Parameterized Parallel Complexity

Marco Cesati[*] **Miriam Di Ianni** [†]

January 31, 1997

Abstract. We consider the framework of Parameterized Complexity and we investigate the issue of which problems do admit efficient fixed parameter parallel algorithms. In particular, we introduce two classes of efficiently parallelizable parameterized problems, PNC and FPP, according to the degree of efficiency we want to obtain. We sketch both some FPP-algorithms solving natural parameterized problems and a useful tool for proving membership to FPP based on the concept of treewidth. We then focus our attention on parameterized parallel intractability and prove that a necessary condition for a parameterized problem to be complete for the class of sequentially fixed parameter tractable problems (with respect to reductions preserving membership to PNC) is that it is not in NC for some fixed value of the parameter (unless P = NC). Finally, we give two alternative characterizations of both PNC and FPP and we use them to prove the PNC-completeness of two natural parameterized problems.

Topics: Computational Complexity, Parameterized Complexity, Parallel Computations.

1 Introduction

The theory of NP-completeness [18] is a theoretical framework to explain the apparent asymptotical intractability of many problems. Yet, while many natural problems are intractable in the limit, the way by which they arrive at the intractable behaviour can vary considerably. The standard NP-completeness model is often too coarse to give insight into this variation. Informally, consider the following situation: suppose the instance of an NP-complete problem Π consists of two parts x and k, and suppose the problem has some practical relevance especially for k belonging to a small subset S of the set of all its possible values. In this case it makes sense to pose the question: is Π tractable when k is restricted to belong to S? Of course, the answer to the question depends on the problem Π. For instance, if $\Pi =$ COLORABILITY and k is the number of colours needed to properly color a graph, the answer is probably no (indeed, COLORABILITY is NP-complete for every $k \geq 3$ [18]). On the other hand, several degrees of tractability for fixed values of k could exist. The best known polynomial-time algorithm for DOMINATING SET [18] has running time $\mathcal{O}(n^{k+1})$, where n is the number of vertices and k denotes the cardinality of the dominating set. Instead, VERTEX COVER [18] can be solved in time $\mathcal{O}(n)$ when the cardinality of the vertex cover is fixed. Thus, these two latter problems seem to have different "parameterized" complexity.

[*]Dipartimento di Informatica, Sistemi e Produzione, University of Rome "Tor Vergata", via della Ricerca Scientifica, I-00133 Rome, Italy. E-mail: `cesati@dsi.uniroma1.it`

[†]Dipartimento di Scienze dell'Informazione, University of Rome "La Sapienza", via Salaria 113, I-00198 Rome, Italy. E-mail: `diianni@dsi.uniroma1.it`

The first authors who defined a formal setting to deal with parameterized complexity are Downey and Fellows: they assigned the degree of fixed parameter tractability (FPT) to problems behaving like Vertex Cover, and then they introduced several classes containing likely fixed parameter intractable problems [12, 1, 16, 13, 15, 14].

In this paper we investigate the issue of which problems do admit efficient fixed parameter parallel algorithms. A first attempt to formalize the concept of efficiently fixed parameter parallelizable problems has been pursued by Bodlaender, Downey and Fellows: in a one-page abstract [6] they suggested the introduction of the class PNC as the parameterized analogue of NC. However, neither theoretical results nor applications to concrete natural problems were presented. We now want to give a deeper insight to such concept.

According to the degree of efficiency we are interested to obtain, several kinds of efficient parallelization for parameterized problems can be considered. While in the classical complexity theory an efficient parallel algorithm is usually defined as a PRAM-algorithm running in polylogarithmic time and using a polynomial number of processors, in the parameterized setting we must explicitly state the role plaid by the parameter in the degree both of the polylogarithm and of the polynomial. More precisely, we could think of a parallel algorithm requiring a polynomial number of processors, where the degree of the polynomial depends or does not depend on the parameter; similarly, we could think of a parallel algorithm whose running time is bounded by a polylogarithm having degree which is, or is not, a function of the parameter.

The number of available processors is actually a strong physical constraint on the employability of a parallel algorithm; therefore we assume that a parallel algorithm requiring a number of processors bounded by a polynomial whose degree is an increasing function of the parameter cannot be regarded as "efficient" from a parameterized point of view. This fact lead us to consider only two classes of efficiently parallelizable parameterized problems (PNC and FPP), which differ only for the dependence of the degree of the polylogarithm upon the parameter.

In section 2 we formally define the two classes of efficiently parallelizable parameterized problems, PNC and FPP, and we study their relationship with the class of sequentially tractable parameterized problems (FPT). We also show two FPP-algorithms solving the natural parameterized problems Vertex Cover and Max Leaf Spanning Tree. In section 3 we present a useful and non trivial tool for proving FPP-membership of parameterized graph problems based on the concept of treewidth and on the results in [3, 8, 5]. We also state the FPP-membership of the Textwidth and Feedback Vertex Set problems. In section 4 we study the relationship between NC, PNC, and FPP, and we show that there are problems probably not in PNC yet included in NC for any fixed value of the parameter, despite all FPT-complete problems (with respect to any parameterized reduction preserving membership to PNC) must have some value of the parameter such that the problem is not in NC (unless P = NC). Thus, the class NC and the classes FPP and PNC are somewhat "orthogonal". In section 5 we give two alternative characterizations of both FPP and PNC, and we use them to prove the PNC-completeness of two parameterized problems. Finally, in section 6 we state some conclusions.

1.1 The parameterized complexity setting

Let Σ be a finite alphabet. A *parameterized problem* is a set $L \subseteq \Sigma^* \times \Sigma^*$. Tipically, the second component represents a parameter $k \in \mathbb{N}$. The kth slice of the problem is defined as

$L_k = \{ x \in \Sigma^* \colon \langle x, k \rangle \in L \}$. The class FPT of *fixed parameter tractable problems* contains all parameterized problems that have a (sequential, deterministic) solving algorithm with running time bounded by $f(k)\,|x|^\alpha$, where $\langle x, k\rangle$ is the instance of the problem, k is the parameter, f is an arbitrary function and α is a constant independent of x and k.

Given two parameterized problems L and L', L *fixed parameter reduces* to L' if there exist two functions f and g, a constant α and an algorithm Φ computing for each pair $\langle x, k\rangle$ a pair $\langle x', g(k)\rangle$ such that Φ runs in time $f(k)\,|x|^\alpha$ and $\langle x, k\rangle \in L$ if and only if $\langle x', g(k)\rangle \in L'$.

Problems that are likely fixed parameter intractable are organized as a hierarchy of classes W[1] $\subseteq$ W[2] $\subseteq \ldots$, each defined as the closure under parameterized reductions of a "kernel" problem. A boolean expression X is called *t-normalized* if X is in *product-of-sums-of-products...* (t alternations) form. For each value of t we consider the WEIGHTED t-NORMALIZED SATISFIABILITY problem, whose yes-instances are t-normalized boolean expressions having satisfying truth assigments with a parameterized number of 1's (i.e., variables set to *true*). The class W[t] (for $t \geq 2$) contains all parameterized problems that fixed parameter reduce to WEIGHTED t-NORMALIZED SATISFIABILITY, while W[1] is defined as containing all parameterized problems which fixed parameter reduce to WEIGHTED 2-NORMALIZED SATISFIABILITY restricted to size-2 clauses. The W hierarchy contains many natural parameterized problems; see for example [15, 11].

2 The classes PNC and FPP

From now on, with the term "parallel algorithm" we always refer to a PRAM algorithm.

The first class of efficiently parallelizable parameterized problems, PNC, has been defined by Bodlaender, Downey and Fellows [6]. PNC (*parameterized analog of* NC) contains all parameterized problems which have a parallel (deterministic) solving algorithm with at most $g(k)\,|x|^\beta$ processors and running time bounded by $f(k)(\log|x|)^{h(k)}$, where $\langle x, k\rangle$ is the instance of the problem, k is the parameter, f, g and h are arbitrary functions, and β is a constant independent of x and k. The definition of PNC is coherent with the intuition behind the concept of efficiently parallelizable problem, as stated in the following lemma.

Lemma 1 PNC *is a subset of* FPT.

Proof. Let L be a problem in PNC, and let Φ be a parallel algorithm which decides any instance $\langle x, k\rangle$ of L in at most $f(k)(\log n)^{h(k)}$ steps and with at most $g(k)\,n^\beta$ processors (where $n = |x|$). Then, let Ψ be the sequential algorithm which simulates Φ in at most $f(k)\,g(k)\,n^\beta\,(\log n)^{h(k)}$ steps. Since

$$\lim_{n\to\infty} \frac{(\log n)^{h(k)}}{n} = 0,$$

there exists $N = N(h(k))$ such that, for each $n > N$, $(\log n)^{h(k)} < n$. Moreover, if $n < N$, then $(\log n)^{h(k)} < (\log N)^{h(k)} = l(k)$. Thus, for each n, $(\log n)^{h(k)} < l(k)\,n$ and therefore the algorithm Ψ terminates in at most $f(k)\,g(k)\,l(k)\,n^{\beta+1}$ steps. □

A drawback with the definition of PNC is the exponent in the logarithm which bounds the running time. Since it depends on k, it grows very rapidly thus making the running time very close to a linear function also for not too large values of the parameter. The class of *fixed-parameter parallelizable problems* FPP contains all parameterized problems that have a parallel (deterministic) solving algorithm with at most $g(k)\,|x|^{\beta}$ processors and running time bounded by $f(k)(\log|x|)^{\alpha}$, where $\langle x,k\rangle$ is the instance of the problem, k is the parameter, f and g are arbitrary functions, and α and β are constants independent of x and k. Observe that, by definition, FPP $\subseteq$ PNC.

The previously defined classes are worthy of being considered when it is proved that they include natural parameterized problems. A first step in this direction is taken in the remaining of this section.

Let us consider the Vertex Cover problem: given a graph $G = (V, E)$, is there a set of vertices $V' \subseteq V$ of cardinality at most k, such that for every edge $uv \in E$, at least one of its ends, u or v, is in V' (k being the parameter)? The following sequential algorithm is due to Buss [9, 14]. Observe that, for a simple graph G, any vertex of degree greater than k must belong to every k-vertex cover of G.

- **Step 1.** Compute the set $W = \{\, v \mid v \in V \text{ and degree}\,(v) > k \,\}$ and $p = |W|$. If $p > k$, then there is no k-vertex cover, otherwise, let $k' = k - p$.
- **Step 2.** Compute $H = G - W$; if $|E(H)| > k'k$ then reject. Otherwise, compute the graph H' obtained from H by discarding all vertices of degree 0.
- **Step 3.** Now H' is a graph of degree smaller than $k+1$, with at most kk' edges and $2kk'$ vertices. Exhaustively search in H' a k'-vertex cover; if it does not exist, then reject, else the k'-vertex cover of H' plus W is a k-vertex cover of G.

It is easy to verify that the algorithm is correct, because a simple graph with a k'-vertex cover and degree bounded by k has no more than kk' edges, and therefore no more than $2kk'$ non-isolated vertices. The algorithm has running time in $\mathcal{O}(n + k^k)$ and is easily parallelizable. Indeed, in Appendix A an EREW PRAM algorithm is shown which requires $4\log n + \mathcal{O}(k^k)$ time and uses n^2 processors. Thus, Vertex Cover belongs to FPP.

Let us now consider the Max Leaf Spanning Tree problem: given a graph $G = (V, E)$, is there a spanning tree of G with at least k leaves (k being the parameter)? In [11] it is shown that Max Leaf Spanning Tree can be solved in time $\mathcal{O}(n + (2k)^{4k})$ (where $n = |V|$). The simple algorithm follows.

- **Step 1.** Check whether G is connected (if not, the answer is *no*), and whether there is a vertex of degree $\geq k$ (in this case the answer is *yes*).
- **Step 2.** If the answer is still undetermined, then resolve any *useless* vertex of G (a vertex v is useless if it has neighbors u, w of degree 2; such a vertex v is resolved by deleting v from G and adding an edge between u and w). If the new graph G' has at least $3k(k+1)$ vertices, then the answer is *yes*.
- **Step 3.** Otherwise, exhaustively analyze G' and answer accordingly, since G' has a k-leaf spanning tree if and only if G does.

As in the case for Vertex Cover, this algorithm is easily parallelizable (just consider that the problem of deciding whether a graph G is connected is in NC, see for example [20]). Thus, Max Leaf Spanning Tree is in FPP.

3 Treewidth and FPP-membership

The concept of *treewidth* was introduced by Robertson and Seymour [23], and it has proved to be a useful tool in the design of graph algorithms. A *tree decomposition* of an undirected graph $G = (V, E)$ is a pair $\langle T, \mathcal{U} \rangle$, where $T = (X, F)$ is a tree and $\mathcal{U} = \{ U_x \,|\, x \in X \}$ is a family of subsets of V such that

1. $\bigcup_{x \in X} U_x = V$;
2. for all $vw \in E$, there exists an $x \in X$ such that $\{ v, w \} \subseteq U_x$;
3. for all $x, y, z \in X$, if y is on the path from x to z in T, then $U_x \cap U_z \subseteq U_y$.

The *width* of a tree decomposition $\langle T, \{ U_x \,|\, x \in X \} \rangle$ is $\max_{x \in X} |U_x| - 1$. The *treewidth* of a graph G, denoted as $tw(G)$, is the smallest treewidth of any tree decomposition of G.

Let us consider the parameterized TREEWIDTH problem: given a graph G and an integer k, has G treewidth exactly k? Although the general problem is NP-complete [2], Bodlaender [4] showed that there exists a linear time algorithm able to compute, for any fixed k, an optimal tree decomposition of a graph G if its treewidth is at most k. This proves that parameterized TREEWIDTH belongs to FPT. Successively, many authors showed parallel algorithms for such problem. An optimal parallel algorithm has been achieved by Bodlaender and Hagerup [8] (a direct consequence is that TREEWIDTH belongs also to FPP):

Theorem 1 *For all constants $k \geq 1$ and all integers $n \geq 2$, the following problem can be solved on an EREW PRAM using $\mathcal{O}((\log n)^2)$ time, $\mathcal{O}(n)$ operations and $\mathcal{O}(n)$ space: given a graph G with n vertices, construct a minimum-width tree decomposition of G or correctly decide that $tw(G) > k$.*

The concept of *treewidth* turns out to be an useful tool for proving FPP-membership. Indeed, in [3] it is shown that many problems that are (likely) intractable for general graphs are in NC when restricted to graphs of bounded treewidth. In particular, the authors prove that some graph properties, namely *MS properties* (definable in monadic second order logic with quantifications over vertex and edge sets) and *EMS properties* (that involve counting or summing evaluations over sets definable in monadic second order logic), are verifiable in NC for graphs of bounded treewidth. A careful study of the proofs in [3] reveals that all such properties are also verifiable in FPP under the same hypothesis[1]; therefore

> *all problems involving MS or EMS properties are in* FPP *when restricted to graphs of parameterized treewidth.*

In other words, for any (E)MS property P, the following BOUNDED TREEWIDTH GRAPHS VERIFYING P problem belongs to FPP: given a graph G, is G satisfying P and such that $tw(G) \leq k$ (k being the parameter)?

[1]A crucial idea is that any optimal tree decomposition of width k can be transformed in $\mathcal{O}(\log n)$ time and $\mathcal{O}(n)$ operations on a EREW PRAM into a binary tree decomposition of depth $\mathcal{O}(\log n)$ and width at most $3k + 2$ [8].

Many NP-hard graph problems are (E)MS problems (in Appendix B there is a list of MS and EMS properties). Therefore, all problems listed in Appendix B are in FPP when restricted to graphs with parameterized treewidth.

In a few cases it is possible to prove FPP-membership even without restrictions on the treewidth of the instance graph, since some properties directly imply a bound on the treewidth. Examples of such properties characterize the following classes of graphs:

- series-parallel graphs (treewidth ≤ 2) [5];
- graphs with bandwidth at most k (treewidth $\leq k$) [5];
- graphs with cutwidth at most k (treewidth $\leq k$) [5];
- graphs with node search number at most k (treewidth $\leq k$) [22];
- graphs with vertex separation number at most k (treewidth $\leq k-1$) [22, 21];
- graphs with tangle number at most k (treewidth $\leq k$) [25];
- graphs with a feedback vertex set of size k (treewidth $\leq k+1$) [5];
- outerplanar graphs (treewidth ≤ 2) [5];
- k-outerplanar graphs (treewidth $\leq 3k-1$) [5];
- planar graphs with radius d (treewidth $\leq 3d+1$) [24];
- Halin graphs (treewidth $= 3$) [28, 5];
- $r \times c$ grid graphs (treewidth $= \min\{r, c\}$) [5].

Therefore some very natural problems (like Feedback Vertex Set) are in FPP, because (i) yes-instances have bounded treewidth, and (ii) the corresponding property is (E)MS. It should be stressed that, in order to have FPP-membership, both conditions (i) and (ii) must hold. For instance, the parameterized Bandwith problem is not characterized by an (E)MS property and, indeed, it is hard for any class W[t] [7]. Conversely, problems like Clique or Dominating Set are characterized by EMS properties, yet they do not appear on the above list and are hard for W[1] (over the class of all graphs).

4 Relationship between FPP, PNC, and NC

Trivially, the slices of every parameterized problem belonging to FPP or PNC are included in NC. In order to "separate" FPP and PNC from FPT, it is important to verify whether the converse is true, that is, whether the parameterized versions of all problems whose slices belong to NC are included in FPP or PNC. Consider the Clique problem: given a graph G with n vertices and an integer k, decide if G contains a complete subgraph of k nodes. Each kth slice Clique$_k$ is trivially in NC. Indeed, Clique$_k$ can be easily decided by a parallel algorithm that uses $\mathcal{O}(n^k)$ processors and requires constant time: each processor verify if k nodes assigned to it are a clique, and the algorithm accepts if at least one processor discovers a clique. Since Clique is W[1]-complete [15], it cannot be included in PNC (unless FPT= W[1]). This means that slice-membership to NC is not a sufficient condition for membership to PNC (and, thus, to FPP).

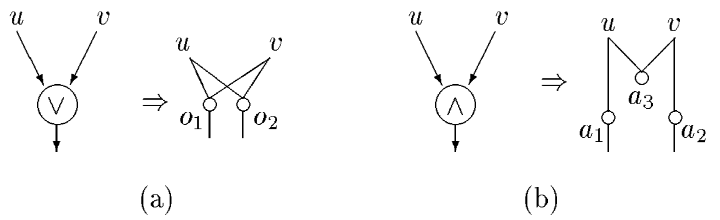


Figure 1: Transformation of inner gates into subgraphs: (a) OR gate, (b) AND gate.

The above fact shows that the classes FPP and PNC are somewhat "orthogonal" to NC, as much as the definition of FPT is orthogonal to the one of P. Nevertheless, there is a strong relation between P-completeness and FPT-completeness, where the completeness for FPT is defined with respect to parameterized reductions preserving membership to FPP or PNC. Consider the parameterized LONGEST PATH (in short, LP) problem: given a graph G and two nodes u and v, decide if G contains a path between u and v of length at least k (k being the parameter). It turns out that the slices LP_k of LP are among the most difficult problems in P, as stated in the following theorem.

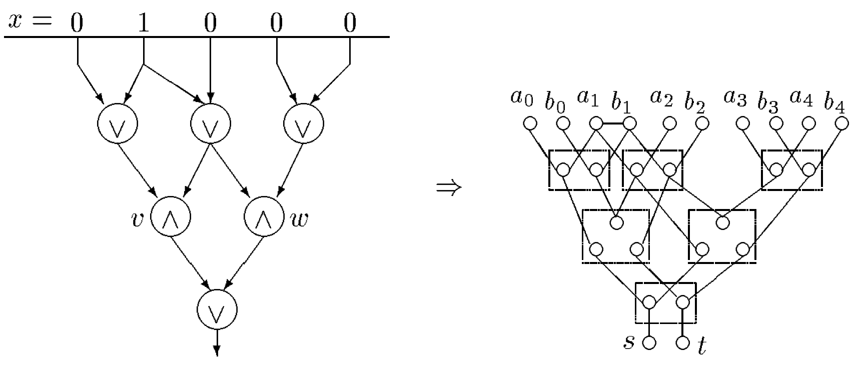
Theorem 2 *For any fixed $k > 1$, the kth slice LP_k of* LP *is* P*-complete.*

Proof. For any fixed $k > 0$, LP_k is trivially included in P.

Let us consider the MONOTONE CIRCUIT VALUE PROBLEM (in short, MCVP): given an encoding $\overline{C}$ of a logical circuit C and a binary input vector $x = x[1], \ldots, x[n]$, does $C(x)$ accept? The circuit is *monotone* because NOT gates are not allowed; moreover, each internal gate has fanin 2 and unbounded fanout. MCVP is a well-known P-complete problem [19]; we now show a logspace reduction from MCVP to LP_2.

Let $\langle \overline{C}, x\rangle$ be an instance of MCVP. The corresponding instance $\langle G_{C,x}, s, t\rangle$ of LP_2 is derived as follows. Each input line i is mapped to a pair of nodes a_i and b_i: they are adjacent if and only if $x[i] = 1$. Each internal gate is transformed into a subgraph (a so-called *metanode*) connected to the rest of the graph by 4 input edges and 2 output edges. An OR gate is transformed into a metanode with two vertices, o_1 and o_2, each of which incident on two input and one output edges: if the input lines of the OR gate are connected to the output lines of, respectively, gate u and gate v, then o_1 (and similarly o_2) is incident to one output edge of the metanode corresponding to u and one output edge of the metanode corresponding to v (see Figure 1 (a)). An AND gate is transformed into a metanode with 3 nodes, a_1, a_2 and a_3: if the input lines of the AND gate are connected to the output lines of, respectively, gate u and gate v, then a_1 (a_2) is incident to one output edge of the metanode corresponding to u (v) and a_3 is incident to the two remaining output edges of the metanodes corresponding to u and v; the two output edges are incident to a_1 and a_2 (see Figure 1 (b)). Finally, the two output edges of the metanode corresponding to the output gate of the circuit are connected, respectively, to node s and node t. In Figure 2 an example of the reduction is shown. Since each metanode can be constructed independently from the others and each gate of the circuit is related to only four metanodes, the reduction can be done in logspace.

In the following we show that $C(x)$ outputs 1 if and only if there is a path from s to t in G of length at least 2. We say that the value of some gate of the circuit C is 1 if the value



(b)

Figure 2: An instance of MCVP transformed into an instance of LP$_2$.

of the boolean expression computed by it on the input vector x is *true*, it is 0 otherwise. For instance, the value of gate v in Figure 2 is 1, the value of w is 0. We now claim that there is a path connecting the two output lines of a metanode of $G_{C,x}$ if and only if the value of the corresponding gate of $C(x)$ is 1. This can be easily verified for the gates connected to the input lines, and the claim follows by induction on the depth of the gate. This is sufficient to prove that $G_{C,x}$ contains a path from s to t if and only if $C(x) = 1$.

Of course, the result for any $k > 2$ follows by replacing the edges incident to s and t with simple paths of total length k. □

The same proof allows us to show that the Disjoint Connecting Paths problem (given a graph G and a set of k pairs of nodes, decide if G contains k disjoint paths connecting each pair, where k is the parameter) has P-complete slices. More noticeably, the previous theorem has an important consequence. Let L, L' be parameterized problems. We say that L FPP-reduces (PNC-reduces) to L' if there is an FPP-algorithm (PNC-algorithm) Φ such that $\Phi(x,k)$ computes $\langle x', k'\rangle$ and $\langle x,k\rangle \in L$ if and only if $\langle x', k'\rangle \in L'$.

Corollary 1 *Let $\leq_\Phi$ be either the* FPP*-reduction or the* PNC*-reduction. If there exists an* FPT*-hard problem L with respect to $\leq_\Phi$ such that its slices L_k are included in* NC*, then* P=NC.

Proof. Since L is FPT-hard and LP is included in FPT [17], then LP $\leq_\phi$ L. This means that there is a PNC-algorithm A transforming an instance $\langle\langle G,s,t\rangle, k\rangle$ of LP into an instance $\langle X, l(k)\rangle$ of L such that $\langle\langle G,s,t\rangle, k\rangle$ is a yes-instance of LP if and only if $\langle X, l(k)\rangle$ is a yes-instance of L. Let us fix a value k. The PNC-algorithm A provides an NC-reduction from LP$_k$ to $L_{l(k)}$; since $L_{l(k)} \in$ NC, LP$_k$ belongs to NC too. From Theorem 2, LP$_k$ is P-complete, and thus the lemma follows. □

Corollary 1 implies that every FPT-hard parameterized problem must have at least one slice which is not in NC unless P = NC.[2] Thus, the notion of FPT-completeness does not add anything to the classical notion of P-completeness to characterize hardly parallelizable problems

[2]Moreover, it is possible to consider the parameterized analogue of the classical logspace reduction, and in this case Corollary 1 would imply that any FPT-complete problem must have at least one P-complete slice.

in the parameterized context. In particular, this means that if we want to distinguish between parallel intractability in the two contexts, we must find an hardly parallelizable parameterized problem having slices in NC. This is the topic of next section. However, notice that in this way we will rely on the hypothesis FPP $\neq$ PNC; therefore we can hope to characterize "FPP's intractability" by proving the PNC-completeness of some problem, but we have no way to characterize "PNC's intractability".

5 Alternative characterizations of FPP and PNC

NC is primarly defined as the class of problems that can be decided by uniform families of polynomial size, polylogarithmic depth circuits. Here "uniform" means that the circuit C_n, able to decide instances of size n, can be derived efficiently (i.e., in logspace) from n. Successively, it is proved that uniform families of circuits and PRAM algorithms are polynomially related.

In this paper, we have followed the inverse pattern, by initially defining the classes FPP and PNC in terms of PRAM algorithms. In this section, we extend the relation between PRAM algorithms and uniform families of circuits to the parameterized setting. In this case, *uniform* means that the circuit C_n^k, able to decide instances of size n when the value of the parameter is k, can be derived in space $f(k)(\log n)^{\alpha}$, for some function f and constant α. Since the proof of the first part of next theorem is trivial and the proof of the second part closely resembles the one of the original theorem in [26], it is omitted.

Theorem 3 *A uniform circuit family of size $s(n, k)$ and depth $d(n, k)$ can be simulated by a PRAM-PRIORITY algorithm using $O(s(n, k))$ active processors and running in time $O(d(n, k))$.*

Conversely, there are a constant c and a polynomial p such that, for any FPP *(*PNC*) algorithm Φ operating in time $T(n, k)$ with processor bound $P(n, k)$, there exist a constant d_Φ and, for any pair $\langle n, k\rangle$, a circuit C_n^k of size at most $d_\Phi\, p(T(n, k), P(n, k), n)$ and depth $c\,T(n, k)$ realizing the same input-output behaviour of Φ on inputs of size n. Furthermore, the family $\{C_n^k\}$ is uniform.*

The previous theorem allows us to show the first PNC-complete problem. It is denoted as Bounded Size-Bounded Depth-Circuit Value Problem (in short, BS-BD-CVP) and is defined as follows: given a constant α, three functions f, g and h, a boolean circuit C with n input lines, an input vector x and an integral parameter k, decide if the circuit C (having size $g(k)\, n^{\alpha}$ and depth $h(k)(\log n)^{f(k)}$) accepts x.

Corollary 2 BS-BD-CVP *is* PNC*-complete with respect to* FPP*-reductions.*

Proof *(sketch).* The problem is hard for PNC. Indeed, Theorem 3 insures that for any parameterized problem L in PNC there are three functions f, g and h, a constant α and a family of boolean circuits $\{C_n^k\}$ of size $g(k)\, n^{\alpha}$ and depth $h(k)(\log n)^{f(k)}$ such that, for every input x of size n and every parameter k, $\langle x, k\rangle \in L$ if and only if $C_n^k(x) = 1$. Since it is possible to build C_n^k in logarithmic space, a triple $\langle C_n^k, x, k\rangle$ such that $\langle x, k\rangle \in L$ if and only if $\langle C_n^k, x, k\rangle \in$ BS-BD-CVP can be derived with an FPP-algorithm.

It is also easy to verify that the problem is included in PNC. Just consider the simple parallel algorithm which activates a processor for each gate of the circuit and computes the value of the output gate in time proportional to the depth of the circuit. □

A second characterization of the classes of parameterized parallel complexity is based on particular alternating Turing machines. Recall that an alternating Turing machine is a non deterministic Turing machine AT whose non final internal states can be *existential* or *universal*: $AT(x)$ accepts if and only if the computations tree contains a subtree S such that:

- if S contains a node u corresponding to a global state whose inner state is existential then it also contains one child of u in $AT(x)$;
- if S contains a node u corresponding to a global state whose inner state is universal then it also contains all children of u in $AT(x)$;
- the boolean expression obtained by associating in S an $\vee$ to any universal node, and $\wedge$ to any universal node, the value *true* to any final accepting state and the value *false* to any final rejecting state is true.

In order to allow underlinear time, we consider *random access alternating Turing machines*, i.e., alternating Turing machines having an appropriate index tape onto which they write the index of the cell of the input tape they need to read. Furthermore, such machines are supplied with six special states, corresponding to the existential and universal versions of the states *read-input*, *0-read* and *1-read*. When the machine enters the existential (universal) state *read-input* it passes into the existential (universal) state *0-read* or *1-read* depending on if the input bit (read in the cell of the input tape whose index is contained in the index tape) is 0 or 1.

There is a strong relation between the time required by a random access alternating Turing machine and the depth of a simulating circuit, and between the space required by a random access alternating Turing machine and the size of a simulating circuit. This fact was pointed out by Chandra, Kozen and Stockmeyer [10] for the classical complexity setting, and it also holds for the parameterized one, as stated in the next theorem. As a consequence of Theorem 3, its proof is equivalent to the one of the corresponding theorem in [10] and thus it is omitted.

Theorem 4 *For any $\alpha > 0$, the class of parameterized problems that can be decided by random access alternating Turing machines with SPACE*$(n,k) \in \mathcal{O}(g(k)\log n)$ and TIME$(n,k) \in \mathcal{O}(h(k)(\log n)^{\alpha})$, for some functions g and h, is included in* FPP. *Conversely, for each parameterized problem in* FPP *there exist an $\alpha > 0$ and a random access alternating Turing machine deciding the problem with SPACE*$(n,k) \in \mathcal{O}(g(k)\log n)$ and TIME$(n,k) \in \mathcal{O}(h(k)(\log n)^{\alpha})$.*

Similarly, the class of parameterized problems that can be decided by random access alternating Turing machines with SPACE$(n,k) \in \mathcal{O}(g(k)\log n)$ and TIME$(n,k) \in \mathcal{O}(h(k)(\log n)^{f(k)})$, for some functions f, g and h, is included in* PNC. *Conversely, for each parameterized problem in* PNC *there exist three functions f, g and h and a random access alternating Turing machine AT deciding the problem with SPACE*$(n,k) \in \mathcal{O}(g(k)\log n)$ and TIME$(n,k) \in \mathcal{O}(h(k)(\log n)^{f(k)})$.*

The previous theorem is a useful tool which allows us to define another PNC-complete problem. We denote it as RANDOM ACCESS ALTERNATING TURING MACHINE COMPUTATION (in short, RA-ATMC): given a random access alternating Turing machine AT, an input word x and an integral parameter k, does $AT(x)$ halts within $\mathcal{O}(f(k)(\log|x|)^k)$ steps with SPACE*$(n) \in \mathcal{O}(g(k)\log n)$?

Corollary 3 RA-ATMC *is* PNC*-complete with respect to* FPP*-reductions.*

Proof *(sketch).* The problem is hard for PNC. Indeed, Theorem 4 insures that for any parameterized problem in PNC there are a random access alternating Turing machine AT and three functions, f, g and h, such that $\langle x, k\rangle \in L$ if and only if $AT(x,k)$ accepts in TIME$(n,k) \in \mathcal{O}(h(k)(\log n)^{f(k)})$ and SPACE*$(n,k) \in \mathcal{O}(g(k)\log n)$. Thus, a triple $\langle AT, x, k\rangle$ such that $\langle x, k\rangle \in L$ if and only if $\langle AT, x, k\rangle \in$ RA-ATMC can be derived in constant time with one processor.

The problem is included in PNC. Indeed, a universal random access alternating Turing machine U accepting an input $\langle AT, x, k\rangle$ if and only if $AT(x,k)$ accepts in TIME$(n,k) \in \mathcal{O}(h(k)(\log n)^{f(k)})$ and SPACE*$(n,k) \in \mathcal{O}(g(k)\log n)$ can be easily derived. □

6 Conclusions

In this paper we have introduced two classes of efficiently parallelizable parameterized problems according to the degree of efficiency we want to obtain. Then, we have restricted our attention on one of them, FPP, which seems to represent a reasonable degree of efficiency while remaining rich enough. Indeed, we have shown many natural parameterized problems belonging to it, both by directly describing FPP-algorithms and by presenting a useful tool for proving FPP-membership based on the concept of treewidth. We have also proved a necessary condition for FPT-completeness: at least one slice cannot be in NC unless P = NC. Finally, we have proposed two alternative characterizations of both FPP and PNC and, by using them, we have shown two PNC-complete problems.

Acknowledgments. We would like to thank Michael R. Fellows and Todd H. Wareham for their helpful comments and their encouragement.

References

[1] Karl A. Abrahamson, Rodney G. Downey, and Michael R. Fellows. Fixed-parameter intractability II (extended abstract). In *Proceedings of 10th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science 665*, 374–385, 1993.

[2] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2), 277–284, 1987.

[3] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12, 308–340, 1991.

[4] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the 25th ACM Annual Symposium on Theory of Computation STOC'93*, 226–234, 1993.

[5] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. Technical Report, Department of Computer Science, Utrecht University, 1995.

[6] Hans L. Bodlaender, Rodney G. Downey, and Michael R. Fellows. Applications of parameterized complexity to problems of parallel and distributed computation, 1994. Private communication.

[7] Hans L. Bodlaender, Michael R. Fellows, and Michael T. Hallett. Beyond NP-completeness for problems of bounded width: Hardness for the W hierarchy (extended abstract). In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing, STOC'94*, 449–458, 1994.

[8] Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. Technical Report UU-CS-1995-25, Department of Computer Science, Utrecht University, 1995.

[9] S. Buss. Private communication. 1995.

[10] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. of the ACM*, 28, 114–133, 1981.

[11] Rod G. Downey and Michael R. Fellows. *Parameterized Complexity.* To appear, 1996.

[12] Rodney G. Downey and Michael R. Fellows. Fixed-parameter intractability (extended abstract). In *Proceedings of the 7th IEEE Conference on Structure in Complexity Theory*, 36–49, 1992.

[13] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87, 161–178, 1992.

[14] Rodney G. Downey and Michael R. Fellows. Parameterized computational feasibility. In P. Clote and J. Remel, editors, *Feasible Mathematics II*, 219–244. Birkhäuser, Boston, 1994.

[15] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM J. Comput.*, 873–921, 1995.

[16] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for $W[1]$. *Theoretical Computer Science*, 141, 109–131, 1995.

[17] Michael R. Fellows and Michael A. Langston. On search, decision and the efficiency of polynomial-time algorithms. In *21th ACM Symposium on Theory of Computing*, 501–512, 1989.

[18] Michael R. Garey and Davis S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[19] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. A compendium of problems complete for P. Technical Report TR91-11, Department of Computer Science and Engineering, University of Washington, December 20, 1991.

[20] Joseph JáJá. *An Introduction to Parallel Algorithms.* Addison–Wesley Publishing Company, 1992.

[21] N. G. Kinnersley. The vertex-separation number of a graph equals its path width. *Inform. Proc. Letters*, 42, 345–350, 1992.

[22] L. M. Kirousis and C. H. Papadimitriou. Searching and pebbling. *Theor. Comp. Sc.*, 47, 205–218, 1986.

[23] Neil Robertson and Paul D. Seymour. Graph Minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7, 309–322, 1986.

[24] Neil Robertson and Paul D. Seymour. Graph Minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36, 49–64, 1984.

[25] Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58, 22–33, 1993.

[26] Larry Stockmeyer and Uzi Vishkin. Simulation of parallel random access machines by circuits. *SIAM J. Comput.*, 13(2), 409–422, 1984.

[27] K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *J. Assoc. Comput. Math*, 29, 623–641, 1982.

[28] T. V. Wimer. *Linear Algorithms on k-Terminal Graphs.* PhD thesis, Dept. of Computer Science, Clemson University, 1987.

Appendix A

We show a parallel implementation of the algorithm for VERTEX COVER sketched in section 2:

```
program { input: $n$, $E$, $k$ }
{ $n$ is the number of vertices of $G$, $E$ is the adjacency matrix of $G$, $k$ is the size of the vertex cover }
begin
  { Phase 1: compute the degree of any vertex and mark any vertex having degree $> k$ }
  for $i := 1$ to $n$ do in parallel
  begin
    for $j := 1$ to $n$ do in parallel
      $D(i,j) := E(i,j)$;
    for $j := 1$ to $\log n$ do
      for $h := 1$ to $n/2^j$ do in parallel
        $D(i,h) := D(i,2h-1) + D(i,2h)$;
    if $D(i,1) > k$ then $m(i) := 1$
    else $m(i) := 0$;
  end;
  { Phase 2: delete all edges which are incident on any marked vertex }
  for $i := 1$ to $n$ do in parallel
    for $j := i+1$ to $n$ do in parallel
      if $(m(i) = 1)$ or $(m(j) = 1)$ then
      begin
        $E(i,j) := 0$; $E(j,i) := 0$;
      end;
  { Phase 3: compute the new degree of any vertex }
  for $i := 1$ to $n$ do in parallel
  begin
    for $j := 1$ to $n$ do in parallel
      $D(i,j) := E(i,j)$;
    for $j := 1$ to $\log n$ do
      for $h := 1$ to $n/2^j$ do in parallel
        $D(i,h) := D(i,2h-1) + D(i,2h)$;
  end;
  { Phase 4: compute the number of marked vertices in the old graph }
  for $i := 1$ to $\log n$ do
    for $j := 1$ to $n/2^i$ do in parallel
      $m(j) := m(2j-1) + m(2j)$;
  if $m(1) > k$ then reject;
  { Phase 5: count the edges in the new graph }
  for $i := 1$ to $\log n$ do
    for $j := 1$ to $n/2^i$ do in parallel
      $D(j,1) := D(2j-1,1) + D(2j,1)$;
  if $D(1,1)/2 > k(k - m(1))$ then reject;
  { now the graph has at most $k(k - m(1))$ edges }
  search exhaustively a vertex cover of size $k - m(1)$ in the new graph
    and, according to the result, accept or reject;
end.
```

It is easy to verify that the running time of the previous algorithm is $4 \log n + \mathcal{O}(k^k)$ and the number of processors is n^2. Therefore, we have proved that VERTEX COVER belongs to FPP.

Appendix B

The following lists are taken from [3, 27]; for definitions see also [18].

MS properties:

domatic number for fixed K, graph K-colorability for fixed K, achromatic number for fixed K, monochromatic triangle, partition into triangles, partition into isomorphic subgraphs for fixed connected H, partition into Hamiltonian subgraphs, partition into forests for fixed K, partition into cliques for fixed K, partition into perfect matchings for fixed K, covering by cliques for fixed K, covering by complete bipartite subgraphs for fixed K, induced subgraph with property P (for monadic second-properties P and fixed K), induced connected subgraph with property P (for monadic second-order properties P and fixed K), induced path for fixed K, cubic subgraph, Hamiltonian completion for fixed K, Hamiltonian circuit, directed Hamiltonian circuit, Hamiltonian path, directed Hamiltonian path, subgraph isomorphism for fixed H, graph contractibility for fixed H, graph homomorphism for fixed H, path with forbidden pairs for fixed n, kernel, degree constrained spanning tree for fixed K, disjoint connecting paths for fixed K, chordal graph completion for fixed K, chromatic index for fixed K, spanning tree parity problem, distance d chromatic number for fixed d and k, thickness $\leq K$ for fixed K, membership for each minor-closed class of graphs.

EMS properties:

vertex cover, dominating set, feedback vertex set, feedback arc set, partial feedback edge set for fixed maximum cycle length l, minimum maximal matching, partition into cliques, clique, independent set, induced subgraph with property P (for monadic second-order properties), induced connected subgraph with property P (for monadic second-order properties), induced path, balanced complete bipartite subgraph, bipartite subgraph, degree-bounded connected subgraph for fixed d, planar subgraph, transitive subgraph, uniconnected subgraph, minimum K-connected subgraph for fixed K, minimum equivalent digraph, Hamiltonian completion, multiple choice matching for fixed J, k-closure, path distinguishers, maximum leaf spanning tree, maximum length-bounded disjoint paths for fixed J, maximum fixed-length disjoint paths for fixed J, minimum edge (vertex) deletion for any MS property, bounded diameter spanning tree for fixed D, multiple choice branching for fixed m, Steiner tree in graphs, maximum cut, longest circuit, longest path, partition into isomorphic subgraphs for fixed H, partition into perfect matching, Kth best spanning tree for fixed K, bounded component spanning forest for fixed K, minimum cut into bounded sets, shortest weight-constrained path, Kth shortest path for fixed K.