ADVICE CLASSES OF PARAMETERIZED COMPLEXITY-CORRIGENDUM

LIMING CAI, JAINER CHEN, ROD DOWNEY, AND MIKE FELLOWS

1. Introduction

The goal of this note is to correct an error in the paper [1] of the authors. In that paper the authors introduced the *advice* view of parameterized complexity. This paper formalized the notion that a language L was fixed parameter tractable iff there was a c and a function f such that on input $\langle x, k\rangle$ we could decide $\langle x, k\rangle \in L$ in time $O(|x|^c) + f(k)$ where we viewed the $f(k)$ as advice for the slices of the language with parameter k. Nowadays we would view this as all FPT problems were kernelizable. (For simplicity, we will ignore issues of uniformity here.)

In the same paper we began to also look at analogs of this idea for other complexity classes. In particular, we introduce the class Logspace + Advice. On input $\langle x, k\rangle$, these were problems which were solvable in space $c \log |x| + f(k)$ for some f, and fixed constant c.

In Theorem 2.3 of that paper we claimed that parameterized Vertex Cover is in the class Logspace + Advice. The proof offered contained a flaw which we wish to repair with this note. This flaw was noticed by Murali Krishna Enduri, and later Jyothi Jyothi, both PhD students. The proof in the original paper was based on the method of bounded search trees which seems intrinsically flawed. The goal of this note is to give a correct proof of the Theorem.

We are indebted to Murali and Jyothi for pointing out the flaw.

We remark that the method we use below seems applicable to a wide class of problems which are proven to be FPT by the method of reduction to a problem kernel with simple local reduction rules. (See for example, Downey and Fellows [3], Chapters 4 and 5, or Cygan, et. al. [2].) Thus it might well be of independent interest.

2. The proof

Theorem 2.1. k-Vertex Cover *is in* Logspace+Advice.

Proof. The algorithm is based on Buss kernelization. An outline of the algorithm is given as follows:

Input: a graph G and parameter k

1. computer the number n of vertices of G;

2. identify the set L of vertices of degree larger than k in G;
3. If $|L| > k$ then stop (no solution) else $k_1 = k - |L|$;
4. identify the set S of vertices of positive degree in the graph $G \setminus L$;
5. If $|S| > 2k_1^2$ (which implies that the number of edges in $G \setminus L$ is larger than k_1^2), then stop (no solution);
6. Construct the graph G' that is $G \setminus L$ with all degree-0 vertices removed;
7. Decide if the graph G' has a vertex cover of at most k_1 vertices.

We assume that the graph G is given in an adjacency list (the algorithm can be modified for the representation of adjacency matrices). Thus, the graph G is given as a list

$$v_1 : w_{1,1}, \ldots, w_{1,s_1}; v_2 : w_{2,1}, \ldots, w_{2,s_2}, \ldots, v_n : w_{n,1}, \ldots, w_{n,s_n},$$

where for each i, $w_{i,1}$, …, w_{i,s_i} are the neighbors of v_i. Without loss of generality, we simply assume that the vertices of G are named by the integers 1, 2, …, n.

Thus, the input (G, k) is given in the input tape of a Turing machine M, and we are going to describe how M uses $O(\log n) + f(k)$ space to decide if there is a vertex cover of at most k vertices for the graph G. For this, we describe how each step of the algorithm above is implemented in the Turing machine M.

A. Step 1. Counting the number n of vertices in G can be easily done in space $O(\log n)$.We save the integer n in the worktape, using $O(\log n)$ space.

B. Steps 2-3. We can easily compute, in space $O(\log n)$, the degree of a vertex by counting the number of neighbors of the vertex, given the adjacency list representation of G. Thus, computing the number $|L|$ of vertices of degree larger than k can be done in $O(\log n + \log k)$ space. Note that we do not store the set L in the worktape.

C. Steps 4-5. Checking if a vertex v_i has a positive degree in the graph $G \setminus L$, i.e., if v_i is in the set S, can be done in $O(\log n + \log k)$ space, as follows. Look at the adjacency list of v_i in G, which is $v_i : w_{i,1}, \ldots, w_{i,s_i}$. First check that $s_i \le k$, then for each j, check if $w_{i,j}$ is of degree $\le k$ in G by checking the adjacency list for the vertex $w_{i,j}$. The vertex v_i is in the set S if and only if $s_i \le k$ and at least one of the neighbors of v_i is also of degree $\le k$. Thus, we can identify all vertices in S (note that we do not store the set S in the worktape). So computing the number $k_2 = |S|$ can be done in $O(\log n + \log k)$ space.

D. Step 6. After having the number $k_2 = |S|$, which is bounded by $2k_1^2 \le 2k^2$ if we reach step 6, we rename the vertices in S as u_1, …, u_{k_2}, where for each h, u_h is encoded using $O(\log k)$ space (e.g., u_h can be simply the integer h), and u_h is the h-th vertex in S following the vertex order given in the adjacency list of G. By **C** above, in space $O(\log n + \log k)$, we can check if a vertex of G is in the set S. Thus, for each vertex u_h in S, we can find the corresponding vertex v_t in G, and vice versa, in $O(\log n + \log k)$ space, For example, for the vertex u_h in S, we simply scan the adjacency list of G, and identify and count the vertices in S, until we encounter the h-th one, which will be the vertex in G that corresponds to u_h in S. For the other direction, given a vertex v_t in G, we scan the adjacency list of G to find the rank h of v_t in the set S, for which the vertex u_h is the corresponding vertex in S.

Now we are ready to construct the graph G' in step 6. For each h, $1 \leq h \leq k_2$, we first find the vertex v_t in the adjacency list of G that corresponding to u_h in S. Now by scanning the adjacency list for v_t in G, we will find all vertices in S that are adjacent to v_t. For each of them, we find the corresponding vertex u_j in S, and add an edge $[u_h, u_j]$ in the adjacency list for u_h for the graph G'. Thus, identifying each edge in the graph G' takes $O(\log n + \log k)$ space. Since each vertex u_h of G' is encoded using $O(\log k)$ bits while the graph G' has no more than $2k^2$ vertices thus no more than $4k^4$ edges, the entire graph G' can be stored in the worktape using space $O(k^4 \log k)$.

E. Step 7. Since the graph G' has its size bounded by a function of k, step 7 will derive a decision in space bounded by a function $f(k)$ of k.

Summarizing the above discussion gives an algorithm of space $O(\log n) + f(k)$ for the Vertex Cover problem.

3. Other claims in [1]

In Theorem (2.4) of [1], it is claimed that the problems Restricted Alternating Hitting Set, Weight $\leq k$ q-CNF Sat, Planar Dominating Set are all in Logspace+Advice.

At this stage, we don't know if these statements are correct:

In the case of Restricted Alternating Hitting Set, the Vertex Cover methods above will work in the case $k_1 = 2$, and hence the simpler Alternating Vertex Cover is indeed in Logspace+Advice, using our method and a bit of counting. For $k_1 > 2$ the problem concerns hypergraphs and there is no longer a high degree rule, like in the Buss' kernel. The same problem applies for Weight $\leq k$ q-CNF Sat (q fixed), which is again about q-hypergraphs. In the case of Planar Dominating Set, the kernelization is quite complex, so it is not completely clear that local methods like ours will work. All of these seem nice problems.

Due to the celebrated of Reingold we know that graph conectivity is in Logspace, so Theorem (2.7) can now be improved to say k-Leaf Spanning Tree is in (uniform) Logspace+Advice.

□

References

[1] L. Cai, J. Chen, R. Downey, and M. Fellows, *Advice Classes of Parameterized Complexity* Ann. Pure and Applied Logic, **84** (1997), 119-138.

[2] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuck, M. Pilipczuck and S. Saurabh), *Parameterized Algorithms* Springer 2015, ISBN 978-3-319-21274-6, pp. 3-555.

[3] R. Downey and M. Fellows, *Fundamentals of Parameterized Complexity*, Springer-Verlag, 2013, texts in computer science, ISBN 978-1-4471-5559-1, xxx+763 pages.

Department of Computer Science, University of Georgia, Athens, GA 30602, USA
E-mail address: cai@cs.uga.edu

Texas A&M University Department of Computer Science College Station Texas 77843-3112, USA
E-mail address: chen@cse.tamu.edu

School of Mathematics and Statistics, Victoria University, PO Box 600, Wellington, New Zealand
E-mail address: rod.downey@vuw.ac.nz

Department of Informatics, University of Bergen, Postboks 7803 5020 Bergen, Norway
E-mail address: Michael.Fellows@uib.no