Modular-Width: An Auxiliary Parameter for Parameterized Parallel Complexity

Faisal N. Abu-Khzam[1,4], Shouwei Li[2(✉)], Christine Markarian[3], Friedhelm Meyer auf der Heide[2], and Pavel Podlipyan[2]

[1] Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon
[2] Heinz Nixdorf Institute and Department of Computer Science, Paderborn University, Paderborn, Germany
sli@mail.uni-paderborn.de
[3] Department of Mathematical Sciences, Haigazian University, Beirut, Lebanon
[4] School of Engineering and Information Technology, Charles Darwin University, Darwin, Australia

Abstract. Many graph problems such as maximum cut, chromatic number, hamiltonian cycle, and edge dominating set are known to be *fixed-parameter tractable* (FPT) when parameterized by the treewidth of the input graphs, but become W-hard with respect to the clique-width parameter. Recently, Gajarský *et al.* proposed a new parameter called modular-width using the notion of modular decomposition of graphs. They showed that the chromatic number problem and the partitioning into paths problem, and hence hamiltonian path and hamiltonian cycle, are FPT when parameterized by this parameter. In this paper, we study modular-width in parameterized parallel complexity and show that the weighted maximum clique problem and the maximum matching problem are *fixed-parameter parallel-tractable* (FPPT) when parameterized by this parameter.

1 Introduction

Parameterized complexity has become a mainstream framework of theoretical computer science in the last two decades. The central idea of this theory is to study the complexity of NP or even PSPACE-hard problems with respect to one (or more) parameter(s) rather than restricting the analysis to the input size. This has led to the development of the class of *fixed-parameter tractable* (FPT) problems. In short, a parameterized problem is FPT if it has a deterministic algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$, where n is the input size, k is the parameter, and f is an arbitrary computable function.

To further reduce the running time of fixed-parameter algorithms, parallel computing is employed, and this is broadly known as *parameterized parallel complexity*. To the best of our knowledge, there are three complexity classes

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901) and the International Graduate School "Dynamic Intelligent Systems".

M. Xiao and F. Rosamond (Eds.): FAW 2017, LNCS 10336, pp. 139–150, 2017.
DOI: 10.1007/978-3-319-59605-1_13

defined in the literature with the aim of capturing parametrized analogues of the class NC, known as PNC (parameterized analog of NC), introduced in [2], FPP (fixed-parameter parallelizable), introduced in [6], and FPPT (fixed-parameter parallel-tractable), introduced in [1], respectively. Let $f(k)$, $g(k)$, and $h(k)$ be arbitrary functions of the parameter k, and α, β be positive constants. (1) PNC is the class of all parametrized problems solvable in time $f(k) \cdot (\log n)^{h(k)}$ using $g(k) \cdot n^{\beta}$ parallel processors. (2) FPP is the class of all parametrized problems solvable in time $f(k) \cdot (\log n)^{\alpha}$ using $g(k) \cdot n^{\beta}$ parallel processors. (3) FPPT is the class of all parametrized problems solvable in time $f(k) \cdot (\log n)^{\alpha}$ using n^{β} parallel processors. It is easy to observe that FPPT $\subseteq$ FPP $\subset$ PNC according to the definition.

We also note here that FPP $\subseteq$ FPPT, because a parallel algorithm with runtime $f(k) \cdot (\log n)^{\alpha}$ using $g(k) \cdot n^{\beta}$ parallel processors can be simulated by one with running time $g(k)f(k) \cdot (\log n)^{\alpha}$ using n^{β} parallel processors. The definition of FPPT is, therefore, a simplified version of FPP, and it was introduced in [1] to emphasize the need to have a polynomial number of processors. In fact, the parameter k may be treated like another input variable (rather than a constant), so a number of processors that varies as an arbitrary (super-polynomial) function of k is not desired. Yet, from a theoretical standpoint, an FPP-algorithm may give more information about the parametrized complexity of a problem than FPPT, if we are interested in the maximum possible size of k (as a function $k(n)$ of n) so that the problem is in NC as long as $k \leq k(n)$. For example, an algorithm with running time $\mathcal{O}(k \cdot \log^{\alpha} n)$ using $\mathcal{O}\left(2^k \cdot n^{\beta}\right)$ parallel processors is an NC-algorithm for $k = \mathcal{O}(\log n)$. Expressing its performance in the FPPT model $\mathcal{O}\left(k \cdot 2^k \cdot \log^{\alpha} n\right)$ using $\mathcal{O}(n^{\beta})$ processors would only yield the bound $k = \mathcal{O}(\log \log n)$. The results in this paper can be interpreted in either of these models.

One of the most ubiquitous parameters studied in both sequential and parallel classes is *treewidth*, which roughly measures how tree-like a graph is. Algorithms for problems on graphs of bounded treewidth are much more efficient than their counterparts in general graphs (e.g. see [3,4]). A celebrated metatheorem of Courcelle and Di Ianni [8] states that any problem expressible in monadic second-order logic is FPT when parameterized by the treewidth of the input graphs. Similarly, Cesati *et al.* in [6] showed that all problems involving *MS* (definable in monadic second-order logic with quantifications over vertex and edge sets) or *EMS properties* (that involve counting or summing evaluations over sets definable in monadic second-order logic) are FPP when restricted to graphs of bounded treewidth. Moreover, Lagergren [17] presented an efficient parallel algorithm for the tree decomposition problem for fixed treewidth. Hagerup *et al.* [16] showed that the maximum network flow problem is in NC when parameterized by treewidth. Despite the fact that these results are noteworthy, one of the main drawbacks of treewidth is that a large number of interesting instances are excluded since graphs of small treewidth are necessarily sparse. The notion of clique-width (as well as rank-width [20], boolean-width [5] and shrub-depth [14]), which is stronger than treewidth, tries to address this problem by covering a

larger family of graphs, including many dense graphs. However, the price for this generality is exorbitant. Several natural problems such as maximum cut, chromatic number, hamiltonian cycle, and edge dominating set, which were shown to be FPT for bounded treewidth, become W[1]-hard when parameterized by these measures [10–12].

Consequently, a parameter called *modular-width*, that covers a significantly larger class of graphs has been introduced by Gajarský *et al.* in [13] where it was shown that several problems, such as the chromatic number and the partitioning into paths, and hence hamiltonian path and hamiltonian cycle, which are W[1]-hard for both clique-width and shrub-depth, are FPT when parameterized by modular-width.

In this paper, we extend the study of modular-width to parameterized parallel complexity and show that the weighted maximum clique problem and the maximum matching problem are FPPT for bounded modular-width. Our algorithms are based on the following ideas/techniques: we use an algebraic expression to represent the input graph. Then, we construct the modular decomposition tree of the input graph and a computation tree that corresponds to the maximal strong modules in the maximal decomposition tree. For each node of the computation tree, we compute an optimal solution by giving the optimal solutions for the children of this node in the modular decomposition tree. The optimal solution for each node can be obtained by integer linear programming. Then, we use a bottom-up dynamic programming approach along with the modular decomposition tree to obtain the global solution. In order to explore and evaluate the tree efficiently, we use a parallel tree contraction technique due to Miller and Reif [18].

2 Preliminaries

All graphs considered in this paper are simple, undirected and loopless. We use the classical graph theoretic notations and definitions (e.g. see [15]). The neighborhood of a vertex x in a graph $G = (V, E)$ is denoted by $N(x)$. Given a subset of vertices $X \subseteq V$, $G[X]$ denotes the subgraph induced by X.

Let M be a subset of vertices of a graph G, and x be a vertex of $V \setminus M$. We say that vertex x *splits* M (or is a *splitter* of M) if x has both a neighbor and a non-neighbor in M. If x does not split M, then M is *homogeneous* with respect to x.

Definition 1. Given a graph $G = (V, E)$, $M \subseteq V$ is called a *module* if M is homogeneous with respect to any vertex $x \in V \setminus M$ (i.e. $M \subseteq N(x)$ or $M \cap N(x) = \emptyset$).

Let M and M' be disjoint sets. We say that M and M' are adjacent if any vertex of M is adjacent to all the vertices in M' and non-adjacent if the vertices of M are non-adjacent to the vertices of M'. Thus, it is not hard to observe that two disjoint modules are either adjacent or non-adjacent.

A module M is *maximal* with respect to a set S of vertices if $M \subset S$ and there is no module M' such that $M \subset M' \subset S$. If the set S is not specified, we assume that $S = V$. A module M is a *strong module* if it does not overlap with any other module. Note that, one-vertex subsets and the empty set are modules and are known as the *trivial* modules. A graph is called *prime* if all of its modules are trivial.

Definition 2. Let $P = \{M_1, \ldots, M_k\}$ be a partition of the vertex set of a graph $G = (V, E)$. If for all i, $1 \leq i \leq k$, M_i is a non-trivial module of G, then P is a *modular partition* of G.

A non-trivial modular partition $P = \{M_1, \ldots, M_k\}$ which only contains maximal strong modules is a *maximal modular partition*. Note that each undirected graph has a unique maximal modular partition. If G (resp. $\overline{G}$) is not connected then its connected (resp. co-connected) components are the elements of the maximal modular partition.

Definition 3. For a modular partition $P = \{M_1, \ldots, M_k\}$ of a graph $G = (V, E)$, we associate a *quotient graph* $G_{/p}$, whose vertices are in one-to-one correspondence with the parts of P. Two vertices v_i and v_j of $G_{/p}$ are adjacent if and only if the corresponding modules M_i and M_j are adjacent in G.

The inclusion tree of the strong modules of G, called the *modular decomposition tree*, entirely represents the graph if the representative graph of each strong module is attached to each of its nodes (see Fig. 1). It is easy to observe that there are only three relations, $M \subseteq M'$, $M' \subseteq M$, or $M \cap M' = \emptyset$, for any

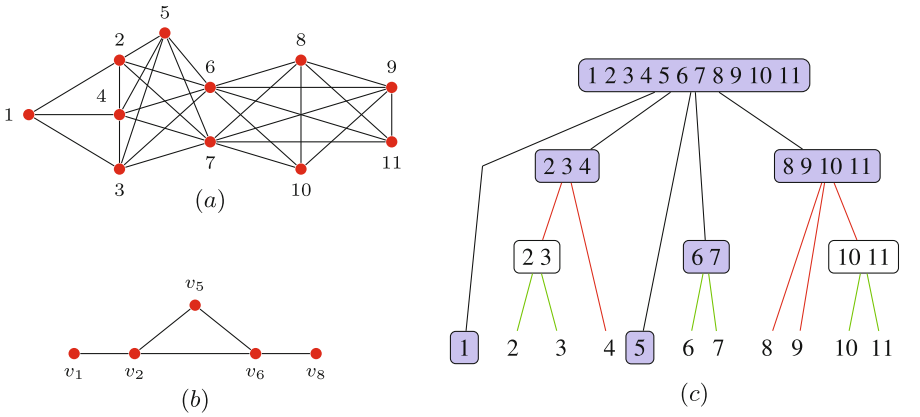

Fig. 1. (a) Shows the graph G; $\{\{1\}, \{2, 3\}, \{4\}, \{5\}, \{6, 7\}, \{9\}, \{8, 10, 11\}\}$ is a modular partition of G. The maximal modular partition of G is $P = \{\{1\}, \{2, 3, 4\}, \{5\}, \{6, 7\}, \{8, 9, 10, 11\}\}$ and (b) represents its quotient graph. (c) is the modular decomposition tree of G. The maximal strong modules are in blue. The green edges indicate that the root node is parallel, the red edges indicate that the root is series, and the black edges indicate that the root is a prime graph. (Color figure online)

two nodes M and M' in the modular decomposition tree. The *modular-width* is the maximum degree of the modular decomposition tree. An excellent feature of modular decomposition is that it can be computed in $\mathcal{O}(\log^2 n)$ time with $\mathcal{O}(n+m)$ parallel processors [9], thus the modular-width stays also within the same resource bounds.

Theorem 1 (Modular decomposition theorem [7]). *For any graph $G = (V, E)$, one of the following three conditions is satisfied:*

1. *G is not connected;*
2. *$\overline{G}$ is not connected;*
3. *G and $\overline{G}$ are connected and the quotient graph $G_{/P}$, where P is the maximal modular partition of G, is a prime graph.*

Theorem 1 indicates that, the quotient graphs associated with the nodes of the modular decomposition tree of the strong modules are of three types: an *independent set* if G is not connected (the node is labeled *parallel*); a clique if $\overline{G}$ is not connected (the node is labeled *series*); a prime graph otherwise.

Parallel tree contraction is a "bottom-up" technique for constructing parallel algorithms on trees. There are two basic operations called *rake* and *compress*. During each contraction, processors are assigned to leaves of the tree and perform local modifications by removing these leaves, hence creating new leaves that are processed at the next round. This operation is called rake. Clearly, removing leaves is not sufficient for a tree that is thin and tall, like a linked list, which would take a linear number of rounds to reduce the tree to a point. Thus, a complementary operation called compress that reduces a chain of vertices, each with a single child to a chain of half the length is introduced. Ideally, rake and compress work on different parts of the tree simultaneously. During the run of the algorithm, the rake operation tends to produce chains that are then reduced by the compress operation. Thus, the whole tree can be evaluated in $\mathcal{O}(\log n)$ time using $\mathcal{O}(n)$ parallel processors.

3 Parallel Algorithms on Modular Decomposition

In this section, we show how modular-width can be used to derive efficient parallel algorithms for the weighted maximum clique problem and the maximum matching problem. Our results imply that these two problems are FPPT.

The input to our algorithms is assumed to be a graph of modular-width at most k, and we shall represent the input graph as an algebraic expression consisting of the following operations:

1. G has only one vertex. This corresponds to a leaf node in the modular decomposition tree.
2. G is a *disjoint union* of two graphs G_1 and G_2 of modular-width at most k. The disjoint union of G_1 and G_2 defined as a graph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2$. This corresponds to a parallel node in the modular decomposition tree.

3. G is a *complete join* of two graphs G_1 and G_2 of modular-width at most k. The complete join of G_1 and G_2 is defined as a graph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2 \cup \{\{u, v\} : u \in V_1 \text{ and } v \in V_2\}$. This corresponds to a series node in the modular decomposition tree.
4. The *substitution* operation with respect to a graph G is the reverse of the quotient operation and defined as replacing a vertex of G by $G_i = (V_i, E_i)$ of modular-width at most k while preserving the neighborhood,

$$G_{x \to G_i} = (V \setminus \{x\} \cup V_i,$$
$$(E \setminus \{(x, y) \in E\} \cup E_i \cup \{(y, z) : (x, y) \in E, z \in V_i\})).$$

This is corresponding to the maximal modular partition of G, and G_i is one of the maximal strong module of G.

Throughout the rest of this paper, we may assume that a graph $G = (V, E)$ and the modular decomposition of G, of modular-width at most k, are already given. Otherwise, we can apply the algorithm presented in [9] to obtain one. Under this assumption, it is easy to note that the modular decomposition tree of G can be constructed in constant time using $\mathcal{O}(n)$ parallel processors. Moreover, the number of maximal strong modules in the decomposition tree of G is at most n, which is equal to the cardinality of the maximal modular partition of G.

The central idea of our algorithm is a bottom-up dynamic programming approach along the modular decomposition tree of the algebraic expression as defined above. For each node of the modular decomposition tree, we compute a record for the graph represented by the subtree of the modular decomposition below that node. That is, given the optimal solutions for the children of each node in the modular decomposition tree, we can compute an optimal solution for the node itself. In order to explore the decomposition tree efficiently, we take the parallel tree contraction technique due to Miller and Reif [18].

3.1 The Weighted Maximum Clique Problem

Let us consider the weighted maximum clique problem which is known to be NP-complete for general graphs. Given a graph $G = (V, E)$ and weights on each vertex, is there a clique with maximum weight ω?

Theorem 2. *The weighted maximum clique problem parameterized by the modular-width k can be solved in $\mathcal{O}(2^k \cdot \log n)$ time using $\mathcal{O}(n)$ parallel processors. Thus it is FPPT.*

Proof. Clearly, graph G with bounded modular-width k can be represented by the four operations mentioned above according to Theorem 1. We only need to show that each operation can be done efficiently, and the whole decomposition tree can be evaluated by the parallel tree contraction technique.

First, each leaf node in the modular decomposition tree is an isolated vertex, which can be represented by the first operation of the algebraic expression. Thus,

the maximum clique weight of each vertex is trivially its own weight. Obviously, this can be done in constant time with a linear number of parallel processors.

Next, we consider other operations on combining two modules to form a larger module:

- If G is the disjoint union of G_1 and G_2, then the maximum clique weight of G would be:
$$\omega(G) = \max\{\omega(G_1), \omega(G_2)\},$$
since the disjoint union operation corresponds to a parallel node, which implies two modules are non-adjacent.
- If G is the complete join of G_1 and G_2, then the maximum clique weight of G would be:
$$\omega(G) = \omega(G_1) + \omega(G_2),$$
since complete join corresponding to a series node, which implies any vertex in G_1 is adjacent to all the vertices in G_2.

The last case is for G is a substitution of G_i for $1 \leq i \leq k$, which means G is neither obtained by disjoint union operation nor complete join operation. In other words, the quotient graph of G is prime, and the vertices are in one-to-one correspondence with G_i for $1 \leq i \leq k$. In this scenario, graph G can be treated as a graph with at most k vertices, and the weight of each vertex is equal to the maximum clique weight of the corresponding module G_i for $1 \leq i \leq k$. Since each G_i looks like a black box to the other G_j in the maximal modular decomposition of G for $1 \leq i, j \leq k$ and $i \neq j$, and there has no efficient algorithm for the maximum weighted clique problem, we have no choice but take a brute-force strategy to evaluate G if the maximum clique weight of each G_i for $1 \leq i \leq k$ are given, and this can be done in $\mathcal{O}(2^k)$ time.

Now we show how to parallelize the algorithm by the parallel tree contraction technique. We construct a computation tree corresponding to the modular decomposition tree of G, such that each tree node corresponds to a maximal strong module of size at most k and has at most k children. Suppose v is an internal node in the computation tree, we call v is *half-evaluated* when all but one of its children has been evaluated. With the parallel tree contraction technique, it can be compressed later. Suppose the unevaluated child is v_1 and its maximum clique weight is ω', the maximum clique weight of v without v_1 evaluated to a, and the maximum clique weight among evaluated children $v_2, \ldots, v_k$ of v is b, a and b are known values. Then the maximum clique weight of v is

$$\omega = \max\{a, \omega' + b\}.$$

During each contraction progress, we can take the above function recursively and have

$$\omega'' = \max\{c, \omega + d\},$$

where c and d are two known values for next round, then

$$\omega'' = \max\{\max\{b + c, d\}, \omega' + (a + c)\}.$$

Thus, the running time is $\mathcal{O}(2^k \cdot \log n)$ using $\mathcal{O}(n)$ parallel processors, because $\mathcal{O}(2^k)$ time is required to compute a maximum weight clique for a prime graph with at most k vertices, the parallel tree contraction takes $\mathcal{O}(\log n)$ time using a linear number of parallel processors, and half-evaluating a node requires $\mathcal{O}(\log k)$ time. □

3.2 The Maximum Matching Problem

A matching in a graph is a set of edges such that no two edges share a common vertex. We now consider the maximum matching problem which seeks a matching of maximum size (i.e., the largest number of edges). The existence of an NC algorithm for this problem has been open for several decades, even if the graph is planar. By considering the modular-width as the parameter, we prove the following:

Theorem 3. *The maximum matching problem parameterized by the modular-width k can be solved in $\mathcal{O}(2^k \cdot \log n)$ time using $\mathcal{O}(n)$ parallel processors. Therefore the problem is FPPT.*

Proof. We follow the same strategy as Theorem 2 and evaluate different operations on combining modules. Let n_1, n_2 denote the number of vertices and u_1, u_2 denote the number of unmatched vertices of graphs G_1 and G_2. We use the pair $<n_i, u_i>$ to track the maximum matching of graph G_i.

First, each leaf node i in the modular decomposition tree is an isolated vertex, thus

$$n_i = 1 \text{ and } u_i = 1.$$

Next, we consider various operations on combining two modules to form a larger module:

- If G is the disjoint union of G_1 and G_2, the values of G would be:

$$n = n_1 + n_2 \text{ and } u = u_1 + u_2,$$

  since disjoint union corresponds to a parallel node, which implies there is no edge between G_1 and G_2.
- If G is the complete join of G_1 and G_2, the values of G would depend on the values of n_1, n_2, u_1 and u_2. We have to consider different cases for this scenario. In fact, no matter for which cases,

$$n = n_1 + n_2$$

  always valid, we only need to consider u.
  1. If $u_1 > n_2$, then the unmatched vertices in G_1 are more than the vertices of G_2, then the values of G would be:

$$u = u_1 - n_2,$$

     because we could match all vertices of G_2 through the edges between unmatched vertices in G_1 and all vertices of G_2.

2. Symmetrically, if $u_2 > n_1$, the values of G would be:

$$u = u_2 - n_1.$$

3. The last case comes to $u_1 < n_2$ and $u_2 < n_1$. In this circumstances, we are able to match almost all vertices in G_1 and G_2, and only have one unmatched vertex left over if there is an odd number of vertices in G, the values of G would be

$$u = n_1 + n_2 \pmod 2.$$

Without loss of generality, suppose $u_1 - u_2 \geq 2$, we can further match the u_2 unmatched vertices in G_2 by the additional edges of complete join operation. After that, all vertices in G_2 are matched, and only $u_1 - u_2 \geq 2$ vertices left in G_1 still unmatched. Suppose x, y are two unmatched vertices in G_1; we know that x, y are also adjacent to all vertices in G_2 because of the complete join operation, and all vertices of G_2 are matched. Then there must be at least one edge (a, b) in the matching of G_2, such that $x-a-b-y$ is an augmenting path and the matching can be extended by 1. Thus the number of unmatched vertices in G only depends on the parity of $u_1 + u_2$, which is equal to the parity of $n_1 + n_2$.

Thus for the complete join of G_1 and G_2, we have

$$n = n_1 + n_2,$$

and

$$u = \max\{u_1 - n_2, u_2 - n_1, n_1 + n_2 \pmod 2\}.$$

Obviously, this can be done in a constant time given the values of G_1 and G_2.

– Finally, we consider the case where G is a prime graph, which is obtained by the substitution operation on modules $G_1, \ldots, G_k$. As claimed in the proof of Theorem 2, G can be treated as a prime graph with at most k vertices, and each vertex corresponding to a module G_i for $1 \leq i \leq k$ in this case.

It is well-known that the maximum matching problem can be formulated as integer linear programming. Once again, let u_i denote the number of unmatched vertices in G_i, E denotes the edge set among $G_1, \ldots, G_k$, and $e_{i,j}$ denote the number of matched edges between G_i and G_j for $1 \leq i, j \leq k$. Then finding a maximum matching in G is equivalent to solving the following problem:

$$\begin{aligned}
&\text{Maximize } \left\{ \sum_{(i,j)\in E} e_{i,j} + \sum_{i} e_{i,i} \right\} \text{ subject to} \\
&2e_{i,i} + \sum_{(i,j)\in E} e_{i,j} \leq n_i \text{ for } i = 1, \ldots, k \\
&e_{i,i} \leq \frac{(n_i - u_i)}{2} \text{ for } i = 1, \ldots, k \\
&e_{i,j} \in [1, k] \text{ for } 1 \leq i, j \leq k.
\end{aligned}$$

For each prime graph, we can compute the matching by taking the maximum of our objective function at every feasible solution. This can be done in $\mathcal{O}(2^k)$ time since the graph has a bounded modular-width k.

Now, we show how to parallelize the algorithm using the parallel tree contraction technique.

Let $n_1, \ldots, n_k$ denote the number of vertices and $u_1, \ldots, u_k$ denote the number of unmatched vertices in $G_1, \ldots, G_k$. Suppose $n_1, \ldots, n_k$ and $u_2, \ldots, u_k$ are known, but u_1 is not. Then the number of unmatched vertices of the graph G can be represented as a function u of u_1 of the form $\max\{p, u_1 - q\}$ for a proper choice of constants p and q, such that $p = u(n_1 \pmod 2)$ and $q = n_1 - u(n_1)$.

As argued in the complete join operation of two graphs, u_1 must have the same parity as n_1. For any x of the same parity as n_1 between 2 and n_1, it is clear that $u(x-2) \leq u(x) \leq u(x-2)+2$. We will show that $u(u_1)$ is a piecewise linear function, consisting of a constant portion for low values of u_1 followed by a portion with slope 1 for high values of u_1 in Lemma 1. Thus $u(u_1)$ has the form $\max\{p, u_1 - q\}$. We choose $p = u(n_1 \pmod 2)$ so the formula is correct at the low end. For the high end, we choose $q = n_1 - u(n_1)$.

We now use the parallel tree contraction technique. Composing functions of the form $u(x) = \max\{p, x - q\}$ leaves another function of the same form which can be computed in constant time. Therefore, the tree contraction can be done in $\mathcal{O}(\log n)$ time with $\mathcal{O}(n)$ parallel processors. □

Lemma 1. *If x has the same parity as n_1 and also $4 \leq x \leq n_1$, then it cannot satisfy: $u(x) = u(x-2) = u(x-4) + 2$.*

Proof. Let M' be the matching used to calculate $u(x-4)$ and M be the matching used to calculate $u(x)$. Then we have

$$n_1 = 2 * |M'| + (x-4) = 2 * |M| + x;$$

thus,

$$|M'| = |M| + 2;$$

It follows that M' contains at least two edges between vertices in G_1.

Let M'' be the matching M' without an edge e between two of the vertices in G_1, then M'' will be a matching used to calculate $u(x-2)$. If we choose $u_1 = x - 2$, both matchings M and M'' have the same cardinality and both are also maximum. Let G' be the resultant graph from taking the symmetric difference of M and M''; i.e. $(M - M'') \cup (M'' - M)$. Every connected component of G' must be either an even cycle whose edges alternate between M and M'' or an even length path whose edges alternate between M and M'' with distinct endpoints. Now add the edge e that is in $M' \setminus M''$, its connected component must be an odd path. If there is another matched edge e' between two of the vertices in G_1, and e' is not in the same connected component, then we can take the edges from M in the component of e', add them to the rest of M', and have a larger matching for the case $u_1 = x - 2$. Alternatively, if there does not exist a matching edge between vertices in G_1, that is also in a different component

from e, we can still modify the matching to obtain a larger one when $u_1 = x - 2$. Let e' be another edge from M' that is contained in G_1. Any vertex outside of G_1 that is adjacent to the vertices of e is also adjacent to the vertices of e'. We can add an edge between one of these vertices and a vertex of e' so that an even cycle is created. We use this even cycle to take a different set of edges. This new set has the property that it no longer includes edge e'. We have found a larger matching for the case $u_1 = x - 2$, contradicting our assumption. In both cases, $u(x-2) = u(x) + 2$ follows from our premises. □

4 Concluding Remarks and Future Work

In this paper, we showed that the weighted maximum clique problem and the maximum matching problem are FPPT when parameterized by modular-width. It would be interesting to find out whether other problems are FPPT when parameterized by this parameter. It was shown that the maximum network flow problem is FPPT with respect to treewidth as parameter [16]. We know that the maximum network flow problem is not easier than the maximum matching problem from a parallel complexity standpoint, being P-Complete. However, we believe that the maximum network flow problem would also fall in FPPT when parameterized by modular-width. Moreover, it was shown that the chromatic number, hamiltonian cycle, maximum cut, and edge dominating set problems are FPT when parameterized by treewidth but become W[1]-hard when parameterized by clique-width. Also, when parameterized by modular-width, chromatic number and hamiltonian cycle are FPT, while the other two are still open. We conjecture that the chromatic number problem parameterized by modular-width is not FPPT, mainly because Miyano [19] showed that most of the lexicographically first maximal subgraph problems are still P-complete even if the instances are restricted to graphs with bounded degree three.

References

1. Abu-Khzam, F.N., Li, S., Markarian, C., Meyer auf der Heide, F., Podlipyan, P.: On the parameterized parallel complexity and the vertex cover problem. In: Chan, T.-H.H., Li, M., Wang, L. (eds.) COCOA 2016. LNCS, vol. 10043, pp. 477–488. Springer, Cham (2016). doi:10.1007/978-3-319-48749-6_35
2. Bodlaender, H., Downey, R., Fellows, M.: Applications of parameterized complexity to problems of parallel and distributed computation (1994, unpublished extended abstract)
3. Bodlaender, H.L.: Treewidth: characterizations, applications, and computations. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 1–14. Springer, Heidelberg (2006). doi:10.1007/11917496_1
4. Bodlaender, H.L., Koster, A.M.: Combinatorial optimization on graphs of bounded treewidth. Comput. J. **51**(3), 255–269 (2008)
5. Bui-Xuan, B.-M., Telle, J.A., Vatshelle, M.: Boolean-width of graphs. Theoret. Comput. Sci. **412**(39), 5187–5204 (2011)

6. Cesati, M., Di Ianni, M.: Parameterized parallel complexity. In: Pritchard, D., Reeve, J. (eds.) Euro-Par 1998. LNCS, vol. 1470, pp. 892–896. Springer, Heidelberg (1998). doi:10.1007/BFb0057945
7. Chein, M., Habib, M., Maurer, M.-C.: Partitive hypergraphs. Discret. Math. **37**(1), 35–50 (1981)
8. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Inf. Comput. **85**(1), 12–75 (1990)
9. Dahlhaus, E.: Efficient parallel modular decomposition (extended abstract). In: Nagl, M. (ed.) WG 1995. LNCS, vol. 1017, pp. 290–302. Springer, Heidelberg (1995). doi:10.1007/3-540-60618-1_83
10. Fomin, F.V., Golovach, P.A., Lokshtanov, D., Saurabh, S.: Clique-width: on the price of generality. In: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 825–834. Society for Industrial and Applied Mathematics (2009)
11. Fomin, F.V., Golovach, P.A., Lokshtanov, D., Saurabh, S.: Algorithmic lower bounds for problems parameterized by clique-width. In: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 493–502. Society for Industrial and Applied Mathematics (2010)
12. Fomin, F.V., Golovach, P.A., Lokshtanov, D., Saurabh, S.: Intractability of clique-width parameterizations. SIAM J. Comput. **39**(5), 1941–1956 (2010)
13. Gajarský, J., Lampis, M., Ordyniak, S.: Parameterized algorithms for modular-width. In: Gutin, G., Szeider, S. (eds.) IPEC 2013. LNCS, vol. 8246, pp. 163–176. Springer, Cham (2013). doi:10.1007/978-3-319-03898-8_15
14. Ganian, R., Hliněný, P., Nešetřil, J., Obdržálek, J., Ossona de Mendez, P., Ramadurai, R.: When trees grow low: shrubs and fast MSO_1. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 419–430. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32589-2_38
15. Habib, M., Paul, C.: A survey of the algorithmic aspects of modular decomposition. Comput. Sci. Rev. **4**(1), 41–59 (2010)
16. Hagerup, T., Katajainen, J., Nishimura, N., Ragde, P.: Characterizing multiterminal flow networks and computing flows in networks of small treewidth. J. Comput. Syst. Sci. **57**(3), 366–375 (1998)
17. Lagergren, J.: Efficient parallel algorithms for graphs of bounded tree-width. J. Algorithms **20**(1), 20–44 (1996)
18. Miller, G.L., Reif, J.H.: Parallel tree contraction-part I: fundamentals (1989)
19. Miyano, S.: The lexicographically first maximal subgraph problems: P-completeness and NC algorithms. Math. Syst. Theory **22**(1), 47–73 (1989)
20. Oum, S.-I.: Rank-width and vertex-minors. J. Comb. Theory Ser. B **95**(1), 79–100 (2005)