Lower Bounds for Three Algorithms for the Transversal Hypergraph Generation

Matthias Hagen*

University of Kassel, Research Group Programming Languages / Methodologies,
Wilhelmshöher Allee 73, D–34121 Kassel, Germany
hagen@uni-kassel.de

Abstract. The computation of all minimal transversals of a given hypergraph in output-polynomial time is a long standing open question known as the transversal hypergraph generation. One of the first attempts on this problem—the sequential method [Ber89]—is not output-polynomial as was shown by Takata [Tak02]. Recently, three new algorithms improving the sequential method were published and experimentally shown to perform very well in practice [BMR03, DL05, KS05]. Nevertheless, a theoretical worst-case analysis has been pending. We close this gap by proving lower bounds for all three algorithms. Thereby, we show that none of them is output-polynomial.

1 Introduction

The transversal hypergraph generation is the problem to compute, for a given hypergraph $\mathcal{H} \subseteq 2^V$ with vertex set V, the transversal hypergraph $Tr(\mathcal{H})$ that consists of all minimal subsets of V having a non-empty intersection with each hyperedge of $\mathcal{H}$. This problem has many applications in such different fields like artificial intelligence and logic [EG95, EG02], computational biology [Dam06], database theory [MR92], data mining and machine learning [GKMT97], mobile communication systems [SS98], and distributed computing [GB85].

Due to the importance of the transversal hypergraph generation there have been various approaches to solve it. But since the size of $Tr(\mathcal{H})$ may be exponential in the size of $\mathcal{H}$, we cannot find an algorithm that runs in time polynomial in the size of the input $\mathcal{H}$. Therefore, another notion of fast solvability has to be used. An algorithm is said to be output-polynomial if its running time is bounded polynomially in the size of the input and output [JPY88]. Finding an output-polynomial algorithm for the transversal hypergraph generation is a long standing open problem [Pap97].

One of the earliest approaches is the sequential method [Ber89]. It computes the transversal hypergraph by iteratively combining transversals of specific subhypergraphs of the input in a brute-force manner. The worst-case analysis of the

* Partially supported by a Landesgraduiertenstipendium Thüringen and the Deutsche Forschungsgemeinschaft (DFG) through project OPAL (optimal algorithms for hard problems in computational biology), NI-369/2.

sequential method took many years until Takata showed that it is not output-polynomial [Tak02]. So far, this is the only proven nontrivial lower bound for any algorithm for the transversal hypergraph generation.

In recent years, several improvements of the sequential method have been published. We focus on the DL-algorithm of Dong and Li [DL05], the BMR-algorithm of Bailey, Manoukian, and Ramamohanarao [BMR03], and the KS-algorithm of Kavvadias and Stavropoulos [KS05]. All three algorithms have been empirically tested on practical instances. Especially the BMR-algorithm performs very well on instances from the data mining field. But while the practical performance of the algorithms has been examined, a theoretical worst-case analysis of their running times has been pending. We close this gap by giving nontrivial lower bounds for all three algorithms. Furthermore, the bounds show that none of the three algorithms is output-polynomial.

The paper is organized as follows. Section 2 contains some basic definitions, a brief recapitulation of the sequential method and its analysis by Takata. In Section 3 we show the DL- and the BMR-algorithm not to be output-polynomial. Section 4 contains the analysis of the KS-algorithm. Some concluding remarks follow in Section 5.

2 Basic Definitions and the Sequential Method

A *hypergraph* $\mathcal{H} = (V, E)$ consists of a set V of vertices and a finite family E of subsets of V—the edges. If there is no danger of ambiguity, we also use the edge set to refer to $\mathcal{H}$. The *size* of $\mathcal{H}$ is the number of occurrences of vertices in the edges. A *transversal* of $\mathcal{H}$ is a set $t \subseteq V$ that has a non-empty intersection with each edge of $\mathcal{H}$. A transversal t is *minimal* if no proper subset of t is a transversal. The set of all minimal transversals of $\mathcal{H}$ forms the *transversal hypergraph* $Tr(\mathcal{H})$. A hypergraph $\mathcal{H}$ is *simple* if it does not contain two hyperedges e, f with $e \subseteq f$. By $\min(\mathcal{H})$ we denote the simple hypergraph consisting of the minimal hyperedges of $\mathcal{H}$ with respect to set inclusion. Since $\min(\mathcal{H})$ can be easily computed in polynomial time and $Tr(\mathcal{H}) = Tr(\min(\mathcal{H}))$ holds for every hypergraph $\mathcal{H}$, we concentrate on the transversal hypergraph generation for simple hypergraphs. But even for simple hypergraphs the size of the transversal hypergraph may be exponential. Hence, there cannot be an algorithm computing the transversal hypergraph in polynomial time. A suitable notion of fast solvability for such kind of problems is that of output-polynomial time [JPY88]. An algorithm is said to be *output-polynomial* if its running time is bounded polynomially in the sum of the sizes of the input and output.

Given simple hypergraphs $\mathcal{H} = \{e_1, e_2, \ldots, e_m\}$ and $\mathcal{H}' = \{e'_1, e'_2, \ldots, e'_{m'}\}$ there are two different "unions", namely

$$\begin{aligned}&\mathcal{H} \cup \mathcal{H}' = \{e_1, e_2, \ldots, e_m, e'_1, e'_2, \ldots, e'_{m'}\} \text{ and}\\ &\mathcal{H} \vee \mathcal{H}' = \{e_i \cup e'_j : i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, m'\}.\end{aligned}$$

Proposition 2.1 ([Ber89]). *Let $\mathcal{H}$ and $\mathcal{H}'$ be two simple hypergraphs. Then $Tr(\mathcal{H} \cup \mathcal{H}') = \min(Tr(\mathcal{H}) \vee Tr(\mathcal{H}'))$.*

Algorithm 1 The Sequential Method

```
1: $Tr(\mathcal{H}_1) \leftarrow \{\{v\} : v \in e_1\}$
2: for $i \leftarrow 2, \dots, m$ do
3:     $Tr(\mathcal{H}_i) \leftarrow \min(Tr(\mathcal{H}_{i-1}) \vee \{\{v\} : v \in e_i\})$
4: end for
5: output $Tr(\mathcal{H}_m)$
```

The sequential method [Ber89] uses Proposition 2.1 to generate the transversal hypergraph as follows. For a hypergraph $\mathcal{H} = \{e_1, e_2, \dots, e_m\}$ let $\mathcal{H}_i = \{e_1, e_2, \dots, e_i\}$, $i = 1, 2, \dots, m$. We then have

$$Tr(\mathcal{H}_i) = \min(Tr(\mathcal{H}_{i-1}) \vee Tr(\{e_i\})) = \min(Tr(\mathcal{H}_{i-1}) \vee \{\{v\} : v \in e_i\})$$

and $Tr(\mathcal{H}) = Tr(\mathcal{H}_m)$. This implies a straightforward iterative computation process—the sequential method. A pseudocode listing is given in Algorithm 1. Despite the simplicity of the sequential method it took a couple of years until Takata [Tak02] presented a nontrivial lower bound using the following inductively defined family of hypergraphs.

$$\begin{aligned} &\mathcal{G}_0 = \{\{v_1\}\} && \text{and} \\ &\mathcal{G}_i = (\mathcal{A} \cup \mathcal{B}) \vee (\mathcal{C} \cup \mathcal{D}), && \text{where } \mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D} \text{ are vertex-disjoint copies of } \mathcal{G}_{i-1}. \end{aligned}$$

Takata showed the sequential method not to be output-polynomial based on the following observations.

Lemma 2.2 ([Tak02]). *We have $|V_{\mathcal{G}_i}| = 4^i$, $|\mathcal{G}_i| = 2^{2(2^i-1)}$, $|Tr(\mathcal{G}_i)| = 2^{2^i-1}$. For $i \geq 2$ and any $e \in \mathcal{G}_i$, it holds that $|Tr(\mathcal{G}_i \setminus \{e\}) \setminus Tr(\mathcal{G}_i)| \geq 2^{(i-2)2^i+2}$.*

From Lemma 2.2 it follows that, independent of the edge ordering, the penultimate (intermediate) result computed by the sequential method on input $\mathcal{G}_i$ is superpolynomial in the size of the input and output (cf. the original paper [Tak02] for more details).

3 The Algorithms of Dong and Li, and Bailey, Manoukian and Ramamohanarao

The border-differential algorithm of Dong and Li [DL05] comes from the data mining field and is intended for mining emerging patterns. The analogy to the generation of hypergraph transversals was already pointed out by Bailey, Manoukian, and Ramamohanarao [BMR03]. A pseudocode listing of the DL-algorithm is given in Algorithm 2.

The algorithm was experimentally evaluated on many practical data mining cases [DL05] whereas a theoretical analysis of the running time was left open. For this purpose the conversion of the algorithm to the hypergraph setting is very fruitful. The only observable difference between the sequential method and

Algorithm 2 The DL-Algorithm

```
1: $Tr(\mathcal{H}_1) \leftarrow \{\{v\} : v \in e_1\}$
2: for $i \leftarrow 2, \ldots, m$ do
3:     $Tr_{guaranteed} \leftarrow \{t \in Tr(\mathcal{H}_{i-1}) : t \cap e_i \neq \emptyset\}$
4:     $e_i^{covered} \leftarrow \{v \in e_i : \{v\} \in Tr_{guaranteed}\}$
5:     $Tr(\mathcal{H}_{i-1})' \leftarrow Tr(\mathcal{H}_{i-1}) \setminus Tr_{guaranteed}$
6:     $e_i' \leftarrow e_i \setminus e_i^{covered}$
7:     for all $t' \in Tr(\mathcal{H}_{i-1})'$ in increasing cardinality order do
8:         for all $v \in e_i'$ do
9:             if $t' \cup \{v\}$ is not superset of any $t \in Tr_{guaranteed}$ then
10:                $Tr_{guaranteed} \leftarrow Tr_{guaranteed} \cup \{t' \cup \{v\}\}$
11:            end if
12:        end for
13:    end for
14:    $Tr(\mathcal{H}_i) \leftarrow Tr_{guaranteed}$
15: end for
16: output $Tr(\mathcal{H}_m)$
```

the DL-algorithm is that the DL-algorithm takes special care on how to perform the minimization of $Tr(\mathcal{H}_{i-1}) \vee \{\{v\} : v \in e_i\}$. But as Takata's analysis showed, the minimization is not the bottleneck of the sequential method. Thus, we can extend Takata's analysis of the sequential method in a straightforward way to the DL-algorithm and get the same lower bound.

Theorem 3.1. *The DL-algorithm is not output-polynomial. Its running time is at least $n^{\Omega(\log\log n)}$, where n denotes the size of the input and output.*

Nevertheless, for hypergraphs with only a few edges of small size the DL-algorithm has been shown experimentally to perform well [DL05]. This property is exploited by the BMR-algorithm [BMR03] (cf. Algorithm 3 for the listing) as it uses the DL-algorithm as a subroutine that computes all minimal transversals for small hypergraphs (line 16 of the listing). The BMR-algorithm on input $\mathcal{H}$ is invoked by the top-level call with the set E of edges of $\mathcal{H}$ and an empty set V_{part}. The global variable Tr is initially empty.

A bottleneck for the running time of the BMR-algorithm is that possibly many of the recursively computed transversals—the set Tr' in the listing—actually are not minimal for the input hypergraph $\mathcal{H}$. We concentrate on this issue and construct a family $\mathcal{G}_i'$ of hypergraphs for which the BMR-algorithm computes too many such non-minimal transversals to run in output-polynomial time. Let $\mathcal{G}'(i) = \{e_i, f_i\}$, where $e_i = \{v_{i^2-i+1}, \ldots, v_{i^2}\}$ and $f_i = \{v_{i^2+1}, \ldots, v_{i^2+i}\}$. We inductively define

$$\mathcal{G}_1' = \{\{v_1\}, \{v_2\}\}, \text{ and}$$
$$\mathcal{G}_i' = (\mathcal{G}_{i-1}' \cup \{\{w_i\}\}) \vee \mathcal{G}'(i), \text{ for } i \geq 2.$$

Note that $\mathcal{G}_{i-1}'$, $\{\{w_i\}\}$ and $\mathcal{G}'(i)$ are pairwise vertex-disjoint simple hypergraphs for $i \geq 2$. To calculate the size of $\mathcal{G}_i'$ and of $Tr(\mathcal{G}_i')$ we have to solve the recurrences

Algorithm 3 The BMR-Algorithm

```
Input: a simple hypergraph, given by the set $E$ of its hyperedges, and a set $V_{part}$ of partitioning vertices
1: $V \leftarrow$ set of all vertices in $E$
2: order vertices in increasing frequency $\Rightarrow [v_1, \dots, v_k]$
3: for $i \leftarrow 1, \dots, k$ do
4:     $E_{part} \leftarrow \emptyset$
5:     $V \leftarrow V \setminus \{v_i\}$
6:     for all $e \in E$ do
7:         if $v_i \notin e$ then
8:             $E_{part} \leftarrow \min(E_{part} \cup (e \setminus V))$
9:         end if
10:    end for
11:    $V_{part} \leftarrow V_{part} \cup \{v_i\}$
12:    $a \leftarrow$ average edge cardinality of $E_{part}$ multiplied by $|E_{part}|$
13:    if $|E_{part}| \geq 2$ and $a \geq 50$ then
14:        recursively call the BMR-algorithm on input $E_{part}$, $V_{part}$
15:    else
16:        compute $Tr(E_{part})$ via the DL-algorithm
17:        $Tr' \leftarrow Tr(E_{part}) \vee \{V_{part}\}$
18:        $Tr \leftarrow \min(Tr \cup Tr')$
19:    end if
20:    $V_{part} \leftarrow V_{part} \setminus \{v_i\}$
21: end for
22: return $Tr$
```

$|\mathcal{G}'_i| = 2 \cdot |\mathcal{G}'_{i-1}| + 2$ and $|Tr(\mathcal{G}'_i)| = |Tr(\mathcal{G}'_{i-1})| + i^2$. With the initial conditions $|\mathcal{G}'_1| = 2$ and $|Tr(\mathcal{G}'_1)| = 1$ we obtain

$$|\mathcal{G}'_i| = 2^{i+1} - 2 \quad \text{and} \quad |Tr(\mathcal{G}'_i)| = \frac{2i^3 + 3i^2 + i}{6}$$

by iteration. As for the number $|V_{\mathcal{G}'_i}|$ of vertices of $\mathcal{G}'_i$, we have $|V_{\mathcal{G}'_i}| = i^2 + 2i - 1$.

The BMR-algorithm iteratively partitions the input hypergraph to obtain smaller hypergraphs where the transversal generation is feasible. The partitioning depends on the vertex frequencies. Hence, we first have to analyze the frequencies of the vertices in $\mathcal{G}'_i$.

Lemma 3.2. *For $i \geq 2$ let $\#_v(i,j)$ and $\#_w(i,j)$ respectively denote the number of occurrences of vertex v_j and w_j in $\mathcal{G}'_i$. Then*

$$\begin{aligned}
&\#_w(i,j) = 0 && \text{for } j \geq i,\\
&\#_w(i,j) > \#_w(i,j+1) && \text{for } 2 \leq j < i,\\
&\#_w(i,2) = \#_v(i,1) = \#_v(i,2), &&\\
&\#_v(i,j) = 0 && \text{for } j \geq i^2 + i,\\
&\#_v(i,j) = \#_v(i,k) && \text{for } l^2 - l + 1 \leq j \leq k \leq l^2 + l, \text{ with } 1 \leq l \leq i,\\
&\#_v(i,j) < \#_v(i,k) && \text{for } 1 \leq j < l^2 - l + 1 \leq k \leq l^2 + l, \text{ with } 2 \leq l \leq i.
\end{aligned}$$

All of the above (in)equalities follow directly from the definition of $\mathcal{G}'_i$ or can be easily proven by induction. From Lemma 3.2 it follows that the vertices from $\mathcal{G}'(i)$ are the last vertices in the vertex ordering computed by the BMR-algorithm on input $\mathcal{G}'_i$. This is crucial for the next step of our analysis in which we examine the recursive calls produced by the BMR-algorithm on input $\mathcal{G}'_i$.

Lemma 3.3. *For $i \geq 4$, the BMR-algorithm on input $\mathcal{G}'_i$ recursively calls the BMR-algorithm at least $2i$ times with a modified $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ as input. Here, modified means that all edges of $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ may additionally include at most half of the vertices of $\mathcal{G}'(i)$.*

Proof. We only examine the last $2i$ vertices processed by the BMR-algorithm. From Lemma 3.2 we know that these are exactly the vertices from $\mathcal{G}'(i)$—contained in the edges e_i and f_i. Let $v'_1, v'_2, \ldots, v'_{2i}$ be any ordering of these vertices. We consider the BMR-algorithm on that ordering.

Let the j-th vertex v'_j, $1 \leq j \leq 2i$, from the above ordering be the current partitioning vertex (line 3 of the BMR-algorithm). After partitioning (lines 5 to 10), the remaining hypergraph has the form

$$(\mathcal{G}'_{i-1} \cup \{\{w_i\}\}) \vee (\{v'_1, \ldots, v'_{j-1}\} \cap x_i),$$

where $x_i = f_i$ if $v'_j \in e_i$, and $x_i = e_i$ if $v'_j \in f_i$. Hence, the remaining hypergraph always is a $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ with at most half of the vertices from $\mathcal{G}'(i)$ in every edge.

Altogether, for each of the last $2i$ vertices the minimal transversals of a modified $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ have to be computed. Note that a modified $\mathcal{G}'_3$ has 15 edges of average size at least 5.4 and thus $a \geq 81$ (line 12). Hence, for $i \geq 4$ the last $2i$ vertices invoke recursive calls of the BMR-algorithm with a modified $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ as input. □

With Lemma 3.3 at hand we can analyze the number of non-minimal transversals computed by the BMR-algorithm.

Lemma 3.4. *Let $i \geq 4$. For the number $\eta(i)$ of non-minimal transversals computed by the BMR-algorithm on input $\mathcal{G}'_i$ we have $\eta(i) \geq 2^{i-1} \cdot i!$.*

Proof. From Lemma 3.3 it follows that there are $2i$ recursive calls with a modified $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ as input. Such a recursive call produces at least all of the minimal and non-minimal transversals of $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ augmented by the current partitioning vertex as transversals for $\mathcal{G}'_i$. But since at least the partitioning vertex is dispensable in these transversals, none of them is minimal for $\mathcal{G}'_i$. There are at least $\eta(i-1) + |Tr(\mathcal{G}'_{i-1})|$ such non-minimal transversals per recursive call. Hence, we have to solve the recurrence

$$\begin{aligned}\eta(i) &\geq 2i \cdot (\eta(i-1) + |Tr(\mathcal{G}'_{i-1})|) \\ &\geq 2i \cdot \eta(i-1).\end{aligned}$$

A straightforward computation yields $\eta(3) = 34$. Hence, $\eta(3) \geq 2^2 \cdot 3!$ and we get $\eta(i) \geq 2^{i-1} \cdot i!$ by iteration. □

Putting all pieces together we are able to give a superpolynomial lower bound on the running time of the BMR-algorithm.

Theorem 3.5. *The BMR-algorithm is not output-polynomial. Its running time is at least $n^{\Omega(\log \log n)}$, where n denotes the size of the input and output.*

Proof. We consider the BMR-algorithm on input $\mathcal{G}'_i$. By $m_i = |V_{\mathcal{G}'_i}| \cdot (|\mathcal{G}'_i| + |Tr(\mathcal{G}'_i)|)$ we denote an upper bound on the size of the input and output. For $i \geq 22$ we have

$$m_i = (i^2 + 2i - 1) \cdot \left(2^{i+1} - 2 + \frac{2i^3 + 3i^2 + i}{6}\right) \leq 2^{3i}.$$

The running time of the BMR-algorithm on input $\mathcal{G}'_i$ is at least $\eta(i)$, the number of non-minimal transversals generated. Thus, to analyze the running time we will show that $\eta(i)$ is superpolynomial in m_i. It suffices to show that

$$2^{i-1} \cdot i! > (2^{3i})^c, \qquad \text{for any constant } c.$$

This is equivalent to $i - 1 + \log(i!) > c \cdot 3i$, for any constant c. Using Stirling's formula we have $\log(i!) \geq i \cdot \log i - i$ and thus it suffices to show $i - 1 + i \cdot \log i - i > c \cdot 3i$, for any constant c. This is equivalent to

$$\frac{\log i}{3} - \frac{1}{3i} > c, \qquad \text{for any constant } c.$$

Since the last equation obviously holds for sufficiently large i, we have proven that $\eta(i)$ is superpolynomial in m_i, namely $\eta(i) = m_i^{\Omega(\log \log m_i)}$. □

4 The Algorithm of Kavvadias and Stavropoulos

A first drawback of the sequential method or the BMR-algorithm that Kavvadias and Stavropoulos [KS05] observe is the memory requirement. Since newly computed transversals have to be checked for minimality against the previously computed minimal transversals, all the previously generated minimal transversals have to be stored. The KS-algorithm tries to overcome this potentially exponential memory requirement by two techniques. The first is to combine vertices that belong exactly to the same hyperedges.

Definition 4.1 (generalized vertex, [KS05]). *Let $\mathcal{H}$ be a hypergraph with vertex set V. The set $X \subseteq V$ is a generalized vertex of $\mathcal{H}$ if all vertices in X belong to exactly the same hyperedges of $\mathcal{H}$.*

While adding edge e_i, and hence generating the minimal generalized transversals of $\mathcal{H}_i$ out of the minimal generalized transversals of $\mathcal{H}_{i-1}$, the generalized vertices have to be updated according to e_i. Kavvadias and Stavropoulos characterize the following three types of generalized vertices X of a minimal generalized transversal t of $\mathcal{H}_{i-1}$.

– type α: $X \cap e_i = \emptyset$. Hence, X is a generalized vertex of $\mathcal{H}_i$.
– type β: $X \subset e_i$. Hence, X is a generalized vertex of $\mathcal{H}_i$.
– type γ: $X \cap e_i \neq \emptyset$ and $X \not\subset e_i$. Here, X is divided into $X_1 = X \setminus (X \cap e_i)$ and $X_2 = X \cap e_i$. Both X_1 and X_2 are generalized vertices of $\mathcal{H}_i$.

Let $\kappa_\alpha(t,i)$, $\kappa_\beta(t,i)$, and $\kappa_\gamma(t,i)$ denote the number of generalized vertices of type α, β, and γ in t according to e_i. When edge e_i is added, the minimal generalized transversal t of $\mathcal{H}_{i-1}$ has to be split into $2^{\kappa_\gamma(t,i)}$ generalized transversals of $\mathcal{H}_{i-1}$—the so-called *offsprings* of t—since all combinations of newly generalized vertices have to be generated. If $\kappa_\beta(t,i) \neq 0$, all these newly generated offsprings are also minimal transversals of $\mathcal{H}_i$. But if $\kappa_\beta(t,i) = 0$, there is a special offspring t_0 of t that contains all the X_1-parts of the γ-type generalized nodes of t. Hence, $t_0 \cap e_i = \emptyset$ and t_0 has to be augmented by a vertex from e_i to be a transversal of $\mathcal{H}_i$. All the other offsprings of t already are minimal transversals of $\mathcal{H}_i$ since they contain at least one X_2-part of a generalized vertex from t.

The second technique to overcome the potentially exponential memory requirement is based on the observation that the sequential method is a form of breadth-first search through a "tree" of minimal transversals. At the ith-level of the "tree" the nodes are the minimal transversals of the partial hypergraph $\mathcal{H}_i$. The descendants of a minimal transversal t at level i are the minimal transversals of $\mathcal{H}_{i+1}$ that include t. Note that, since a node at level $i+1$ may have several ancestors at level i, the structure is not really a tree but very tree-like. The bottom level consists of the minimal transversals of $\mathcal{H}$. When cycling through this "tree" breadth-first, one has to wait very long for the first minimal transversal to be output and some nodes are visited several times because they have more than one ancestor. To overcome the long time that may pass till the first minimal transversal is output, the KS-algorithm uses a depth-first strategy. And to really cycle through a tree and not a tree-like structure with some cycles, Kavvadias and Stavropoulos introduce the notion of so-called appropriate vertices.

Definition 4.2 (appropriate vertex, [KS05]). *Let $\mathcal{H} = \{e_1, \ldots, e_m\}$ be a hypergraph with vertex set V and let t be a minimal transversal of the partial hypergraph $\mathcal{H}_i$ of $\mathcal{H}$. A generalized vertex $v \in V \setminus t$ is an appropriate vertex for t if no other vertex in $t \cup \{v\}$ except v can be removed and the remaining set still be a transversal of $\mathcal{H}_i$. The set $appr(t,e)$ contains all appropriate vertices for t in edge e.*

Note that the special offspring t_0 of a minimal generalized transversal t of $\mathcal{H}_{i-1}$ has to be augmented by a vertex from $appr(t, e_i)$ only. All the other vertices from e_i can be skipped. Expanding only with appropriate vertices ensures that no non-minimal transversals are generated and avoids regenerations. Another advantage is that the previously described transversal "tree" structure becomes a real tree (cf. the original paper [KS05] for more details).

All the described techniques—generalized vertices, depth-first strategy, appropriate vertices—together with the main idea of the sequential method—processing the edges one after the other—are used in the KS-algorithm (cf. Algorithm 4 for the listing).

Algorithm 4 The KS-Algorithm

```
1: express $e_1$ as a set of one generalized vertex
2: compute the transversal $t = Tr(e_1)$
3: AddNextHyperedge$(t, e_2)$

4: procedure AddNextHyperedge$(t, e_i)$
5:     update the set of generalized vertices
6:     express $t$ and $e_i$ as sets of generalized vertices of level $i$
7:     $l \leftarrow 1$
8:     while GenerateNextTransversal$(t, l)$ do
9:         if $e_i$ is the last hyperedge then
10:            output $t'$ without using generalized vertices
11:        else
12:            AddNextHyperedge$(t', e_{i+1})$
13:            $l \leftarrow l + 1$
14:        end if
15:    end while
16: end procedure

17: function GenerateNextTransversal$(t, l)$
18:    if $\kappa_\beta(t, i) \neq 0$ then
19:        if $l \leq 2^{\kappa_\gamma(t,i)}$ then
20:            $t' \leftarrow$ the $l$-th offspring of $t$
21:            return true
22:        else
23:            return false
24:        end if
25:    else if $\kappa_\beta(t, i) = 0$ then
26:        if $l \leq 2^{\kappa_\gamma(t,i)} - 1$ then
27:            $t' \leftarrow$ the $l$-th offspring of $t$ except $t_0$
28:            return true
29:        else if $2^{\kappa_\gamma(t,i)} \leq l \leq 2^{\kappa_\gamma(t,i)} - 1 + |appr(t, e_i)|$ then
30:            $t' = t_0$ augmented by the $(l - 2^{\kappa_\gamma(t,i)} + 1)$-th vertex of $appr(t, e_i)$
31:            return true
32:        end if
33:    else
34:        return false
35:    end if
36: end function
```

As for the running time, the KS-algorithm is experimentally shown [KS05] to be competitive to the sequential method, the BMR-algorithm, and an implementation of Algorithm A of Fredman and Khachiyan [BEGK03, FK96]. We will show that the KS-algorithm is not output-polynomial.

First, we note that there are situations in which the KS-algorithm cannot find an appropriate vertex. Consider for example the hypergraph $\mathcal{H} = \{\{v_1, v_5\}, \{v_2, v_5\}, \{v_3, v_6\}, \{v_4, v_6\}, \{v_5, v_6\}\}$. Having processed all but the last edge, there are no generalized vertices left. We concentrate on the path down the transversal tree that corresponds to choosing v_1, v_2, v_3, and v_4. The intermediate transversal is $t = \{v_1, v_2, v_3, v_4\}$. The only edge left is $\{v_5, v_6\}$. But the KS-algorithm cannot find an appropriate vertex in this edge for t. Hence, there are dead ends in the tree, namely leaves that do not contain a minimal transversal of the input $\mathcal{H}$. The next step is to find hypergraphs with too many such dead ends.

Lemma 4.3. *For $i \geq 3$, the number of dead ends the KS-algorithm has to visit for any of Takata's hypergraphs $\mathcal{G}_i$ as input is at least $2^{(i-2)2^i+1}$, independent of the edge ordering.*

Proof. Consider the hypergraph family $\mathcal{G}_i$ of Takata defined in Section 2. From Lemma 2.2 it follows that, whatever ordering of the edges is chosen, there are at least $2^{(i-2)2^i+2}$ nodes in the penultimate level of the transversal tree described above Definition 4.2. The bottom level of the tree obviously contains $|Tr(\mathcal{G}_i)|$ many nodes—one for each minimal transversal. Since $|Tr(\mathcal{G}_i)| = 2^{2^i-1}$ (cf. Lemma 2.2), there is a decrease in the number of nodes from the penultimate level to the bottom level for $i \geq 3$. This decrease can only be caused by dead ends in the penultimate level. Hence, for $i \geq 3$ there are at least $2^{(i-2)2^i+2} - 2^{2^i-1} \geq 2^{(i-2)2^i+1}$ many dead ends in the penultimate level. □

Using Lemma 4.3 we can show that the KS-algorithm is not output-polynomial.

Theorem 4.4. *The KS-algorithm is not output-polynomial. Its running time is at least $n^{\Omega(\log\log n)}$, where n denotes the size of the input and output.*

Proof. We consider the KS-algorithm on input $\mathcal{G}_i$. By $m_i = |V_{\mathcal{G}_i}| \cdot (|\mathcal{G}_i| + |Tr(\mathcal{G}_i)|)$ we denote an upper bound on the size of $\mathcal{G}_i$ and $Tr(\mathcal{G}_i)$. From Lemma 2.2 we have $m_i = 4^i \cdot (2^{2(2^i-1)} + 2^{2^i-1})$, which results in $m_i \leq 2^{2^{i+2}}$.

Let $\hat{\eta}(i)$ denote the number of dead end situations visited by the KS-algorithm on input $\mathcal{G}_i$. The time, the KS-algorithm needs to compute $Tr(\mathcal{G}_i)$, is at least the number of dead end situations visited. Since the KS-algorithm visits the transversal tree depth-first, it visits all the dead end situations in the penultimate level of the tree. With Lemma 4.3 we have $\hat{\eta}(i) \geq 2^{(i-2)2^i+1}$ for $i \geq 3$. Thus, to analyze the running time we will show that $\hat{\eta}(i)$ is superpolynomial in m_i. It suffices to show that $2^{(i-2)2^i} > (2^{2^{i+2}})^c$, for any constant c. This is equivalent to $i - 2 > 4c$, for any constant c. Since this obviously holds for large enough i, we have proven that $\hat{\eta}(i)$ is superpolynomial in m_i, namely $\hat{\eta}(i) = m_i^{\Omega(\log\log m_i)}$. □

5 Concluding Remarks

We have proven superpolynomial lower bounds for the DL-, the BMR-, and the KS-algorithm in terms of the size of the input and output. Thus, like the underlying sequential method, these three algorithms are not output-polynomial.

We are not aware of any other nontrivial lower bounds for algorithms generating the transversal hypergraph although we suppose that none of the known algorithms is output-polynomial. Extending the existing lower bounds to other algorithms seems to be not that straightforward.

Consider for instance the multiplication method suggested by Takata [Tak02]. Very recently Elbassioni proved a quasi-polynomial upper bound on the running time [Elb06]. But giving a superpolynomial lower bound for the multiplication method requires the construction of new hypergraphs. Takata's hypergraphs $\mathcal{G}_i$ and our hypergraphs $\mathcal{G}'_i$ are solved too fast by the multiplication method.

There are also no nontrivial lower bounds known for Algorithms A and B of Fredman and Khachiyan [FK96]. Though Gurvich and Khachiyan [GK97] note that it should be possible to give a superpolynomial lower bound for Algorithm A using hypergraphs very similar to the $\mathcal{G}_i$, the proof is still open. Giving a lower bound for Algorithm B—considered to be the fastest known transversal hypergraph algorithm—seems to be even more involved.

Acknowledgments I thank Martin Mundhenk and the anonymous referees for their valuable comments and suggestions.

References

[BEGK03] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. Extending the Balas-Yu bounds on the number of maximal independent sets in graphs to hypergraphs and lattices. *Mathematical Programming*, 98(1-3):355–368, 2003.

[Ber89] Claude Berge. *Hypergraphs*, volume 45 of *North-Holland Mathematical Library*. North-Holland, 1989.

[BMR03] James Bailey, Thomas Manoukian, and Kotagiri Ramamohanarao. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*, pages 485–488. IEEE Computer Society, 2003.

[Dam06] Peter Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006.

[DL05] Guozhu Dong and Jinyan Li. Mining border descriptions of emerging patterns from dataset pairs. *Knowledge and Information Systems*, 8(2):178–202, 2005.

[EG95] Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

[EG02] Thomas Eiter and Georg Gottlob. Hypergraph transversal computation and related problems in logic and AI. In Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, editors, *Logics in Artificial Intelligence, European Conference, JELIA 2002, Cosenza, Italy, September, 23-26, Proceedings*, volume 2424 of *Lecture Notes in Computer Science*, pages 549–564. Springer, 2002.

[Elb06] Khaled M. Elbassioni. On the complexity of the multiplication method for monotone CNF/DNF dualization. In Yossi Azar and Thomas Erlebach, editors, *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, volume 4168 of *Lecture Notes in Computer Science*, pages 340–351. Springer, 2006.

[FK96] Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996.

[GB85] Hector Garcia-Molina and Daniel Barbará. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860, 1985.

[GK97] Vladimir Gurvich and Leonid Khachiyan. On the frequency of the most frequently occurring variable in dual monotone DNFs. *Discrete Mathematics*, 169(1-3):245–248, 1997.

[GKMT97] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, and Hannu Toivonen. Data mining, hypergraph transversals, and machine learning. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona*, pages 209–216. ACM Press, 1997.

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[KS05] Dimitris J. Kavvadias and Elias C. Stavropoulos. An efficient algorithm for the transversal hypergraph generation. *Journal of Graph Algorithms and Applications*, 9(2):239–264, 2005.

[MR92] Heikki Mannila and Kari-Jouko Räihä. On the complexity of inferring functional dependencies. *Discrete Applied Mathematics*, 40(2):237–243, 1992.

[Pap97] Christos H. Papadimitriou. NP-completeness: A retrospective. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, 7-11 July 1997, Proceedings*, volume 1256 of *Lecture Notes in Computer Science*, pages 2–6. Springer, 1997.

[SS98] Saswati Sarkar and Kumar N. Sivarajan. Hypergraph models for cellular mobile communication systems. *IEEE Transactions on Vehicular Technology*, 47(2):460–471, 1998.

[Tak02] Ken Takata. On the sequential method for listing minimal hitting sets. In *Proceedings Workshop on Discrete Mathematics and Data Mining, 2nd SIAM International Conference on Data Mining, April 11-13, Arlington, Virginia, USA, 2002*, pages 109–120, 2002.