Randomised Enumeration of Small Witnesses Using a Decision Oracle

Kitty Meeks

School of Mathematics and Statistics, University of Glasgow, Glasgow , UK
kitty.meeks@glasgow.ac.uk

Abstract

Many combinatorial problems involve determining whether a universe of n elements contains a witness consisting of k elements which have some specified property. In this paper we investigate the relationship between the decision and enumeration versions of such problems: efficient methods are known for transforming a decision algorithm into a search procedure that finds a single witness, but even finding a second witness is not so straightforward in general. In this paper we show that, if the decision version of the problem belongs to FPT, there is a randomised algorithm which enumerates all witnesses in time $f(k) \cdot poly(n) \cdot N$, where N is the total number of witnesses and f is a computable function. This also gives rise to an efficient algorithm to count the total number of witnesses when this number is small.

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity, G.2.1 Combinatorial Algorithms

Keywords and phrases enumeration algorithms, parameterized complexity, randomized algorithms, color coding

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.22

1 Introduction

Many well-known combinatorial decision problems involve determining whether a universe U of n elements contains a witness W consisting of *exactly* k elements which have some specified property. Specifically, we are concerned with problems for which any decision algorithm can be called with input universe $X \subset U$ in order to determine whether there is a witness W for the *original* problem (i.e. with universe U) such that $W \subseteq X$; we will call such problems *self-contained k-witness problems*. Thus the well-studied problems k-Clique, k-Cycle and k-Path are all self-contained k-witness problems, but others such as k-Vertex Cover and k-Dominating Set are not (as we need to preserve information about the relationship of any potential witness to the entire universe U).

While the basic decision versions of self-contained k-witness problems are of interest, it is often not sufficient for applications to output simply "yes" or "no": we need to *find* a witness. The issue of finding a single witness using an oracle for the decision problem has previously been investigated by Björklund, Kaski, and Kowalik [5], motivated by the fact that the fastest known parameterised algorithms for a number of widely studied problems (such as graph motif [4] and k-path [3]) are non-constructive in nature. Moreover, for some problems (such as k-Clique or Independent Set [2] and **p-**Even Subgraph [15]) the only known FPT decision algorithm relies on a Ramsey theoretic argument which says the answer must be "yes" provided that the input graph avoids certain easily recognisable structures.

Following the first approach used in [5], we assume the existence of a deterministic "oracle" (a black-box decision procedure), as follows.

11th International Symposium on Parameterized and Exact Computation (IPEC 2016).
Editors: Jiong Guo and Danny Hermelin; Article No. 22; pp. 22:1–22:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ORA(X)
Input: $X \subseteq U$
Output: 1 if some witness is entirely contained in X; 0 otherwise.

A naïve approach easily finds a single witness using $\Theta(n)$ calls to **ORA**: we successively delete elements of the universe, following each deletion with an oracle call, and if the oracle answers "no" we reinsert the last deleted element and continue. Assuming we start with a yes-instance, this process will terminate when only k elements remain, and these k elements must form a witness. In [5], ideas from combinatorial group testing are used to make a substantial improvement on this strategy for the extraction of a single witness: rather than deleting a single element at a time, large subsets are discarded (if possible) at each stage. This gives an algorithm that extracts a witness with only $2k\left(\log_2\left(\frac{n}{k}\right)+2\right)$ oracle queries.

However, neither of these approaches for finding a single witness can immediately be extended to find *all* witnesses, a problem which is of interest even if an efficient decision algorithm does output a single witness; indeed, it is not even obvious how to find a second witness. Both approaches for finding a first witness rely on the fact that we can safely delete some subset of elements from our universe provided we know that what is left still contains at least one witness; if we need to look for a second witness, the knowledge that at least one witness will remain is no longer sufficient to guarantee we can delete a given subset. Of course, for any self-contained k-witness problem we can check all possible subsets of size k, and hence enumerate all witnesses, in time $O(n^k)$; indeed, if *every* set of k vertices is in fact a witness then we will require this amount of time simply to list them all. However, we can seek to do much better than this when the number of witnesses is small by making use of a decision oracle.

The enumeration problem becomes straightforward if we have an *extension oracle*,[1] defined as follows.

EXT-ORA(X, Y)
Input: $X \subseteq U$ and $Y \subseteq X$
Output: 1 if there exists a witness W with $Y \subseteq W \subseteq X$; 0 otherwise.

The existence of an efficient procedure **EXT-ORA**(X, Y) for a given self-contained k-witness problem allows us to use standard backtracking techniques to devise an efficient enumeration algorithm. We explore a binary search tree of depth $O(n)$, branching at level i of the tree on whether the i^{th} element of U belongs to the solution. Each node in the search tree then corresponds to a specific pair (X, Y) with $Y \subseteq X \subseteq U$; we can call **EXT-ORA**(X, Y) to determine whether any descendant of a given node corresponds to a witness. Pruning the search tree in this way ensures that no more than $O(n \cdot N)$ oracle calls are required, where N is the total number of witnesses.

Note that, with only the basic decision oracle, we can determine whether there is a witness that does *not* contain some element x (we simply call **ORA**$(U \setminus \{x\})$), but we cannot determine whether there is a witness which *does* contain x. However, as we will show in Section 3, there are natural self-contained k-witness problems for which there is no fpt-algorithm for the extension decision problem unless FPT=W[1]. This motivates the development of enumeration algorithms that do not rely on such an oracle.

The main result of this paper is just such an algorithm; specifically, we prove the following theorem.

[1] Such an oracle is sometimes called an *interval* oracle, as in the enumeration procedure described by Björklund, Kaski, Kowalik and Lauri [6] which builds on earlier work by Lawler [19].

▶ **Theorem 1.1.** *There is a randomised algorithm to enumerate all witnesses of size k in a self-contained k-witness problem exactly once, whose expected number of calls to a deterministic decision oracle is at most $2^{O(k)} \log^2 n \cdot N$, where N is the total number of witnesses. Moreover, if an oracle call can be executed in time $g(k) \cdot n^{O(1)}$ for some computable function g, then the expected total running time of the algorithm is*

$$2^{O(k)} \cdot g(k) \cdot n^{O(1)} \cdot N.$$

The key tool we use to obtain this algorithm is a colour coding method, using a family of k-perfect hash functions. This technique was introduced by Alon, Yuster and Zwick in [1] and has been widely used in the design of parameterised algorithms for decision and approximate counting (see for example [14, Chapters 13 and 14] and [11, Chapter 8]), but to the best of the author's knowledge has not yet been applied to enumeration problems.

Theorem 1.1 is proved in Section 4, before some implications of our enumeration algorithm for the complexity of related counting problems are discussed in Section 5. We begin in Section 2 with some background on relevant complexity theoretic notions, before discussing the hardness of the extension version of some self-contained k-witness problems in Section 3.

2 Parameterised enumeration

There are two natural measures of the size of a self-contained k-witness problem, namely the number of elements n in the universe and the number of elements k in each witness, so the running time of algorithms is most naturally discussed in the setting of parameterised complexity. There are two main complexity issues to consider in the present setting: first of all, as usual, the running time, and secondly the number of oracle calls required.

For general background on the theory of parameterised complexity, we refer the reader to [11, 14]. The theory of parameterised enumeration has been developed relatively recently [12, 8, 7], and we refer the reader to [8] for the formal definitions of the different classes of parameterised enumeration algorithms. To the best of the author's knowledge, this is the first occurrence of a randomised parameterised enumeration algorithm in the literature, and so we introduce randomised analogues of the four types of parameterised enumeration algorithms introduced in [8] (for a problem with total input size n and parameter k, and with $f : \mathbb{N} \to \mathbb{N}$ assumed to be a computable function throughout):

- an expected-total-fpt algorithm enumerates all solutions and terminates in expected time $f(k) \cdot n^{O(1)}$;
- an expected-delay-fpt algorithm enumerates all solutions with expected delay at most $f(k) \cdot n^{O(1)}$ between the times at which one solution and the next are output (and the same bound applies to the time before outputting the first solution, and between outputting the final solution and terminating);
- an expected-incremental-fpt algorithm enumerates all solutions with expected delay at most $f(k) \cdot (n+i)^{O(1)}$ between outputting the i^{th} and $(i+1)^{th}$ solution;
- an expected-output-fpt algorithm enumerates all solutions and terminates in expected time $f(k) \cdot (n+N)^{O(1)}$, where N is the total number of solutions enumerated.

Under these definitions, Theorem 1.1 says that, if the decision version of a self-contained k-witness problem belongs to FPT, then there is an expected-output-fpt algorithm for the corresponding enumeration problem.

3 Hardness of the extension problem

Many combinatorial problems have a very useful property, often referred to as *self-reducibility*, which allows a search or enumeration problem to be reduced to (smaller instances of) the corresponding decision problem in a very natural way (see [8, 18, 21]). A problem is self-reducible in this sense if the existence of an efficient decision procedure (equivalent to $\mathbf{ORA}(X)$) implies that there is an efficient algorithm to solve the extension decision problem (equivalent to $\mathbf{EXT\text{-}ORA}(X)$). While many self-contained k-witness problems do have this property, we will demonstrate that there exist self-contained k-witness problems that do not (unless FPT=W[1]), and so an enumeration procedure that makes use only of $\mathbf{ORA}(X)$ and not $\mathbf{EXT\text{-}ORA}(X)$ is desirable.

In order to demonstrate this, we show that there exist self-contained k-witness problems whose decision versions belong to FPT, but for which the corresponding extension decision problem is W[1]-hard. We will consider the following problem, which is clearly a self-contained k-witness problem.

p-Clique or Independent Set
Input: A graph $G = (V, E)$ and $k \in \mathbb{N}$.
Parameter: k.
Question: Is there a k-vertex subset of V that induces either a clique or an independent set?

This problem is known to belong to FPT [2]: all sufficiently large input graphs are yes-instances by Ramsey's Theorem. We now turn our attention to the extension version of the problem, defined as follows.

p-Extension Clique or Independent Set
Input: A graph $G = (V, E)$, a subset $U \subseteq V$ and $k \in \mathbb{N}$.
Parameter: k.
Question: Is there a k-vertex subset S of V, with $U \subseteq S$, that induces either a clique or an independent set?

It is straightforward to adapt the hardness proof for **p-**Multicolour Clique or Independent Set [20, Proposition 3.7] to show that **p-**Extension Clique or Independent Set is W[1]-hard.

▶ **Proposition 3.1.** **p-**Extension Clique or Independent Set *is* W[1]*-hard.*

Proof. We prove this result by means of a reduction from the W[1]-complete problem **p-**Clique. Let (G, k) be the input to an instance of **p-**Clique. We now define a new graph G', obtained from G by adding one new vertex v, and an edge from v to every vertex $u \in V(G)$. It is then straightforward to verify that $(G', \{v\}, k+1)$ is a yes-instance for **p-**Extension Clique or Independent Set if and only if G contains a clique of size k. ◀

This demonstrates that **p-**Extension Clique or Independent Set is a problem for which there exists an efficient decision procedure but no efficient algorithm for the extension version of the decision problem (unless FPT=W[1]). The reduction given here can easily be adapted to demonstrate that the following problem has the same property.

p-Induced Regular Subgraph
Input: A graph $G = (V, E)$ and $k \in \mathbb{N}$.
Parameter: k.
Question: Is there a k-vertex subset of V that induces a subgraph in which every vertex has the same degree?

Indeed, the same method can be applied to any problem in which putting a restriction on the degree of one of the vertices in the witness guarantees that the witness induces a clique (or some other induced subgraph for which it is W[1]-hard to decide inclusion in an arbitrary input graph).

4 The randomised enumeration algorithm

In this section we describe and analyse our randomised witness enumeration algorithm, thus proving Theorem 1.1.

As mentioned above, our algorithm relies on a colour coding technique. A family $\mathcal{F}$ of hash functions from $[n]$ to $[k]$ is said to be *k-perfect* if, for every subset $A \subset [n]$ of size k, there exists $f \in \mathcal{F}$ such that the restriction of f to A is injective. We will use the following bound on the size of such a family of hash functions, proved in [1].

▶ **Theorem 4.1.** *For all $n, k \in \mathbb{N}$ there is a k-perfect family $\mathcal{F}_{n,k}$ of hash functions from $[n]$ to $[k]$ of cardinality $2^{O(k)} \cdot \log n$. Furthermore, given n and k, a representation of the family $\mathcal{F}_{n,k}$ can be computed in time $2^{O(k)} \cdot n \log n$.*

Our strategy is to solve a collection of $2^{O(k)} \cdot \log n$ *colourful enumeration problems*, one corresponding to each element of a family $\mathcal{F}$ of k-perfect hash functions. In each of these problems, our goal is to enumerate all witnesses that are *colourful* with respect to the relevant element f of $\mathcal{F}$ (those in which each element is assigned a distinct colour by f). Of course, we may discover the same witness more than once if it is colourful with respect to two distinct elements in $\mathcal{F}$, but it is straightforward to check for repeats of this kind and omit duplicate witnesses from the output. It is essential in the algorithm that we use a deterministic construction of a k-perfect family of hash functions rather than the randomised construction also described in [1], as the latter method would allow the possibility of witnesses being omitted (with some small probability).

The advantage of solving a number of colourful enumeration problems is that we can split the problem into a number of sub-problems with the only requirement being that we preserve witnesses in which every element has a different colour (rather than all witnesses). This makes it possible to construct a number of instances, each (roughly) half the size of the original instance, such that every colourful witness survives in at least one of the smaller instances. More specifically, for each k-perfect hash function we explore a search tree: at each node, we split every colour-class randomly into (almost) equal-sized parts, and then branch to consider each of the 2^k combinations that includes one (nonempty) subset of each colour, provided that the union of these subsets still contains at least one witness (as determined by the decision oracle). This simple pruning of the search tree will not prevent us exploring "dead-ends" (where we pursue a particular branch due to the presence of a non-colourful witness), but turns out to be sufficient to make it unlikely that we explore very many branches that do not lead to colourful witnesses.

We describe the algorithm in pseudocode (Algorithm 1), making use of two subroutines. In addition to our oracle **ORA**(X), we also define a procedure **RANDPART**(X) which we use, while exploring the search tree, to obtain a random partition of a subset of the universe.

RANDPART(X)

Input: $X \subseteq U$

Output: A partition (X_1, X_2) of X with $||X_1| - |X_2|| \leq 1$, chosen uniformly at random from all such partitions of X.

Algorithm 1: Randomised algorithm to enumerate all k-element witnesses in the universe U, using a decision oracle.

```
1  Construct a family $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$ of $k$-perfect hash functions from $U$ to $[k]$;
2  for $1 \leq r \leq |\mathcal{F}|$ do
3      Initialise an empty FIFO queue $Q$;
4      if ORA($U$) $= 1$ then
5          Insert $U$ into $Q$;
6      end if
7      while $Q$ is not empty do
8          Remove the first element $A$ from $Q$;
9          if $|A| = k$ then
10             if $A$ is not colourful with respect to $f_s$ for any $s \in \{1, \ldots, r-1\}$ then
11                 Output $A$;
12             end if
13         else
14             for $1 \leq i \leq k$ do
15                 Set $A_i$ to be the set of elements in $A$ coloured $i$ by $f_r$;
16                 Set $(A_i^{(1)}, A_i^{(2)})$ = RANDPART($A_i$);
17             end for
18             for each $\mathbf{j} = (j_1, \ldots, j_k) \in \{1,2\}^k$ do
19                 if $|A_i^{(j_\ell)}| > 0$ for each $1 \leq \ell \leq k$ then
20                     Set $A_{\mathbf{j}} = A_i^{(j_1)} \cup \cdots \cup A_i^{(j_k)}$;
21                     if ORA($A_{\mathbf{j}}$) $= 1$ then
22                         Add $A_{\mathbf{j}}$ to $Q$;
23                     end if
24                 end if
25             end for
26         end if
27     end while
28 end for
```

We prove the correctness of the algorithm in Section 4.1, and bound the expected running time in Section 4.2.

4.1 Correctness of the algorithm

In order to prove that our algorithm does indeed output every witness exactly once, we begin by showing that we will identify a given k-element subset X during the iteration corresponding to the hash-function $f \in \mathcal{F}$ if and only if X is a colourful witness with respect to f.

▶ **Lemma 4.2.** *Let X be a set of k vertices in the universe U. In the iteration of Algorithm 1 corresponding to $f \in \mathcal{F}$, we will execute 10 to 12 with $A = X$ if and only if:*

1. *X is a witness, and*
2. *X is colourful with respect to f.*

Proof. We first argue that we only execute lines 10 to 12 with $A = X$ if X is a witness and is colourful with respect to f. We claim that, throughout the execution of the iteration corresponding to f, every subset B in the queue Q has the following properties:

1. there is some witness W such that $W \subseteq B$, and
2. B contains at least one vertex receiving each colour under f.

Notice that we check the first condition before adding any subset A to Q (lines 4 and 27), and we check the second condition for any $A \neq U$ in line 25 (U necessarily satisfies condition 2 by construction of $\mathcal{F}$), so these two conditions are always satisfied. Thus, if we execute lines 10 to 12 with $A = X$, these conditions hold for X; note also that we only execute these lines with $A = X$ if $|X| = k$. Hence, as there is a witness $W \subseteq X$ where $|W| = |X| = k$, we must have $X = W$ and hence X is a witness. Moreover, as X must contain at least one vertex of each colour, and contains exactly k elements, it must be colourful.

Conversely, suppose that $W = \{w_1, \ldots, w_k\}$ is a witness such that $f(w_i) = i$ for each $1 \leq i \leq k$; we need to show that we will at some stage execute lines 10 to 12 with $A = W$. We argue that at the start of each execution of the while loop, if W has not yet been output, there must be some subset B in the queue such that $W \subseteq B$. This invariant clearly holds before the first execution of the loop (U will have been inserted into Q, as U contains at least one witness W). Now suppose that the invariant holds before starting some execution of the while loop. Either we execute lines 10 to 12 with $A = W$ on this iteration (in which case we are done), or else we proceed to line 19. Now, for $1 \leq i \leq k$, set j_i to be either 1 or 2 in such a way that $w_i \in A_i^{(j_i)}$. The subset $A_{\mathbf{j}}$, where $\mathbf{j} = (j_1, \ldots, j_k)$ will then pass both tests for insertion into Q, and $W \subseteq A_{\mathbf{j}}$ by construction, so the invariant holds when we exit the while loop. Since the algorithm only terminates when Q is empty, it follows that we must eventually execute lines 10 to 12 with $A = W$. ◀

The key property of k-perfect families of hash functions then implies that the algorithm will identify every witness; it remains only to ensure that we avoid outputting any witness more than once. This is the purpose of lines 10 to 12 in the pseudocode. We know from Lemma 4.2 that we find a given witness W while considering the hash-function f if and only if W is colourful with respect to f: thus, in order to determine whether we have found the witness in question before, it suffices to verify whether it is colourful with respect to any of the colourings previously considered. (The most obvious strategy for avoiding repeats would be to maintain a list of all the witnesses we have output so far, and check for membership of this list; however, in general there might be as many as $\binom{n}{k}$ witnesses, so both storing this list and searching it would be costly.) Hence we see that every witness is output exactly once, as required.

4.2 Expected running time

We know from Theorem 4.1 that a family $\mathcal{F}$ of k-perfect hash functions from U to $[k]$, with $|\mathcal{F}| = 2^{O(k)} \log n$, can be computed in time $2^{O(k)} n \log n$; thus line 1 can be executed in time $2^{O(k)} n \log n$ and the total number of iterations of the outer for-loop (lines 2 to 34) is at most $2^{O(k)} \log n$.

Moreover, it is clear that each iteration of the while loop (lines 7 to 33) makes at most 2^k oracle calls. If an oracle call can be executed in time $g(k) \cdot n^{O(1)}$ for some computable function g, then the total time required to perform each iteration of the while loop is at most $\max\{|\mathcal{F}|, kn + 2^k \cdot g(k) \cdot n^{O(1)}\} = 2^{O(k)} \cdot g(k) \cdot n^{O(1)}$.

Thus it remains to bound the expected number of iterations of the while loop in any iteration of the outer for-loop; we do this in the next lemma.

▶ **Lemma 4.3.** *The expected number of iterations of the while-loop in any given iteration of the outer for-loop is at most* $N(1 + \lceil \log n \rceil)$*, where* N *is the total number of witnesses in the instance.*

Proof. We fix an arbitrary $f \in \mathcal{F}$, and for the remainder of the proof restrict our attention to the iteration of the outer for-loop corresponding to f.

We can regard this iteration of the outer for-loop as the exploration of a search tree, with each node of the search tree indexed by some subset of U. The root is indexed by U itself, and every node has up to 2^k children, each child corresponding to a different way of selecting one of the two randomly constructed subsets for each colour. A node may have strictly fewer than 2^k children, as we use the oracle to prune the search tree (line 27), omitting the exploration of branches indexed by a subset of U that does not contain any witness (colourful or otherwise). Note that the search tree defined in this way has depth at most $\lceil \log n \rceil$: at each level, the size of each colour-class in the indexing subset is halved (up to integer rounding).

In this search tree model of the algorithm, each node of the search tree corresponds to an iteration of the while-loop, and vice versa. Thus, in order to bound the expected number of iterations of the while-loop, it suffices to bound the expected number of nodes in the search tree.

Our oracle-based pruning method means that we can associate with every node v of the search tree some representative witness W_v (not necessarily colourful), such that W_v is entirely contained in the subset of U which indexes v. (Note that the choice of representative witness for a given node need not be unique.) We know that in total there are N witnesses; our strategy is to bound the expected number of nodes, at each level of the search tree, for which any given witness can be the representative.

For a given witness W, we define a random variable $X_{W,d}$ to be the number of nodes at depth d (where the root has depth 0, and children of the root have depth 1, etc.) for which W could be the representative witness. Since every node has some representative witness, it follows that the total number of nodes in the search tree is at most

$$\sum_{W \text{ a witness}} \sum_{d=0}^{\lceil \log n \rceil} X_{W,d}.$$

Hence, by linearity of expectation, the expected number of nodes in the search tree is at most

$$\sum_{W \text{ a witness}} \sum_{d=0}^{\lceil \log n \rceil} \mathbb{E}\left[X_{W,d}\right] \quad \leq \quad N \sum_{d=0}^{\lceil \log n \rceil} \max_{W \text{ a witness}} \mathbb{E}\left[X_{W,d}\right].$$

In the remainder of the proof, we argue that $\mathbb{E}[X_{W,d}] \leq 1$ for all W and d, which will give the required result.

Observe first that, if W is in fact a colourful witness with respect to f, then $X_{W,d} = 1$ for every d: given a node whose indexing set contains W, exactly one of its children will be indexed by a set that contains W. So we will assume from now on that W intersects precisely ℓ colour classes, where $\ell < k$.

If a given node is indexed by a set that contains W, we claim that the probability that W is contained in the set indexing at least one of its children is at most $\frac{1}{2}^{k-\ell}$. For this to happen, it must be that for each colour i, all elements of W having colour i are assigned to the same set in the random partition. If c_i elements in W have colour i, the probability of this happening for colour i is at most $\left(\frac{1}{2}\right)^{c_i-1}$ (the first vertex of colour i can be assigned to either set, and each subsequent vertex has probability at most $\frac{1}{2}$ of being assigned to this same set). Since the random partitions for each colour class are independent, the probability that the witness W survives is at most

$$\prod_{W \cap f^{-1}(i) \neq \emptyset} \left(\frac{1}{2}\right)^{c_i - 1} = \left(\frac{1}{2}\right)^{k - |\{i : W \cap f^{-1}(i) \neq \emptyset\}|} = \left(\frac{1}{2}\right)^{k-\ell}.$$

Moreover, if W is contained in the set indexing at least one of the child nodes, it will be contained in the indexing sets for exactly $2^{k-\ell}$ child nodes: we must select the correct subset for each colour-class that intersects W, and can choose arbitrarily for the remaining $k - \ell$ colour classes. Hence, for each node indexed by a set that contains W, the *expected* number of children which are also indexed by sets containing W is at most $\left(\frac{1}{2}\right)^{k-\ell} \cdot 2^{k-\ell} = 1$.

We now prove by induction on d that $\mathbb{E}\left[X_{W,d}\right] \leq 1$ (in the case that W is not colourful). The base case for $d = 0$ is trivial (as there can only be one node at depth 0), so suppose that $d > 0$ and that the result holds for smaller values. Then, if $\mathbb{E}[Y|Z = s]$ is the conditional expectation of Y given that $Z = s$,

$$\begin{aligned} \mathbb{E}[X_{W,d}] &= \sum_{t \geq 0} \mathbb{E}[X_{W,d} | X_{W,d-1} = t] \, \mathbb{P}[X_{W,d-1} = t] \\ &\leq \sum_{t \geq 0} t \, \mathbb{P}[X_{W,d-1} = t] \\ &= \mathbb{E}[X_{W,d-1}] \\ &\leq 1, \end{aligned}$$

by the inductive hypothesis, as required. Hence $\mathbb{E}[X_{W,d}] \leq 1$ for *any* witness W, which completes the proof. ◀

By linearity of expectation, it then follows that the expected total number of executions of the while loop will be at most $|\mathcal{F}| \cdot N\,(1 + \lceil \log n \rceil)$, and hence that the expected number of oracle calls made during the execution of the algorithm is at most $2^{O(k)} \log^2 n \cdot N$. Moreover, if an oracle call can be executed in time $g(k) \cdot n^{O(1)}$ for some computable function g, then the expected total running time of the algorithm is

$$2^{O(k)} \cdot g(k) \cdot n^{O(1)} \cdot N,$$

as required.

5 Application to counting

There is a close relationship between the problems of counting and enumerating all witnesses in a self-contained k-witness problem, since any enumeration algorithm can very easily be adapted into an algorithm that simply counts the witnesses. However, in the case that the number N of witnesses is large, an enumeration algorithm necessarily takes time at least $O(N)$, whereas we might hope for much better if our goal is simply to determine the total number of witnesses.

The family of self-contained k-witness problems studied here includes subgraph problems, whose parameterised complexity from the point of view of counting has been a rich topic for research in recent years [13, 16, 17, 9, 10, 20, 15]. Many such counting problems, including those whose decision problem belongs to FPT, are known to be #W[1]-complete (see [14] for background on the theory of parameterised counting complexity). In this section we demonstrate how our enumeration algorithm can be adapted to give an efficient (randomised) algorithm to solve the counting version of a self-contained k-witness problem *when the total number of witnesses is small.* This complements the fact that a simple random sampling algorithm can be used for *approximate* counting when the number of witnesses is very large [20, Lemma 3.4], although there remain many situations which are not covered by either result.

▶ **Theorem 5.1.** *Let* Π *be a self-contained* k*-witness problem, and suppose that* $0 < \delta \leq \frac{1}{2}$ *and* $M \in \mathbb{N}$*. Then there exists a randomised algorithm which makes at most* $2^{O(k)} \log^2 n \; M \log(\delta^{-1})$ *calls to a deterministic decision oracle for* Π*, and*

1. *if the number of witnesses in the instance of* Π *is at most* M*, outputs with probability at least* $1 - \delta$ *the exact number of witnesses in the instance;*
2. *if the number of witnesses in the instance of* Π *is strictly greater than* M*, always outputs "More than* M*."*

Moreover, if there is an algorithm solving the decision version of Π *in time* $g(k) \cdot n^{O(1)}$ *for some computable function* g*, then the expected running time of the randomised algorithm is bounded by* $2^{O(k)} \cdot g(k) \cdot n^{O(1)} \cdot M \cdot \log(\delta^{-1})$*.*

Proof. Note that our randomised enumeration algorithm can very easily be adapted to give a randomised counting algorithm which runs in the same time as the enumeration algorithm but, instead of listing all witnesses, simply outputs the total number of witnesses when it terminates. We may compute explicitly the expected running time of our randomised enumeration algorithm (and hence its adaptation to a counting algorithm) for a given self-contained k-witness problem Π in terms of n, k and the total number of witnesses, N. We will write $T(\Pi, n, k, N)$ for this expected running time.

Now consider an algorithm A, in which we run our randomised counting algorithm for at most $2T(\Pi, n, k, M)$ steps; if the algorithm has terminated within this many steps, A outputs the value returned, otherwise A outputs "FAIL". Since our randomised counting algorithm is always correct (but may take much longer than the expected time), we know that if A outputs a numerical value then this is precisely the number of witnesses in our problem instance. If the number of witnesses is in fact at most M, then the expected running time of the randomised counting algorithm is bounded by $T(\Pi, n, k, M)$, so by Markov's inequality the probability that it terminates within $2T(\Pi, n, k, M)$ steps is at least $1/2$. Thus, if we run A on an instance in which the number of witnesses is at most M, it will output the exact number of witnesses with probability at least $1/2$.

To obtain the desired probability of outputting the correct answer, we repeat A a total of $\lceil \log(\delta^{-1}) \rceil$ times. If any of these executions of A terminates with a numerical answer that is at most M, we output this answer (which must be the exact number of witnesses by the argument above); otherwise we output "More than M."

If the total number of witnesses is in fact less than or equal to M, we will output the exact number of witnesses unless A outputs "FAIL" every time it is run. Since in this case A outputs "FAIL" independently with probability at most $1/2$ each time we run it, the probability that we output "FAIL" on every one of the $\lceil \log(\delta^{-1}) \rceil$ repetitions is at most $(1/2)^{\lceil \log(\delta^{-1}) \rceil} \leq 2^{\log \delta} = \delta$. Finally, note that if the number of witnesses is strictly greater than M, we will always output "More than M" since every execution of A must in this case return either "FAIL" or a numerical answer greater than M.

The total running time is at most $O\left(\log(\delta^{-1}) \cdot T(\Pi, n, k, M)\right)$ and hence, using the bound on the running time of our enumeration algorithm from Theorem 1.1, is bounded by $2^{O(k)} \cdot g(k) \cdot n^{O(1)} \cdot M \cdot \log(\delta^{-1})$, as required. ◀

6 Conclusions and open problems

Many well-known combinatorial problems satisfy the definition of the self-contained k-witness problems considered in this paper. We have shown that, given access to a deterministic oracle for the decision version of a self-contained k-witness problem (answering the question "does this subset of the universe contain at least one witness?"), there is a randomised algorithm which is guaranteed to enumerate all witnesses and whose expected number of oracle calls is at most $2^{O(k)} \log^2 n \cdot N$, where N is the total number of witnesses. Moreover, if the decision problem belongs to FPT (as is the case for many self-contained k-witness problems), our enumeration algorithm is an expected-output-fpt algorithm.

This result also has implications for counting the number of witnesses. In particular, if the total number of witnesses is small (at most $f(k) \cdot n^{O(1)}$ for some computable function f) then our enumeration algorithm can easily be adapted to give an fpt-algorithm that will, with high probability, determine exactly the number of witnesses in an instance of a self-contained k-witness problem. This in fact satisfies the conditions for a FPTRAS (Fixed Parameter Tractable Randomised Approximation Scheme, as defined in [2]), but without requiring the full flexibility that this definition requires: with probability $1 - \delta$ we will output the exact number of witnesses, rather than just an answer that is within a factor of $1 \pm \epsilon$ of this quantity.

While the enumeration problem can be solved in a more straightforward fashion for self-contained k-witness problems that have certain additional properties, we demonstrated that several self-contained k-witness problems do not have these properties, unless FPT=W[1]. A natural line of enquiry arising from this work would be the characterisation of those self-contained k-witness problems that do have the additional properties, namely those for which an fpt-algorithm for the decision version gives rise to an fpt-algorithm for the extension version of the decision problem.

Another key question that remains open after this work is whether the existence of an fpt-algorithm for the decision version of a self-contained k-witness problem is sufficient to guarantee the existence of an (expected-)delay-fpt or (expected-)incremental-fpt algorithm for the enumeration problem. Finally, it would be interesting to investigate whether the randomised algorithm given here can be derandomised.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

2 V. Arvind and Venkatesh Raman. Approximation algorithms for some parameterized counting problems. In *ISAAC 2002*, volume 2518 of *LNCS*, pages 453–464. Springer-Verlag Berlin Heidelberg, 2002.

3 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. arXiv:1007.1161 [cs.DS], 2010.

4 Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Probably Optimal Graph Motifs. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *LIPIcs*, pages 20–31. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013. doi:10.4230/LIPIcs.STACS.2013.20.

5 Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Fast witness extraction using a decision oracle. In *Algorithms – ESA 2014*, volume 8737 of *LNCS*, pages 149–160. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-662-44777-2_13.

6 Andreas Björklund, Petteri Kaski, Łukasz Kowalik, and Juho Lauri. Engineering motif search for large graphs. In *2015 Proc. of the Seventeenth Workshop on Algorithm*

Engineering and Experiments (ALENEX), pages 104–118. SIAM, 2015. doi:10.1137/1.9781611973754.10.

7 Nadia Creignou, Raïda Ktari, Arne Meier, Julian-Steffen Müller, Frédéric Olive, and Heribert Vollmer. Parameterized enumeration for modification problems. In *Language and Automata Theory and Applications*, volume 8977 of *LNCS*, pages 524–536. Springer International Publishing, 2015. doi:10.1007/978-3-319-15579-1_41.

8 Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. In *Mathematical Foundations of Computer Science 2013*, volume 8087 of *LNCS*, pages 290–301. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-40313-2_27.

9 Radu Curticapean. Counting matchings of size k is #W[1]-hard. In *Automata, Languages, and Programming*, volume 7965 of *LNCS*, pages 352–363. Springer Berlin Heidelberg, 2013.

10 Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014*, 2014.

11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer London, 2013.

12 Henning Fernau. On parameterized enumeration. In *Computing and Combinatorics*, volume 2387 of *LNCS*, pages 564–573. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45655-4_60.

13 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

14 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

15 Mark Jerrum and Kitty Meeks. The parameterised complexity of counting even and odd induced subgraphs. arXiv:1410.3375 [math.CO], to appear in *Combinatorica*, 2014.

16 Mark Jerrum and Kitty Meeks. The parameterised complexity of counting connected subgraphs and graph motifs. *Journal of Computer and System Sciences*, 81(4):702–716, 2015. doi:10.1016/j.jcss.2014.11.015.

17 Mark Jerrum and Kitty Meeks. Some hard families of parameterised counting problems. *ACM Transactions on Computation Theory*, 7(3), June 2015. doi:10.1145/2786017.

18 Samir Khuller and Vijay V. Vazirani. Planar graph coloring is not self-reducible, assuming P $\neq$ NP. *Theoretical Computer Science*, 88(1):183–189, 1991. doi:10.1016/0304-3975(91)90081-C.

19 Eugene L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972. doi:10.1287/mnsc.18.7.401.

20 Kitty Meeks. The challenges of unbounded treewidth in parameterised subgraph counting problems. *Discrete Applied Mathematics*, 198:170–194, 2016. doi:10.1016/j.dam.2015.06.019.

21 C. P. Schnorr. Optimal algorithms for self-reducible problems. In *Proc. of the 3rd ICALP*, pages 322–337. Edinburgh University Press, 1976.