Fixed Structure Complexity

Yonatan Aumann and Yair Dombb

Department of Computer Science
Bar-Ilan University
Ramat Gan 52900, Israel
aumann@cs.bu.ac.il, yairbiu@gmail.com

Abstract. We consider a non-standard parametrization, where, for problems consisting of a combinatorial structure and a number, we parameterize by the combinatorial structure, rather than by the number. For example, in the *Short-Nondeterministic-Halt* problem, which is to determine if a nondeterministic machine M accepts the empty string in t steps, we parameterize by $|M|$, rather than t. We call such parametrization *fixed structure parametrization*. Fixed structure parametrization not only provides a new set of parameterized problems, but also results in problems that do not seem to fall within the classical parameterized complexity classes. In this paper we take the first steps in understanding these problems. We define fixed structure analogues of various classical problems, including graph problems, and provide complexity, hardness and equivalence results.

1 Introduction

Motivating Examples. Consider the classical *Tiling* problem. Given a set of tiles T and integer t, decide whether it is possible to tile the $t \times t$ area with tiles from T. The general problem is NP-complete. Parameterizing by t it is W[1]-complete. But what if we parameterize by $|T|$, is the problem FPT? where does it fall in the fixed parameter hierarchy? The naive algorithm takes $O(|T|^{t^2})$ steps which does not even constitute an XP algorithm for the parameter $|T|$. Is the problem in XP? (I.e. is there an $O(t^{f(|T|)})$ algorithm?

Next, consider the following generalization of the Hamiltonian cycle problem. Given a graph G (on n vertices) and integers $\boldsymbol{m} = (m_1, \ldots, m_n)$, determine if there is a cycle that visits node v_i exactly m_i times. Clearly, this problem is NP-complete. If parameterized by $\boldsymbol{m}$ it is para-NP complete. But what if we parameterize by $|G|$? Does the problem then become FPT? Is it XP? Para-NP-complete?

This paper aims at developing the theory and tools to answer questions such as the above.

Fixed Structure Parametrization. In general, many problems can naturally be viewed as composed of two parts: (i) a combinatorial structure (e.g. a set of tiles, a graph), and (ii) a number(s) (size of area to be tiled, number of visits). The standard parametrization most often takes the "number part" as the parameter, and asks whether the problem is polynomial in the combinatorial structure.

M. Grohe and R. Niedermeier (Eds.): IWPEC 2008, LNCS 5018, pp. 30–42, 2008.

Here, we reverse the question, taking the combinatorial structure as the parameter, and asking whether the problem is polynomial in the "number part".[1] Accordingly, we call our parametrization *fixed structure parametrization*, producing *fixed structure problems*. Interestingly, despite the extended literature on parameterized complexity, *fixed structure* parameterizations have enjoyed little consideration, and in no systematic way.

As it turns out, *fixed structure* problems not only provide a new set of parameterized problems, but also result in problems that do not seem to fall within the classical parameterized complexity classes. Rather they seem to form complexity classes of their own. In this paper we aim at taking the first steps in understanding this *fixed structure complexity*.

Summary of Results. Some fixed structure problems, such as the generalized Hamiltonian cycle problem mentioned above, we could show to be FPT. For others, we identified three different equivalence classes of fixed structure problems, all of which seem to consist of problems that are not FPT. The first of these classes is defined by the following fixed structure variant of short-NSTM-Halt:

Fixed-Structure-short-NTM-Exact-Halt (*FS-Exact-Halt*) Instance: Non-deterministic Turing machine M; integer t (in unary) Parameter: $\|M\|$ Problem: Does M accept the empty string in exactly t steps?

The naive algorithm for this problems runs in time $\Theta(|M|^t)$. Thus, the question is not only if the problem is FPT, but also if it is altogether in XP. We prove:

Theorem 1. *If FS-Exact-Halt* $\in XP$ *then NEXP=EXP.*

Accordingly, in this work we also consider XP reductions, not only FPT ones (when the reductions are also FPT, we note so). We prove:

Theorem 2. *The following are equivalent to FS-Exact-Halt:*

- *Fixed-structure tiling of the* $t \times t$ *torus (under FPT reductions).*
- *Fixed-structure clique, independent set and vertex-cover (under XP reductions).*

The exact definitions of the fixed-structure versions of the graph problems are provided in Section 3. We note that the classical reductions, used for proving the NP-completeness of the above problems, often fail in our fixed structure setting, as they "hardwire" the number part into combinatorial structure (for more details see the proof of Theorem 2 in Section 4).

We define two additional fixed-structure equivalence classes, using similar machine characterizations:

[1] Clearly, there is no formal distinction between the "number part" of a problem and the "combinatorial structure"; both are strings over some alphabet. In practice, however, the two elements are frequently clearly distinct.

FIXED-STRUCTURE-SHORT-NTM-HALT (*FS-Halt*) Instance: Non-deterministic Turing machine M; integer t (in unary) Parameter: $\|M\|$ Problem: Does M accept the empty string in at most t steps?

FIXED-STRUCTURE-SHORT-NTM-NOT-HALT (*FS-Not-Halt*) Instance: Non-deterministic Turing machine M; integer t (in unary) Parameter: $\|M\|$ Problem: Does M have a computation on the empty string not halting for at least t steps?

Note that in all three problems we did not specify the number of tapes. The reason is that we prove that any of them is equivalent for any number of tapes, even under FPT reductions (unlike the case for the classical parametrization [1]).

Under classical complexity, as well under the standard parametrization, the three problems *FS-Halt*, *FS-Not-Halt* and *FS-Exact-Halt* are equivalent. Interestingly, under the fixed-structure parametrization this is not the case, even under XP reductions (hence clearly also under FPT reductions). We prove:

Theorem 3

1. *FS-Halt* $\leq^{fpt}$ *FS-Exact-Halt and FS-Not-Halt* $\leq^{fpt}$ *FS-Exact-Halt, but*
2. *If FS-Halt* $\equiv^{XP}$ *FS-Exact-Halt or FS-Not-Halt* $\equiv^{XP}$ *FS-Exact-Halt then NEXP=EXP.*

Some problems we show are equivalent to *FS-Halt* or to *FS-Not-Halt*:

Theorem 4

- *Fixed-Structure-Short-Post-Correspondence and Fixed-Structure-Short-Grammar-Derivation are equivalent to FS-Halt (under FPT reductions).*
- *Fixed-Structure-Restricted-Tiling (i.e. tiling with a specified origin tile) of the $t \times t$ plane is equivalent to FS-Not-Halt (under FPT reductions).*

Thus, in the fixed structure setting, tiling of the plane and tiling of the torus are *not* equivalent.

Related Work. To the best of our knowledge, there has been no systematic analysis of fixed-structure parametrization. Of the 376 parameterized problems described in the *Compendium of Parameterized Problems* [2], we identified few problems that we would classify as fixed-structure, in the sense we consider here. Notably, the following two: the parameterizations of *Bounded-DFA-Intersection* by k, $|\Sigma|$, and q, which is reported open in [3], and the parametrization of the *Rush-Hour-Puzzle* by C (the set of cars), which is shown to be FPT in [4], by exhaustive search.

Related to fixed-structure problems are those problems where instances are composed of *two* combinatorial structures, and one of the structures is taken as the parameter. Many of the database query problems are of this type [5,6]. In this case there are two combinatorial structures – the query Q and the database d, and the parameter is either $|Q|$ or the number of variables within. The full

characterization of the complexity of several of these problems is still open [5]. It is interesting to note that Vardi's initial work [7] considers both parameterizations – both by $|Q|$ and by $|d|$ (though, naturally, without explicit use of parameterized complexity terminology).

Other problems with two combinatorial structures are ordering problems on graphs, lattices and the like, e.g. given graphs H and G decide whether H is a minor of G (see [8,9]). Some of these problems have been resolved, but the complexity of others is still open. While these problems are not strictly fixed-structure in the sense we consider here, it would be interesting to see if the theory developed here may be relevant to these problems as well.

Organization of the Paper. Unfortunately, the space limitations of this extended abstract allow us to provide only a small fraction of the results and proofs. In particular, all the positive results showing inclusion in FPT are omitted. Here, we focus on hardness results alone. The full set of results and proofs will appear in the full version. The rest of the paper is organized as follows. In the next section we prove Theorems 1 and 3. Section 3 introduces the exact definition of the fixed-structure versions of the graph problems. Section 4 provides the highlights of the proof of Theorem 2. We conclude with open problems in Section 5.

2 Complexity and Hardness

We now prove Theorems 1 and 3, as well as an additional theorem.

Theorem 1 (repeated) *If FS-Exact-Halt* $\in XP$ *then EXP = NEXP.*

Proof. Let $L \in$ NEXP. Then there exists a nondeterministic Turing machine M_L, which for some constant c decides on every input x whether $x \in L$ in time $2^{|x|^c}$. We construct a new Turing machine M'_L as follows. On the empty tape, M'_L first nondeterministically chooses some $x \in \Sigma^*$, and then runs M_L on this x. If M_L accepts x, then M'_L idles until *exactly* $2^{|x|^c} + x$ steps have elapsed since the beginning of its run, and then accepts.[2] Otherwise, M'_L rejects.

Assume that *FS-Exact-Halt* $\in XP$. Then there exists an algorithm A and an arbitrary function $f(\cdot)$, such that $A(M,t)$ decides in $|(M,t)|^{f(|M|)} + f(|M|)$ steps whether M has a computation that accepts the empty string in exactly t steps. Given an input x, for which we want to decide whether $x \in L$, we simply run A on the input (M'_L, t), with $t = 2^{|x|^c} + x$. Note that the function $x \hookrightarrow 2^{|x|^c} + x$ is a bijection, and therefore M'_L accepts in exactly t steps iff $x \in L$. In addition, A runs in time $|(M'_L,t)|^{f(|M'_L|)} + f(|M'_L|) = O(2^{x^{c'}})$, for some c' depending only on M'_L. Thus, $L \in$ EXP and EXP = NEXP. □

Next, we show an interesting, albeit easy, result, which will also serve us in our next proof:

Theorem 5. *FS-Halt and FS-Not-Halt are non-uniform FPT.*

[2] Note that it must be shown that this counting can be performed in the fixed structure setting. The proof is omitted here and provided in the full version.

Proof. Consider the *FS-Halt* problem (the proof for *FS-Not-Halt* is analogous). To prove that the problem is non-uniform FPT we need to construct, for every size k, an algorithm A_k, such that for every M, with $|M| = k$, and every t, decides whether $(M, t) \in$ *FS-Halt* in time $O(t^\alpha)$ for some constant α. We do so by simply creating a table exhaustively listing, for each Turing machine M with $|M| = k$, the minimum number of steps in which M can accept on the empty string. Given an input (M, t) the algorithm consults this table, comparing t to this minimum. Clearly, the algorithm is correct and runs in polynomial time. Note, however, that constructing this table is, in general, undecidable. □

Theorem 3 (repeated)

1. *FS-Halt* $\leq^{fpt}$ *FS-Exact-Halt and FS-Not-Halt* $\leq^{fpt}$ *FS-Exact-Halt, but*
2. *If FS-Halt* $\equiv^{XP}$ *FS-Exact-Halt or FS-Not-Halt* $\equiv^{XP}$ *FS-Exact-Halt then NEXP=EXP.*

Proof. Due to lack of space, the proof of (1) is omitted. The proof of (2) combines the techniques of Theorems 1 and 5. Suppose that R is an XP reduction from *FS-Exact-Halt* to *FS-Halt*. Let L, M_L and M'_L be as defined in the proof of Theorem 1. Then, for any x, $x \in L$ iff M'_L accepts the empty string in exactly $g(x) = 2^{|x|^c} + x$ steps. Denote $R(M'_L, t) = (N, s)$. Then, since R is an XP reduction, $|N|$ must be bounded by a function of M'_L alone. Thus, there is only a finite number such N's (that are the result of applying R to M'_L for some t). Thus, after the reduction we need only consider a finite number of slices of *FS-Halt*. By Theorem 5 there is a polynomial algorithm for each of these slices. Hence, combining R with the union of these algorithms we obtain an EXP algorithm for L. Note however, that this proof is non-constructive, as are the algorithms provided by Theorem 5. □

3 Defining Fixed-Structure Graph Problems

We are interested in defining fixed-structure versions of common graph problems. This seems easy: many graph problems are naturally composed of a graph and a number. Thus, to obtain a fixed structure version simply parameterize by the graph structure. However, this approach results in non-interesting problems. The reason is that the size of the graph necessarily bounds "the number" (e.g. the number of colors is at most the number of nodes), and the resulting problems are trivially FPT. Thus, in order to obtain meaningful fixed structure graph problems, we must be able to define *families* of graphs (of increasing sizes), all of which share a *common* underlying structure. In a way, the grid is an example of such a graph family; it comes in many sizes, but all share a common core structure. Cliques, hypercubes and cycles are other examples of such graph families.

We now give a general definition of such graph families, which we call *parametric graphs*. The basic idea is to define the graphs using *expressions* that accept *parameters*. The expression defines a graph by applying standard *graph operations* to a set of *base graphs* and *parameters*. The *base graphs* can be any explicitly represented graphs. The *operations* combine these graphs to obtain larger and more complex ones. The operations we consider are:

- Union: for graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the union graph $G = G_1 \cup G_2$ is the graph $G = (V_1 \cup V_2, E_1 \cup E_2)$.
- Multiplication (by a scalar): for a graph G and integer i, the i-multiplicity of G, denoted by $i \cdot G$, is the union of i separate copies of G.
- Sum: defined on graphs *over the same set of vertices*, $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$. The sum graph $G = G_1 + G_2$ has the union of the edges from both graphs, $G = (V, E_1 \cup E_2)$.
- Direct product (also known as *tensor product*): for graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, their direct product is the graph $G = G_1 \times G_2 = (V_1 \times V_2, E)$ such that $((v_1, v_2), (w_1, w_2)) \in E$ iff $(v_1, w_1) \in E_1$ and $(v_2, w_2) \in E_2$.

Using these operations it is possible to construct large and complex graphs from smaller ones. For example, the cycle with 14 vertices can be constructed as:

$C_{14} =$ 

Since the sum operation requires that both graphs share the same set of vertices, w.l.o.g. we assume that the vertex set of any graph we consider is simply the integers, $\{1, \ldots, |V|\}$. Thus, following a union, multiplication or product operation, the vertices must be renamed. We do so systematically "lexicographically", as follows. For the the product and multiplication operations, the new order is simply the lexicographic order on the new vertices. For $G = G_1 \cup G_2$, vertexes of each graphs retain their original order, and all those of G_1 are come before those of G_2.

The multiplication operation allows us to define expressions that accept parameters. This way a single expression can define graphs of varying sizes, all sharing a common, underlying combinatorial structure. Thus, we define a *parametric graph* as an expression of the above format that accepts parameters. Using this notion we can define fixed-structure versions of classical graph problems. For example, the fixed-structure version of Independent-Set is the following:

FIXED STRUCTURE INDEPENDENT SET $(\cup, \cdot, +, \times)$
Instance: a *parametric graph* expression G (using the operations $\cup, \cdot, +$, and $\times$); a vector $\boldsymbol{t}$ of integer parameter values to G (in unary); integer ψ.
Parameter: $|G|$
Problem: Does $G(\boldsymbol{t})$ have an independent set of size ψ?

Note the complexity of the problem may depend on the set of operations used in the graph expressions. Hence, the definition of the problem explicitly lists these operations ($(\cup, \cdot, +, \times)$ in our case). Also, note that the problem is not necessarily in NP, as the size of the resultant graph is not polynomially bounded in the input size.

Fixed-structure versions for other graph problems are defined similarly.

4 Problems Equivalent to *FS-Exact-Halt*

We now provide an outline for the proof of Theorem 2. Unfortunately, we cannot provide all the details, but do hope that our exposition provides a flavor of the

problems one encounters in fixed structure reductions, and some of the methods we use to overcome these problems.

Lemma 1. *FS-Exact-Halt* $\leq^{fpt}$ *Fixed-structure-Torus-Tiling.*

Proof. Let M be a Turing machine and t an integer. We construct a tile set $T = T(M)$ and an integer $s = s(t)$, such that T has a valid tiling of the $s \times s$ torus iff M accepts the empty string in exactly t steps. The core of the proof follows the reduction used for proving the undecidability of tiling of the infinite first-quadrant. The problem arises, however, when trying to convert this construction to the bounded case, as discussed below.

The basic idea of the reduction from Turing machine computation to tiling is to make each valid tiling represent a run of the Turing machine: every row in the tiling corresponds to a configuration, and one row can be placed on top of another only if the configuration corresponding to the bottom row yields the one corresponding to the top row. Provided that the first row represents the initial configuration of M, we obtain that the infinite first-quadrant can be tiled iff M has a non-halting computation (see, for example, [10] for more details).

In order to use this reduction for proving hardness of torus-tiling, it is not difficult to augment the construction, so that: (i) the row corresponding to the initial configuration can only be placed directly above a row corresponding to an accepting configuration, and (ii) the leftmost end of each row can be placed directly to the right of the rightmost end. Thus, if M accepts in t steps then the $t \times t$ torus can be tiled with $T(M)$. Unfortunately, the converse is not necessarily true. The problem is that the $t \times t$ torus can be split into smaller regions, each corresponding to a shorter accepting computations. For example, the 10×10 torus can be tiled with four copies of a tiling of 5×5 tori. Another subtle point is guaranteeing that the first row indeed corresponds to the machine's initial configuration. For the unbounded case, this is provided by a careful and complex construction, provided in [11]. Unfortunately, this construction does not seem to carry over to the bounded case.

If we were to prove standard NP-completeness, the following simple and standard construction solves both of the above problems. Let $T = T(M)$ be the tile set obtained by the unbounded reduction. We create a new tile set T', such that for each tile $z \in T$, we have (essentially) t^2 copies, $z^{(1,1)}, \ldots, z^{(t,t)}$, one for each torus location. It is now easy to configure the tiles such that tile $z^{(i,j)}$ can appear only at location (i, j) (for all z, i, j). In this way we have eliminated the possibility to cover the torus by copies of smaller tori. In addition, we can force the first row to whatever we wish, by eliminating all but the appropriate tiles for this row. This construction fails, however, in the fixed structure setting. The reason is that the reduction "hardwires" the number t into T'. By doing so, however, we have moved t into the "parameter" part of the instance, which is forbidden in parameterized reductions.

Thus, we provide a solution that works independently of t, as follows. We construct a set of "base tiles" upon which the original "machine simulation" tiles are then superimposed. The "base tiles" are constructed such that they only admit a specific tiling, which forces the tiling to "behave" as desired.

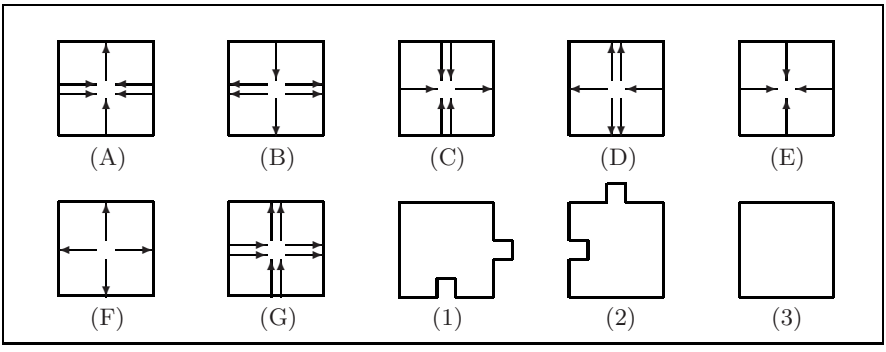

Fig. 1. The base tiles

The Base Tiling. We start by creating the tiles (A)-(G) depicted in Figure 1. One can observe that any torus can be tiled using this set of tiles, and that any tiling of an *odd* sized torus can be decomposed into rectangular regions such that (see Figure 2):

1. Tile (G) is placed at the bottom left corner of the region.
2. The rest of the bottom row is composed of tiles (A) and (B) only.
3. The rest of the leftmost column is composed of tiles (C) and (D) only.

We call such a region a *core-region*. Note that tile (G) can only be placed above tiles (D) or (G), and to the right of tiles (B) or (G). Thus, we obtain that if a core-region R is directly above another core-region S, then R and S have the same width. Likewise, if R is directly to the right of S, then R and S have the same height. Therefore, all core-regions must have the same size.

In order to force the core-regions to be *square*, we use the numbered tiles ((1)-(3)) of Figure 1, which will be superimposed on the tiles (A)-(G). It is immediate that tile (1) can only appear above tile (2), and that tile (2) can only appear to the right to tile (1). Thus, tiles (1) and (2) necessarily form a diagonal. We superimpose tile (1) on tiles (E) and (G); tile (2) on tiles (A), (D) and (F); and tile (3) on tiles (A) though (F). This forces the core-regions to be square, because the diagonal starting with the (G) on the bottom left corner must hit a (G) tile at its other end. In all, we obtain that any tiling of an odd-sized torus using this set of tiles, which we call the *base tiles*, can be decomposed into square core-regions, all of identical sizes.

Machine Simulation Tiling. The original reduction's "machine simulation" tiles are superimposed onto the "base tiles" as follows:

- Onto tile (G) we superimpose the tile representing the beginning of the first configuration (representing the machine's head, its start-state and a blank tape).
- Onto tiles (A) and (B) we superimpose the tiles representing the rest of the first configuration (a blank tape).

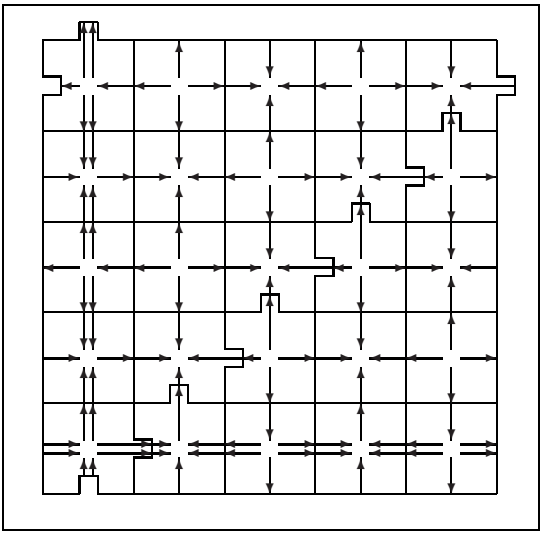

Fig. 2. The base tiling of a 5×5 region

- The rest of the machine simulation tiles are superimposed on all of the remaining base tiles.
- Tiles (A), (B) and (G) are configured such that only they can appear above a tile representing an accepting state.

Using this set of superimposed tiles, every tiling of an $i \times i$ core-region must correspond to a computation of M accepting in exactly i steps.

Guaranteeing a Single Core-Region. We now want to guarantee that the tiling of the torus consists of a *single* core-region, providing that it corresponds to a single computation of the full length (rather than copies of shorter computations). Note that the tiling of a prime-sized torus necessarily consists of a single core-region - covering the entire torus. This is because core-regions must be of the same size, and hence this size must divide the size of the torus. For technical reasons we cannot use prime numbers, but use their third-power instead. Given the Turing machine M, we construct a new machine M' such that for every t, M' accepts in exactly ${P_t}^3$ steps if and only if M accepts in exactly t steps (where P_t is the t-th prime number). We do so using nondeterminism, by which M' first "guesses" the number of steps t, and simulates M to see if it accepts in exactly t steps. In parallel, M' computes P_t^3 on a second tape, and counts the number of steps on a third tape. If the simulation of M on the first tape accepts in exactly t steps, then M' waits until exactly ${P_t}^3$ steps have elapsed and then accepts. We now build the tile set $T(M')$ corresponding to M'. If M has a computation accepting in t steps, then $T(M')$ has a tiling of the ${P_t}^3 \times {P_t}^3$ torus. Conversably, suppose that $T(M')$ has a tiling of the ${P_t}^3 \times {P_t}^3$ torus. Then, this tiling is either composed of a single core-region, or core-regions of size P_t or ${P_t}^2$. However, core-regions of sizes P_t or ${P_t}^2$ would correspond to computations of M' accepting in P_t or ${P_t}^2$ steps, respectively. But M' only accepts in a number of steps which is a cube of

a prime, which neither P_t nor ${P_t}^2$ are. Hence, the tiling of the ${P_t}^3 \times {P_t}^3$ torus necessarily consists of a single core-region, corresponding to a t-step accepting computation of M. □

Lemma 2. *Fixed-structure-Torus-Tiling* $\leq^{fpt}$ *Fixed-structure-Independent-Set.*

Proof. Given a tile set T and a number s, we construct a parametric graph expression G, a vector $\boldsymbol{t}$ of parameter values, and integer ψ, such that $G(\boldsymbol{t})$ has an independent set of size ψ iff the $s \times s$ torus can be tiled using the tiles in T. Here we provide the proof for directed graphs. The proof for undirected graphs is considerably more complex and is provided in the full version.

The basic idea is to create a graph $G = G(\boldsymbol{t})$ that "represents" the torus. For each of the s^2 locations of the torus we create a "super-node" that is a $|T|$-clique. Each vertex in the super-node represents a different tile of T. The s^2 super-nodes are organized in s rows and s columns (as in the torus). Note that in each super-node at most one vertex can be chosen for the independent set. This chosen vertex will represent the tile chosen for this location in the torus tiling. Edges are placed between vertices in adjacent super-nodes (vertical and horizontal), to correspond to the adjacency constraints of the tiling. Specifically, let H_T be the bipartite graph with $|T|$ vertices at each side, such that there is an edge $i \rightarrow j$ iff tile t_i *cannot* be placed to the left of tile t_j. Similarly, let V_T be the bipartite graph with $|T|$ vertices in each part (this time viewed as one part above the other), such that there is an edge $i \rightarrow j$ iff tile t_j cannot be placed on top of tile t_i. Each super-node is connected with its right-neighbor with H_T and with its neighbor on top by V_T. With this construction, G has an independent set of size s^2 iff the $s \times s$ torus can be tiled by T. We now show how to construct the graph expression for G.

The Directed s-Cycle. The basic building block of our torus-graph is the directed s-cycle. For $s = 3 \bmod 4$ the directed s-cycle (denoted $C(s)$) is created using the following expression:

$$C(s) = \left(\left[\begin{smallmatrix}\bullet\\ \uparrow\\ \bullet\end{smallmatrix}\right] \cup \left(\left(\left\lfloor \frac{s}{4} \right\rfloor - 1\right) \cdot \left[\begin{smallmatrix}\bullet\rightarrow\bullet\\ \bullet\rightarrow\bullet\end{smallmatrix}\right]\right) \cup \left[\begin{smallmatrix}\bullet\rightarrow\bullet\\ \bullet\rightarrow\bullet\end{smallmatrix}\right]\right) + \left(\left(\left\lfloor \frac{s}{4} \right\rfloor \cdot \left[\begin{smallmatrix}\bullet\rightarrow\bullet\\ \bullet\rightarrow\bullet\end{smallmatrix}\right]\right) \cup \left(\begin{smallmatrix}\bullet & \bullet\\ \bullet & \end{smallmatrix}\right)\right)$$

For $s = 0, 1$, and $2 \bmod 4$ the construction is similar (placing less nodes at the right-end of the expression).

Using the s-cycle, we construct two graphs, the sum of which is the $s \times s$ torus. The graphs, denoted Tr^H and Tr^V, consist of the horizontal and vertical edges of the torus, respectively. Let e_1 be the graph with a single vertex with a self loop. Then, Tr^H and Tr^V are obtained by multiplying $C(s)$ by s self-loops from the left and from the right, respectively:

$$\mathrm{Tr}^H(s) = (s \cdot e_1) \times C(s) \quad , \quad \mathrm{Tr}^V(s) = C(s) \times (s \cdot e_1)$$

Next, we "blow-up" the graphs Tr^H and Tr^V, substituting each vertex with a "super-node" consisting of $|T|$ vertices, and connecting the "super-nodes" by H_T and V_T, respectively:

$$G^H(s) = \mathrm{Tr}^H(s) \times H_T \quad , \quad G^V(s) = \mathrm{Tr}^V(s) \times V_T$$

(Note that this is where the directness of the graph comes to play, allowing to keep the directions of H_T and V_T.) Together, these graphs have all the vertices and most of the edges, except for the clique edges within each super-node. These are obtained by adding s^2 copies of the fixed clique $K_{|T|}$. The complete graph expression is:

$$G(s) = G^H(s) + G^V(s) + (s^2 \cdot K_{|T|})$$

This concludes the construction of the expression G. The parameter for this graph is s. By construction, G has an independent set of size $\psi = s^2$ iff the $s \times s$ torus can be tiled with T. □

Lemma 3. *Fixed-Structure-Independent-Set* $\leq^{XP}$ *FS-Exact-Halt.*

Proof. Given a graph expression G, parameter vector $\boldsymbol{t}$, and integer ψ, we construct a Turing machine M_G and integer r, such that M_G accepts the empty string in exactly r steps iff $G(\boldsymbol{t})$ has an independent set of size ψ. Assume that $\boldsymbol{t}$ has m entries, and denote $t = \sum_{i=1}^{m} t_i$. First note that it is possible to construct $G(\boldsymbol{t})$ in at most $(|G|+t)^{3|G|}$ steps. This is true since there are at most $|G|$ operations, and the result graph of the i-th operation has at most $(|G|+t)^i$ vertices and $(|G|+t)^{2i}$ edges. Once the graph $G(\boldsymbol{t})$ is constructed, one can guess a subset of the vertices, and check if they are an independent set of size ψ. The only problem is that the machine M_G operates on the empty input. Thus, we cannot explicitly provide it with the parameters $\boldsymbol{t}$ and ψ. Rather, we let the machine "guess" these values, and encode them into the number of steps. Specifically, let $code(\psi, \boldsymbol{t}) = \left(P_{|G|} \cdot P_{\psi}^2 \cdot \prod_{i=1}^{m} P_{t_i}^{i+2}\right)^{3|G|}$, where P_j is the j-th prime number. Note that $code(\cdot,\cdot)$ is a bijection. Accordingly, given G we construct the Turing machine M_G to operate as follows:

1. Nondeterministically "guess" a vector $\boldsymbol{t'} = (t'_1, \ldots, t'_m)$ and integer ψ'.
2. Create the graph $G = G(\boldsymbol{t'})$.
3. Nondeterministically "guess" a subset of the vertices of G and check if they are an independent set of size ψ'. If not, reject.
4. In parallel to the above, compute $code(\psi', \boldsymbol{t'})$. Run for a total of $code(\psi', \boldsymbol{t'})$ steps and accept.

It can be verified that $code(\psi', \boldsymbol{t'})$ steps suffice for steps (1)-(3). We obtain that M_G accepts in exactly $r = code(\psi, \boldsymbol{t})$ steps iff $G(\boldsymbol{t})$ has an independent set of size ψ. Note that $code(\psi, \boldsymbol{t}) \leq (|G| + \psi + \sum_{i=1}^{m} t_i)^{4|G|}$, providing that the reduction is an XP one. □

The equivalence of *Fixed-Structure-Clique* and *Fixed-Structure-Vertex-Cover* follows from the standard reductions between Independent-Set, Clique and vertex-Cover.

5 Open Problems

This work takes the first steps in understanding fixed-structure problems. Many important and interesting problems remain open. Here we list just a few:

- The results presented in this paper are hardness results. We were also able to show that some other fixed-structure problems are FPT. These results are omitted due to lack of space. However, we believe we are still lacking in tools for the design of FPT algorithms for fixed structure problems.
- We identified three core fixed-parameter problems, which we believe define three separate complexity classes. Are these "the right" complexity classes? Are there other important/interesting classes? Is there a hierarchy? Are *FS-Halt* and *FS-Not-Halt* indeed non-equivalent? Are they in FPT? What other problems are equivalent to these problems?
- We showed an XP equivalence between *Fixed-Structure-Independent-Set (FS-IS)* and *FS-Exact-Halt*. With our definition of FS-IS this is all but unavoidable, since the size of the graph may be exponential in its representation, and hence FS-IS need not be in NP. If we add the size of the graph (in unary) to the input, FS-IS becomes NP. Is this problem equivalent to *FS-Exact-Halt* under FPT reductions?
- In this paper, we only covered few fixed structure problems. A whole line of research is to analyze the complexity of the fixed-structure versions of the numerous problems for which the classical parametrization has been studied.
- The notion of graph products provides the basis for many interesting fixed-structure graph problems. For example, what is the fixed structure complexity of Independent-Set on $G \times K_t$ graphs? Similarly, for other graph problems, and other graph structures (e.g. tori, trees, butterflies, etc., instead of K_t). In addition, one may consider other types of graph products, i.e. cartesian, lexicographic and strong products (see [12]).
- We proved that fixed-structure tiling of the plane is equivalent to *FS-Not-Halt* for the version of the problem in which the origin tile is specified (the proof is not provided here). What is the complexity of the general problem when the origin tile is not specified?
- We already noted that problems with two combinatorial-structures, such as the database query problems and graph ordering problems, though different, are somewhat related to fixed-structure problems. Some of these problems are still open. It would be interesting to see if the directions developed here can shed some light on these problems.

Acknowledgements. We are grateful to Mike Fellows for helpful comments on an early version of this work.

References

1. Cesati, M., Di Ianni, M.: Computational models for parameterized complexity. Mathematical Logic Quarterly 43, 179–202 (1997)
2. Cesati, M.: Compendium of parameterized problems (2006), `http://bravo.ce.uniroma2.it/home/cesati/research/compendium/compendium.pdf`
3. Wareham, T.: The parameterized complexity of intersection and composition operations on sets of finite-state automata. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 302–310. Springer, Heidelberg (2001)
4. Fernau, H., Hagerup, T., Nishimura, N., Ragde, P., Reinhardt, K.: On the parameterized complexity of a generalized rush hour puzzle. In: Proceedings of Canadian Conference on Computational Geometry, CCCG, pp. 6–9 (2003)

5. Papadimitriou, C.H., Yannakakis, M.: On the complexity of database queries. In: Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 12–14 (1997)
6. Downey, R.G., Fellows, M.R., Taylor, U.: On the parameteric complexity of relational database queries and a sharper characterization of w[1]. In: Combinatorics, Complexity and Logic, Proceedings of DMTCS 1996 (1996)
7. Vardi, M.: The complexity of relational query languages. In: Proceedings of the 14th ACM Symposium on Theory of Computing, pp. 137–146 (1982)
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
9. Grohe, M., Schwentick, T., Segoufin, L.: When is the evaluation of conjunctive queries tractable? In: Proceedings of 33rd annual ACM Symposium on Theory of Computing, pp. 657–666 (2001)
10. Lewis, H.R., Papadimitriou, C.H.: Elements of the Theory of Computation. Prentice Hall, Englewood Cliffs (1981)
11. Berger, R.: The undecidability of the domino problem. Mem. AMS 66 (1966)
12. Imrich, W., Klavzer, S.: Product Graphs: Structure and Recognition. Wiley, Chichester (2000)