶ **Definition 12** (Factored Graph). *Let $G = f(G_1, \ldots, G_k)$ be a factored graph with tree structure $T(G)$ and factored components $G_{F_1}, \ldots, G_{F_m}$. Define $V(G) = \bigcup_{i=1}^{m} V(G_{F_i})$ and $E(G) = \bigcup_{i=1}^{m} E(G_{F_i})$.*

We note here that different factored graphs can have identical vertex and edge sets. For example, $G_1 = (H_1 \times H_2) \times H_3$ and $G_2 = H_1 \times (H_2 \times H_3)$ both have vertex set $\prod_{i=1}^{3} H_i$ and edges from (h_1, h_2, h_3) to (h'_1, h'_2, h'_3) if and only if $(h_i, h'_i) \in E(H_i)$ for all i. This aligns with our expectation, as the tensor product is associative.

Note that a single vertex in G can belong to multiple factored components. In our running example, if there is a vertex $x \in V(E) \cap V(F)$, then the vertex (c, d, x) is in the factored components corresponding to $C \square (D \times E)$ and $C \square (D \times F)$. However, both factored components have dimension 3. This is formalized in the following lemma, which follows directly from the construction of factored graphs.

▶ **Lemma 13.** *Suppose $v \in V(G)$ is in factored components $G_F, G_{F'}$. Then,* $\dim(G_F) = \dim(G_{F'})$.

This allows us to define the dimension of a vertex $\dim(v)$ as the dimension of any factored component it belongs to.

▶ **Definition 14** (Vertex Dimension). *Let $G = f(G_1, \ldots, G_m)$ be a factored graph and $v \in V(G)$ be a vertex. The* dimension *of v, denoted by* $\dim(v)$*, is the dimension* $\dim(G_F)$ *of any factored component G_F that contains v.*

By construction, the endpoints of any edge must belong to the same factored component.

▶ **Lemma 15.** *Let $(u, v) \in E(G)$ be an edge. Then, u and v belong to the same factored component.*

As a corollary, we show that there are no edges between vertices of different dimensions, since vertices in the same factored component necessarily have the same dimension.

▶ **Corollary 16.** *Let (u, v) be an edge in G. Then, u and v have the same dimension.*

Finally, since every vertex has a unique dimension, it is clear that vertex dimension partitions the vertices of the factored graph. Thus, vertex dimension induces an equivalence relation on $V(G)$.

2.3 Parameterized Complexity

In this section, we review the parameterized complexity classes relevant to this paper with respect to problems on factored graphs. We begin with the definition of fixed-parameter tractability (**FPT**).

▶ **Definition 17** (Fixed-parameter Tractability (**FPT**) for Factored Graph Problems). *A problem on factored graphs is* fixed-parameter tractable *if there exists an algorithm that solves the problem on factored graph inputs of complexity (n, k) in time $O\left(g(k)n^C\right)$, where g is a function of k and C is a constant independent of k.*

We also study the parameterized complexity classes **XP** and **XNL**, which correspond to the parameterized version of the classical complexity classes **P** and **NL**, respectively. More generally, one can define the parameterized version **XC** for any classical complexity class **C** as follows [27]: a parameterized language $L \in \mathbf{XC}$ if and only if $L_k \in \mathbf{C}$ for every parameter k. Here, L_k is referred to as the *k-th slice* of L, which is the subset of L consisting of all instances with parameter k. We now give the definitions of **XP** and **XNL** in the context of factored graph problems.

▶ **Definition 18** (**XP** for Factored Graph Problems). *A problem on factored graphs is* in **XP** *if there exists an algorithm that solves the problem on factored graph inputs of complexity (n, k) in time $O\left(f(k)n^{f(k)}\right)$, where f is a function of k.*

▶ **Definition 19** (**XNL** for Factored Graph Problems). *A problem on factored graphs is* in **XNL** *if there exists a nondeterministic algorithm that solves the problem on factored graph inputs of complexity* (n, k) *in space* $O(f(k) \log n)$*, where* f *is a function of* k*.*

Finally, for **XP**- and **XNL**-completeness, we follow the usual definition that a problem is considered to be *complete* for a complexity class **C** if it belongs to **C** and is *hard* for **C** under some suitable notion of reductions. Here, we use the standard notion of parameterized mapping reduction (see Definition 9.3 in [27]), also known as the **FPT**-reduction [17], under which the class **FPT** is closed. We now give a definition for **XP**- and **XNL**-hardness under this reduction in the context of factored graph problems.

▶ **Definition 20** (**XP**-hardness (resp. **XNL**-hardness) for Factored Graph Problems). *A problem* L' *on factored graphs is* **XP**-hard *(resp.* **XNL**-hard*) if for every parameterized language* $L \in \mathbf{XP}$ *(resp.* $L \in \mathbf{XNL}$*), there exists a mapping* F *such that for every parameter* k*,* L_k *mapping reduces to* $L'_{k'}$ *under* F *using* $O(f(k)n^{O(1)})$ *time, where* f *is a function of* k *and* k' *depends only on* k*.*

3 Technical Overview

In this section, we provide a technical overview of our results. Due to space constraints, the proofs are left to the full version of the paper.

3.1 Lexicographically First Maximal Independent Set

Recall that our first main result is the **XP**-completeness of the LFMIS problem on factored graphs.

▶ **Theorem 1** (**XP**-completeness of LFMIS). *The LFMIS problem on factored graphs is* **XP***-complete under* **FPT***-reductions and not fixed-parameter tractable. In particular, the LFMIS problem on a factored graph* $G = f(G_1, \ldots, G_k)$ *of complexity* (n, k) *requires* $n^{\Omega(\sqrt{k})}$ *time.*

Theorem 1 is a consequence of a generic reduction from an arbitrary language $L \in \mathbf{DTIME}(n^\ell)$ to the LFMIS problem on factored graphs. Specifically, given a language $L \in \mathbf{DTIME}(n^\ell)$ and the corresponding Turing machine that decides L in $O(n^\ell)$ time, we construct a factored graph G of complexity $(O(n), O(\ell^2))$ using $O(\ell^2 n^2)$ time, such that solving the LFMIS on G simulates the Turing machine M. Then, it is fairly straightforward from this reduction that the problem is **XP**-complete. Moreover, we also obtain the explicit lower bound of $n^{\Omega(\sqrt{k})}$ by an application of the Time Hierarchy Theorem to the reduction.

Before proceeding to the detailed explanation, we first provide a high level intuition of this construction. Given a Turing machine, we can encapsulate its computation history in a matrix, where each row represents a configuration of the Turing machine at a specific time. The key idea of the reduction is to construct a graph in such a way that selecting vertices for the LFMIS corresponds to recovering the matrix entries, and thereby simulating the machine. To achieve this, we construct a graph with a grid structure mirroring the matrix. At each grid point, we place a collection of vertices representing all the possible choices for the corresponding matrix entry. The edges are defined according to the machine's transition function to ensure that the LFMIS chooses the single correct vertex from each grid point that agrees with the computation history of the Turing Machine, effectively allowing us to simulate the Turing machine by solving the LFMIS on the graph. It turns out that this graph has a highly regular structure and therefore can be factorized into a more succinct representation. In the remainder of this (sub)section, we explain this reduction in further detail.

Let L be a language in $\mathbf{DTIME}(n^\ell)$ and let M be the corresponding Turing machine that decides L in time $O(n^\ell)$. Given an input x to M, if M halts within time T, then the entire computation history of M on x can be represented by a $T \times T$ matrix W, where the i-th row of W represents the configuration of M at time i. To achieve this, each entry $W_{i,j}$ contains the following information about the j-th tape cell at time i: 1) the symbol occupying the cell, 2) whether the tape head is over the cell, and 3) the machine's current state if the tape head is over the cell. The goal now is to define a graph G such that the LFMIS of G recovers the computation history W, thereby simulating the machine M. We modify the Turing machine so that it suffices to query a single vertex in the graph for the halting state of M.

Explicit Graph Definition

We begin by defining G explicitly as an ordinary graph and then give a factored representation of G. Let S be the set of vertices corresponding to all possible choices for an entry in the matrix W. We define the vertex set of G to be T^2 copies of S, arranged in a $T \times T$ grid-like pattern analogous to the matrix W. Each copy of S is referred to as a *supernode*, and we use $S_{i,j}$ to denote the supernode in the i-th row and j-th column. Intuitively, the supernode $S_{i,j}$ represents all possible choices for the entry $W_{i,j}$. Therefore, we must define the edges of G such that the LFMIS includes the single correct vertex, which we call $w_{i,j}$, from each $S_{i,j}$ that agrees with the entry $W_{i,j}$.

The edge set of G consists of two types of edges: *intra*-supernode and *inter*-supernode edges. The intra-supernode edges are designed to ensure that the LFMIS contains *at most one* vertex within each supernode. This is easily achieved by defining a complete digraph on each supernode. On the other hand, the inter-supernode edges are constructed to ensure that the LFMIS contains *the correct* vertex from each supernode that agrees with the corresponding entry of W. This construction is more subtle and leverages the fact that the configuration at any given time of a deterministic machine uniquely determines the next configuration. Moreover, since the tape head can only move one step left or right at a time, each entry $W_{i,j}$ can be uniquely determined given only the three neighboring entries from the previous row: $W_{i-1,j-1}, W_{i-1,j}, W_{i-1,j+1}$ (instead of the entire previous row). To view this from another perspective: the entries $W_{i-1,j-1}$, $W_{i-1,j}$, and $W_{i-1,j+1}$ each restricts the possible choices for $W_{i,j}$ according to the machine's transition function. We define edges between neighboring supernodes to encode these restrictions. Specifically, for $v \in S_{i,j}$ and $v' \in S_{i+1,j'}$, a directed edge (v, v') indicates that, if v is chosen to represent the j-th tape cell at time i, then v' cannot represent the j'-th tape cell at the next time step $i + 1$, based on the machine's transition function. We hope that if the vertices $w_{i-1,j-1} \in S_{i-1,j-1}$, $w_{i-1,j} \in S_{i-1,j}$, and $w_{i-1,j+1} \in S_{i-1,j+1}$ are correctly contained in the LFMIS, then the edges ensure that $w_{i,j}$ is the unique vertex in $S_{i,j}$ that is not adjacent to $w_{i-1,j-1}, w_{i-1,j}$, or $w_{i-1,j+1}$. This will allow us to build the LFMIS inductively. The base case (which corresponds to the first row) is more technical and involves adding horizontal connections between supernodes.

Factored Graph Construction

Naively, this graph has $\Omega(T^2)$ vertices, where $T^2 = \Omega(n^{2\ell})$. However, since the computation rules of M are local (depending only on the symbol of the work tape under the head) and repetitive (the same rules apply regardless of the head's absolute position), we can give a succinct factored representation of G. Specifically, we show that G has a factored graph representation of complexity $(O(n), O(\ell^2))$. The factorization of G is outlined in Figure 4.

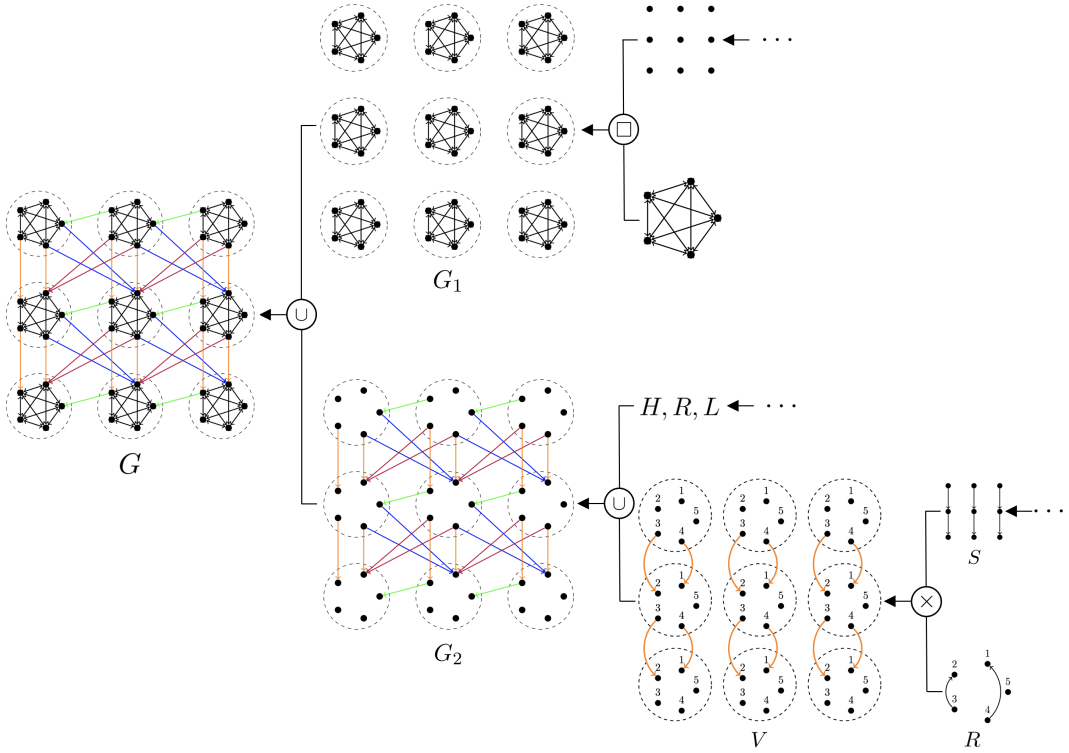


Figure 4 Overview of the Factorization of G. Supernodes are enclosed within dotted circles.

The key idea is to break down the graph G into regular and repetitive substructures. Following this, we begin by decomposing G into two subgraphs, G_1 and G_2, such that both subgraphs share the same vertex set as G, but G_1 contains only the intra-supernode edges and G_2 contains only the inter-supernode edges. G_1 forms a $T \times T$ grid of complete supernodes, which can be represented as the Cartesian product of an empty $T \times T$ grid of vertices and a single complete supernode. Similarly, G_2 is a $T \times T$ grid of supernodes, but with edges connecting neighboring supernodes in four possible directions: vertical (V), horizontal (H), diagonally-right R, and diagonally-left (L). We can further decompose G_2 into four subgraphs V, H, R, and L, which only contain the edges which are in the direction indicated by the name of the subgraph. Each of these subgraphs can be further decomposed into a "structure" graph and a "relation" graph using the tensor product. The structure graph S consists of a $T \times T$ grid of (ordinary) vertices with edges connecting all neighboring vertices in the corresponding direction of the subgraph. The relation graph R encodes the connections between vertices in neighboring supernodes in the corresponding direction. Moreover, the structure graphs themselves can be further factorized. For example, the structure graph S for the subgraph V can be expressed as the Cartesian product of a path of length T and an empty graph on T vertices. When the path length is a perfect power b^k (where $b, k \geq 1$ are integers), it can be further decomposed into a union of k factored graphs of complexities (b, k). Further details can be found in the full version of the paper.

3.2 Counting s-Cliques

In contrast to the first result, our second main result states that counting cliques on factored graphs is in **FPT**. For this result, we assume that all graphs are undirected.

▶ **Theorem 2** (Counting Clique Subgraphs is in **FPT**). *Let H be a clique on s vertices. Then, computing the number of exact copies of H in G, denoted $\#\mathbf{H}(G)$, is fixed-parameter tractable. In particular, there is an algorithm computing $\#\mathbf{H}(G)$ on factored graphs G of complexity (n,k) in $O\left(g(s,k)n^s\right)$ time for some function $g(s,k)$.*

In the technical overview, we use edge counting ($s = 2$) as an illustrative example. Note that if (u,v) is an edge, u, v must have the same dimension (Corollary 16), so we may count the edges in each dimension separately. Thus, fix a dimension $d \leq k$.

There are at most 2^k factored components of dimension d. While some edges may belong to multiple factored components, we can use the inclusion-exclusion principle to avoid double-counting such edges. In particular, it suffices to consider counting the number of edges in an arbitrary intersection of factored components $G_{F_1} \cap \ldots \cap G_{F_m}$. Note that the number of such intersections, while being a large function of k, is crucially independent of n.

Fix such an intersection and consider a pair of vertices u, v with $u = (u_1, \ldots, u_d)$ and $v = (v_1, \ldots, v_d)$. Note that by determining whether u_i, v_i are equal or adjacent for all $i \in [k]$, we can infer from the factored graph tree structure whether $(u,v) \in E(G)$. Thus, we can categorize all pairs of vertices into 2^{2k} groups based on the relations $u_i = v_i$ and $(u_i, v_i) \in E(G_i)$. Note also that these groups are disjoint. Next, we collect the subset of groups that form edges in the current intersection $G_{F_1} \cap \ldots \cap G_{F_m}$. To count the number of edges, it then suffices to sum up the sizes of each group in this collection. For a given group, we can determine its size in $O(kn^2)$ time since the relations on each coordinate are independent. In particular, we can count the number of pairs of vertices in each input graph satisfying the relevant constraints and take the product over all input graphs.

3.3 Reachability

Finally, we study the reachability problem on factored graphs.

▶ **Theorem 3** (**XNL**-completeness of Reachability). *Reachability on factored graphs is* **XNL***-complete under* **FPT***-reductions. Furthermore, the following are equivalent:*

1. *There is a constant C such that* $\mathbf{NL} \subseteq \mathbf{DTIME}(n^C)$.
2. $\mathbf{XNL} \subseteq \mathbf{FPT}$.

We begin with an overview of the proof for the **XNL**-completeness result, followed by that for the equivalence result. In fact, we will see that both proofs rely on the same major building blocks in slightly different ways.

XNL-completeness

We follow the standard framework for showing that reachability is **NL**-complete [57].

Membership in **XNL** can be established using the same algorithm that shows the ordinary reachability problem is in **NL**, even if the input is now given as a factored graph. As in the standard proof, we non-deterministically guess the next vertex in the path and thus only require $O(k \log n)$-space for factored graphs $G = f(G_1, \ldots, G_k)$ of complexity (n,k). Formally, each vertex in G can be specified by $O(k \log n)$ bits, since each vertex is a tuple of at most k coordinates and each coordinate is a vertex in an input graph G_i with at most n vertices. Thus, the algorithm requires $O(k \log n)$ bits to write down the current vertex and, since the factored graph has at most n^k vertices, at most $O(k \log n)$ bits to keep track of the number of steps taken so far.

For **XNL**-hardness, suppose we have some language $L \in \mathbf{XNL}$. Then, there exists a nondeterministic Turing machine M deciding L using $f(k) \log n$-space on inputs of length n and parameter k, for some function f of k. The standard reduction creates an explicit

configuration graph where each vertex encodes a tuple consisting of state, input and work tape positions, and a setting of the work tape. Since the work tape has length $f(k) \log n$, there are at least $n^{f(k)}$ distinct settings of the work tape. Thus, even though reachability is computable in linear time, the size of the graph already depends exponentially on $f(k)$. However, we are able to exploit the locality of Turing machine operations so that the configuration graph can in fact be encoded by a factored graph of complexity $(\text{poly}(n), \text{poly}(f(k)))$. Given some configuration of M on input x (a tuple of state, tape head locations and work tape contents), we split the configuration into segments, where each segment only contains $\log n$ contiguous bits of the work tape contents. Note that each segment, even if it encodes a state and tape head locations, only has $n^{O(1)}$ possible values, since one segment of the work tape only has $\log n$ bits. Thus, we can represent all possible configurations of a segment explicitly using a graph with $n^{O(1)}$ vertices, while a product of $f(k)$ segment graphs can explicitly represent any full configuration of M on input x.

It remains to express the appropriate transitions between configurations using a factored graph. There are two types of transitions: *intra*-segment transitions, where the work tape head stays within the same segment, and *inter*-segment transitions, where the work tape head moves from one segment to another (adjacent) segment. In either case, at most two segments of the work tape are active, where a segment is active during a transition if the work tape head either starts or ends in the segment and inactive otherwise. Thus, we can express individual segments (or pairs of segments) explicitly using input graphs of size $n^{O(1)}$. Since the active segments and inactive segments do not interact, we can encode all *intra*-segment transitions of a single segment using factored graphs of complexity $(n^{O(1)}, f(k))$. Similarly, we can encode all *inter*-segment transitions between a single pair of adjacent segments. Summing over all $O(f(k))$ segments and pairs of adjacent segments, we encode the configuration graph in a factored graph of complexity $(n^{O(1)}, O(f(k)^2))$.

We now briefly describe how the locality of Turing machine computation allows us to express the configuration graph of M on x using factored graphs. In the overview, we describe only *intra*-segment transitions. First, note that if the work tape head is not currently placed in a segment, the work tape contents of this segment cannot change. We thus define an *inactive* graph, where for every possible work tape content, we create a self-loop vertex. If the work tape head is currently placed in the segment, the work tape contents may change. Thus, we define an *active* graph with all possible configurations on a given segment as nodes, and edges encoding valid transitions. For a given segment, taking the tensor product of the active graph for this segment and the inactive graph for all other segments encodes all *intra*-segment transitions. A similar construction can be used to construct factored graphs that encode *inter*-segment transitions.

Equivalence to the Open Problem

For the forward direction, suppose there exists an absolute constant C such that $\mathbf{NL} \subseteq \mathbf{DTIME}(n^C)$. To show that $\mathbf{XNL} \subseteq \mathbf{FPT}$, it now suffices to show reachability on factored graphs is in $\mathbf{FPT}$ due to its $\mathbf{XNL}$-completeness under $\mathbf{FPT}$-reductions. Note that the algorithm used in the proof of $\mathbf{XNL}$-membership is in fact an $\mathbf{NL}$ algorithm for each fixed parameter k. Thus, if $\mathbf{NL} \subseteq \mathbf{DTIME}\left(n^C\right)$ for some C independent of k, then for any fixed k, reachability on factored graphs of complexity (n, k) can be computed in time $O(k^C n^{2C})$ (as factored graphs of this complexity have input size $O(kn^2)$) and is therefore in $\mathbf{FPT}$.

Conversely, if $\mathbf{XNL} \subseteq \mathbf{FPT}$, then in particular, reachability on factored graphs is in $\mathbf{FPT}$. Now, consider any language $L \in \mathbf{NL}$ with its associated nondeterministic Turing machine M that decides L in $S \log n$ space for some constant S. Using the same construction as in the

proof of **XNL**-hardness, we construct a factored graph G of complexity $(n^{O(1)}, O(S^2))$ where solving reachability on G simulates M. Since reachability on factored graphs is in **FPT**, it follows that reachability on G can be solved in time $O(f(S^2)n^{O(1)})$ for some function f, where the exponent on n is independent of S, hence independent of the specific language $L \in \mathbf{NL}$. This shows that $\mathbf{NL} \subseteq \mathbf{DTIME}(n^C)$ for some absolute constant C.

4 Conclusion and Future Work

We have studied the computational complexity of various problems on factored graphs. Even among problems with polynomial time algorithms on explicit graphs, we have shown that their parameterized complexity when the input is represented as a factored graph can differ quite drastically. In the context of parameterized complexity, we have sought to characterize which problems on factored graphs are in **FPT**. On the positive side, counting the number of copies of a small clique is in **FPT**. On the negative side, LFMIS is *unconditionally* not in **FPT**. Finally, we show that determining whether reachability is in **FPT** is equivalent to determining whether $\mathbf{NL} \subseteq \mathbf{DTIME}(n^{O(1)})$.

Can the unconditional lower bounds for LFMIS on factored graphs be used to prove similar unconditional lower bounds for other parameterized problems? One obstacle to doing this is the gap in complexity between the **P**-complete LFMIS problem and the easily parallelizable problems that form the bulk of the literature in parameterized complexity. However, reductions in fine-grained complexity often cut across traditional complexity classes, e.g., [46]. So we do not know a reason why this should not also be the case here. Either finding such unconditional results or explaining their impossibility would both be interesting. We could also hope to prove similar results for other **P**-complete problems.

While our lower bounds separate the problems of interest from **FPT**, they do not rule out significant improvements on the naive $n^{O(k)}$ algorithm of computing the factored graph G explicitly and solving the problem on G itself. For example, our LFMIS lower bound only unconditionally rules out algorithms with time $n^{o(\sqrt{k})}$. An interesting open problem is to provide a more *fine-grained* analysis of the complexity of factored graph problems, possibly distinguishing between the number of product and union operations.

In this work, we have chosen to study factored graphs under graph products and unions, specifically the Cartesian and tensor products. A natural extension is to consider other products and operations on graphs, or other interesting objects. Factored problems on bit strings were introduced in [21], and implicitly on integer-valued vectors in [3]. Do similar results hold for factored problems for these input domains and operations?

References

1 Akeo Adachi, Shigeki Iwata, and Takumi Kasai. Some combinatorial game problems require omega(n^k) time. *J. ACM*, 31(2):361–376, March 1984. doi:10.1145/62.322433.

2 Enric Boix Adserà, Matthew S. Brennan, and Guy Bresler. The average-case complexity of counting cliques in erdős-rényi hypergraphs. *CoRR*, abs/1903.08247:FOCS19–39, 2019. doi:10.48550/arXiv.1903.08247.

3 Haozhe An, Mohit Gurumukhani, Russell Impagliazzo, Michael Jaber, Marvin Künnemann, and Maria Paula Parga Nina. The fine-grained complexity of multi-dimensional ordering properties. *Algorithmica*, 84(11):3156–3191, 2022. doi:10.1007/s00453-022-01014-x.

4 Sanjeev Arora, David Steurer, and Avi Wigderson. Towards a study of low-complexity graphs. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 119–131. Springer, Springer, 2009. doi:10.1007/978-3-642-02927-1_12.

5 Jérémy Barbay, Luca Castelli Aleardi, Meng He, and J. Ian Munro. Succinct representation of labeled graphs. In Takeshi Tokuyama, editor, *Algorithms and Computation, 18th International Symposium, ISAAC 2007, Sendai, Japan, December 17-19, 2007, Proceedings*, volume 4835 of *Lecture Notes in Computer Science*, pages 316–328. Springer, Springer, 2007. doi:10.1007/978-3-540-77120-3_29.

6 Christoph Berkholz. Lower bounds for existential pebble games and k-consistency tests. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 25–34. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.14.

7 Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In Amihood Amir and Laxmi Parida, editors, *Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21-23, 2010. Proceedings*, volume 6129 of *Lecture Notes in Computer Science*, pages 138–150. Springer, Springer, 2010. doi:10.1007/978-3-642-13509-5_13.

8 Guy E. Blelloch, Jeremy T. Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In Guy E. Blelloch and Maurice Herlihy, editors, *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 308–317. ACM, 2012. doi:10.1145/2312005.2312058.

9 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. *CoRR*, abs/2105.14882, 2021. doi:10.48550/arXiv.2105.14882.

10 Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Ann. Pure Appl. Log.*, 84(1):119–138, 1997. Asian Logic Conference. doi:10.1016/S0168-0072(95)00020-8.

11 Chris Calabro, Russell Impagliazzo, Valentine Kabanets, and Ramamohan Paturi. The complexity of unique k-sat: An isolation lemma for k-cnfs. *J. Comput. Syst. Sci.*, 74(3):386–393, 2008. doi:10.1016/j.jcss.2007.06.015.

12 Neil J. Calkin, Alan M. Frieze, and Ludek Kucera. On the expected performance of a parallel algorithm for finding maximal independent subsets of a random graph. *Random Struct. Algorithms*, 3(2):215–222, 1992. doi:10.1002/rsa.3240030210.

13 Ashok K. Chandra and Larry J. Stockmeyer. Alternation. In *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*, pages 98–108. IEEE Computer Society, 1976. doi:10.1109/SFCS.1976.4.

14 Hubie Chen. Periodic constraint satisfaction problems: Polynomial-time algorithms. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 199–213. Springer, Springer, 2003. doi:10.1007/978-3-540-45193-8_14.

15 Hubie Chen. Periodic constraint satisfaction problems: Tractable subclasses. *Constraints An Int. J.*, 10(2):97–113, 2005. doi:10.1007/s10601-005-0551-z.

16 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.

17 Yijia Chen, Jörg Flum, and Martin Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 13–29. IEEE Computer Society, 2003. doi:10.1109/CCC.2003.1214407.

18 Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Inf. Control.*, 64(1-3):2–21, 1985. doi:10.1016/S0019-9958(85)80041-3.

19 Stephen A. Cook and Pierre McKenzie. Problems complete for deterministic logarithmic space. *J. Algorithms*, 8(3):385–394, 1987. doi:10.1016/0196-6774(87)90018-6.

20 Don Coppersmith, Prabhakar Raghavan, and Martin Tompa. Parallel graph algorithms that are efficient on average. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 260–269. IEEE, IEEE Computer Society, 1987. doi:10.1109/SFCS.1987.46.

21 Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. New techniques for proving fine-grained average-case hardness. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 774–785. IEEE, IEEE, 2020. doi:10.1109/FOCS46700.2020.00077.

22 Constantinos Daskalakis, Alex Fabrikant, and Christos H. Papadimitriou. The game world is flat: The complexity of nash equilibria in succinct games. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 513–524. Springer, Springer, 2006. doi:10.1007/11786986_45.

23 Konstantinos Daskalakis and Christos H. Papadimitriou. The complexity of games on highly regular graphs. In Gerth Stølting Brodal and Stefano Leonardi, editors, *Algorithms - ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings*, volume 3669 of *Lecture Notes in Computer Science*, pages 71–82. Springer, Springer, 2005. doi:10.1007/11561071_9.

24 Laxman Dhulipala, Quanquan C. Liu, Julian Shun, and Shangdi Yu. Parallel batch-dynamic k-clique counting. In Michael Schapira, editor, *2nd Symposium on Algorithmic Principles of Computer Systems, APOCS 2020, Virtual Conference, January 13, 2021*, pages 129–143. SIAM, SIAM, 2021. doi:10.1137/1.9781611976489.10.

25 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.

26 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theor. Comput. Sci.*, 141(1&2):109–131, 1995. doi:10.1016/0304-3975(94)00097-3.

27 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

28 Rodney G. Downey, Michael R. Fellows, and Kenneth W. Regan. Descriptive complexity and the W hierarchy. In Paul Beame and Samuel R. Buss, editors, *Proof Complexity and Feasible Arithmetics, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, April 21-24, 1996*, volume 39 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 119–134. Citeseer, DIMACS/AMS, 1996. doi:10.1090/dimacs/039/07.

29 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space complexity of parameterized problems. In Dimitrios M. Thilikos and Gerhard J. Woeginger, editors, *Parameterized and Exact Computation - 7th International Symposium, IPEC 2012, Ljubljana, Slovenia, September 12-14, 2012. Proceedings*, volume 7535 of *Lecture Notes in Computer Science*, pages 206–217, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-33293-7_20.

30 Arash Farzan and J. Ian Munro. Succinct representations of arbitrary graphs. In Dan Halperin and Kurt Mehlhorn, editors, *Algorithms - ESA 2008, 16th Annual European Symposium, Karlsruhe, Germany, September 15-17, 2008. Proceedings*, volume 5193 of *Lecture Notes in Computer Science*, pages 393–404. Springer, Springer, 2008. doi:10.1007/978-3-540-87744-8_33.

31 Joan Feigenbaum, Daphne Koller, and Peter W. Shor. A game-theoretic classification of interactive complexity classes. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*, pages 227–237. IEEE, IEEE Computer Society, 1995. doi:10.1109/SCT.1995.514861.

32 Jörg Flum and Martin Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM J. Comput.*, 31(1):113–145, 2001. doi:10.1137/S0097539799360768.

33 Jörg Flum and Martin Grohe. Describing parameterized complexity classes. *Inf. Comput.*, 187(2):291–319, 2003. doi:10.1016/S0890-5401(03)00161-5.

34 Lance Fortnow, Russell Impagliazzo, Valentine Kabanets, and Christopher Umans. On the complexity of succinct zero-sum games. *Comput. Complex.*, 17(3):353–376, 2008. doi:10.1007/s00037-008-0252-2.

35 Hana Galperin and Avi Wigderson. Succinct representations of graphs. *Inf. Control.*, 56(3):183–198, 1983. doi:10.1016/S0019-9958(83)80004-7.

36 Gianluigi Greco, Enrico Malizia, Luigi Palopoli, and Francesco Scarcello. The complexity of the nucleolus in compact games. *ACM Trans. Comput. Theory*, 7(1):3:1–3:52, 2014. doi:10.1145/2692372.2692374.

37 Shreya Gupta, Boyang Huang, Russell Impagliazzo, Stanley Woo, and Christopher Ye. The computational complexity of factored graphs. *CoRR*, abs/2407.19102, 2024. doi:10.48550/arXiv.2407.19102.

38 Xin He, Ming-Yang Kao, and Hsueh-I Lu. Linear-time succinct encodings of planar graphs via canonical orderings. *SIAM J. Discret. Math.*, 12(3):317–325, 1999. doi:10.1137/S0895480197325031.

39 Monika Henzinger, Andrea Lincoln, and Barna Saha. The complexity of average-case dynamic subgraph counting. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 459–498. SIAM, SIAM, 2022. doi:10.1137/1.9781611977073.23.

40 Franz Höfting and Egon Wanke. Polynomial algorithms for minimum cost paths in periodic graphs. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 493–499. ACM/SIAM, 1993. URL: http://dl.acm.org/citation.cfm?id=313559.313868.

41 Bruce Hoppe and Éva Tardos. The quickest transshipment problem. *Math. Oper. Res.*, 25(1):36–62, 2000. doi:10.1287/moor.25.1.36.15211.

42 Shweta Jain and C. Seshadhri. The power of pivoting for exact clique counting. In James Caverlee, Xia (Ben) Hu, Mounia Lalmas, and Wei Wang, editors, *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, pages 268–276. ACM, 2020. doi:10.1145/3336191.3371839.

43 Neil D. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci.*, 11(1):68–85, 1975. doi:10.1016/S0022-0000(75)80050-X.

44 Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 334–343. ACM, 1988. doi:10.1145/62212.62244.

45 Takumi Kasai, Akeo Adachi, and Shigeki Iwata. Classes of pebble games and complete problems. *SIAM J. Comput.*, 8(4):574–586, 1979. doi:10.1137/0208046.

46 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.21.

47 Sebastian Maneth and Fabian Peternek. A survey on methods and systems for graph compression. *CoRR*, abs/1504.00616, 2015. arXiv:1504.00616, doi:10.48550/arXiv.1504.00616.

48 Madhav V. Marathe, Harry B. Hunt III, Daniel J. Rosenkrantz, and Richard Edwin Stearns. Theory of periodically specified problems: Complexity and approximability. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity, Buffalo, New York, USA, June 15-18, 1998*, page 106. IEEE, IEEE Computer Society, 1998. doi:10.1109/CCC.1998.694596.

49 Madhav V. Marathe, Harry B. Hunt III, Richard Edwin Stearns, and Venkatesh Radhakrishnan. Approximation schemes for pspace-complete problems for succinct specifications (preliminary version). In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 468–477. ACM, 1994. doi:10.1145/195058.195233.

50 Satoru Miyano. The lexicographically first maximum subgraph problems: P-completeness and NC algorithms. *Math. Syst. Theory*, 22(1):47–73, 1989. doi:10.1007/BF02088292.

51 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 026(2):415–419, 1985. URL: http://eudml.org/doc/17394.

52 James B. Orlin. The complexity of dynamic languages and dynamic optimization problems. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 218–227. ACM, 1981. doi:10.1145/800076.802475.

53 Christos H. Papadimitriou and Tim Roughgarden. Computing correlated equilibria in multi-player games. *J. ACM*, 55(3):14:1–14:29, 2008. doi:10.1145/1379759.1379762.

54 Victor Parque and Tomoyuki Miyashita. On succinct representation of directed graphs. In *2017 IEEE International Conference on Big Data and Smart Computing, BigComp 2017, Jeju Island, South Korea, February 13-16, 2017*, pages 199–205. IEEE, IEEE, 2017. doi:10.1109/BIGCOMP.2017.7881738.

55 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075. SIAM, SIAM, 2010. doi:10.1137/1.9781611973075.86.

56 Jessica Shi, Laxman Dhulipala, and Julian Shun. Parallel clique counting and peeling algorithms. In Michael Bender, John Gilbert, Bruce Hendrickson, and Blair D. Sullivan, editors, *Proceedings of the 2021 SIAM Conference on Applied and Computational Discrete Algorithms, ACDA 2021, Virtual Conference, July 19-21, 2021*, pages 135–146. SIAM, SIAM, 2021. doi:10.1137/1.9781611976830.13.

57 Michael Sipser. Introduction to the theory of computation. *SIGACT News*, 27(1):27–29, 1996. doi:10.1145/230514.571645.

58 Ryuhei Uehara. A measure of parallelization for the lexicographically first maximal subgraph problems. In Rolf H. Möhring, editor, *Graph-Theoretic Concepts in Computer Science, 23rd International Workshop, WG '97, Berlin, Germany, June 18-20, 1997, Proceedings*, volume 1335 of *Lecture Notes in Computer Science*, pages 333–341. Springer, Springer, 1997. doi:10.1007/BFb0024508.

59 Ryuhei Uehara. Another measure for the lexicographically first maximal subgraph problems and its threshold value on a random graph. In *1999 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '99), 23-25 June 1999, Fremantle, Australia*, pages 350–355. IEEE, IEEE Computer Society, 1999. doi:10.1109/ISPAN.1999.778963.

60 Ryuhei Uehara. A measure for the lexicographically first maximal independent set problem and its limits. *Int. J. Found. Comput. Sci.*, 10(4):473–482, 1999. doi:10.1142/S0129054199000332.

61 Emanuele Viola and Avi Wigderson. Local expanders. *Comput. Complex.*, 27(2):225–244, 2018. doi:10.1007/s00037-017-0155-1.

62 Klaus W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986. doi:10.1007/BF00289117.

63 Egon Wanke. Paths and cycles in finite periodic graphs. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *Mathematical Foundations of Computer Science 1993, 18th International Symposium, MFCS'93, Gdansk, Poland, August 30 - September 3, 1993, Proceedings*, volume 711 of *Lecture Notes in Computer Science*, pages 751–760. Springer, Springer, 1993. doi:10.1007/3-540-57182-5_66.