Fundamenta Informaticae XX (2014) 1–6
DOI 10.3233/FI-2012-0000
IOS Press

A Lower Bound for the HBC Transversal Hypergraph Generation

Khaled Elbassioni
Masdar Institute of Science and Technology
Abu Dhabi, UAE
kelbassioni@masdar.ac.ae

Matthias Hagen
Bauhaus-Universität Weimar
D–99423 Weimar, Germany
matthias.hagen@uni-weimar.de

Imran Rauf
National University of Computer and Emerging Sciences
Karachi, Pakistan
imran.rauf@nu.edu.pk

Abstract. The computation of transversal hypergraphs in output-polynomial time is a long standing open question. An Apriori-like level-wise approach (referred to as the HBC-algorithm or MTminer) was published in 2007 by Hébert, Bretto, and Crémilleux [A Data Mining Formalization to Improve Hypergraph Minimal Transversal Computation, Fundamenta Informaticae, 80(4), 2007, 415–433] and was experimentally demonstrated to have very good performance on hypergraphs with small transversals.

In this short note extending the paper by Hagen [Lower bounds for three algorithms for transversal hypergraph generation, Discrete Applied Mathematics, 157(7), 2009, 1460–1469], we prove a superpolynomial lower bound for the HBC-algorithm. This lower bound also shows that the originally claimed upper bound on the HBC-algorithm's running time is wrong.

Keywords: HBC-algorithm, MTminer, algorithm analysis, lower bound, transversal hypergraph generation

Address for correspondence: Bauhaus-Universität Weimar, D–99423 Weimar, Germany

1. Introduction

Transversal hypergraph generation is the problem to compute, given a 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 covering problem has many applications in very different fields [11, Chapter 3] and several algorithmic approaches have been proposed. However, finding an output-polynomial algorithm is a long standing open problem [16]. Thereby, an algorithm is said to be output-polynomial if its running time is polynomial in the combined size of input and output [14].

Many special cases of transversal hypergraph generation are output-polynomial or fixed-parameter tractable [5, 10, 7] but the currently best algorithmic upper bound for the general problem is $n^{o(\log n)}$ due to Fredman and Khachiyan [8]. For four algorithms there are known *lower* bounds on the running time [17, 12]. These bounds show the sequential method [3] and three improvements (the BMR-algorithm [2], the DL-algorithm [4], and the KS-algorithm [15]) not to be output-polynomial.

In this short note extending the paper by Hagen [12], we focus on the level-wise approach of the HBC-algorithm [13]. While the authors experimentally demonstrate that their HBC-algorithm is very fast on hypergraphs with small transversals, we prove a superpolynomial lower bound for arbitrary inputs.

The paper is organized as follows. After introducing basic definitions and notation in Section 2, we analyze the HBC-algorithm in Section 3. Some concluding remarks follow in Section 4.

2. Preliminaries

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 set $s \subseteq V$ *hits* an edge $e \in E$ if $s \cap e \neq \emptyset$. A *transversal* of $\mathcal{H}$ is a set $t \subseteq V$ that hits 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 in the input size. A suitable notion of fast solvability for such kind of problems is that of output-polynomial time [14]. 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.

3. The Algorithm of Hébert, Bretto, and Crémilleux

The HBC-algorithm [13] (cf. Algorithm 1 for a pseudocode listing) is a level-wise approach to transversal hypergraph generation similar to the classic Apriori algorithm [1]. Note that our presentation of the algorithm slightly differs from the original paper [13] as we avoid using Galois connections and the like.

The HBC-algorithm generates transversal candidates in a level-wise manner. In the first step, one element subsets of V are candidates for being minimal transversals. Those that really are transversals are

Algorithm 1 The HBC-Algorithm

```
    Input:   hypergraph $\mathcal{H}$ on vertex set $V$
    Output:  transversal hypergraph $Tr(\mathcal{H})$
 1: $Tr \leftarrow \{\{v\} : v \in E$ for each $E \in \mathcal{H}\}$
 2: $C_1 \leftarrow \{\{v\} : v \in \mathcal{H}\} \setminus Tr$
 3: $i \leftarrow 1$
 4: while $C_i \neq \emptyset$ do
 5:     for all $a, b \in C_i$, $|a \cap b| = i - 1$ do
 6:         $c \leftarrow a \cup b$
 7:         if $c \setminus \{v\} \in C_i$ for all $v \in c$ then
 8:             if $c \setminus \{v\}$ hits fewer edges of $\mathcal{H}$ than $c$ for all $v \in c$ then
 9:                 if $c$ is a transversal of $\mathcal{H}$ then
10:                     $Tr \leftarrow Tr \cup \{c\}$
11:                 else
12:                     $C_{i+1} \leftarrow C_{i+1} \cup \{c\}$
13:                 end if
14:             end if
15:         end if
16:     end for
17:     $i \leftarrow i + 1$
18: end while
19: output $Tr$
```

added to the set Tr (line 1 of the listing) that finally will contain all minimal transversals of $\mathcal{H}$. All other one element subsets of V are added (line 2) to the candidate set C_1 of level 1. In the $(i+1)$-th step, the HBC-algorithm combines candidates of the i-th step that have a large enough intersection (lines 5 and 6). For each candidate c it is then checked whether all its subsets are candidates at level i and whether c hits more edges than all its subsets (lines 7 and 8). If so, a final check determines whether c is a transversal or not and whether c has to be added to Tr or C_{i+1}, respectively (lines 9 to 13).

In [13] it is claimed that the HBC-algorithm runs in $O(2^{t(\mathcal{H})}|Tr(\mathcal{H})|)$ time, where $t(\mathcal{H})$ denotes the size of a largest minimal transversal of $\mathcal{H}$. This bound seems to be intuitive as the HBC-algorithm checks all the subsets of each minimal transversal. Furthermore, it would solve a longstanding open question. Namely, polynomial time decision of $\mathcal{G} = Tr(\mathcal{H})$ for given hypergraphs $\mathcal{G}$ and $\mathcal{H}$ when each edge of $\mathcal{G}$ only has logarithmic size. Unfortunately, as we will show in this paper, the claimed upper bound is wrong. We shall give a lower bound that also shows the HBC-algorithm not to be output-polynomial.

To obtain our lower bound, we will observe the behavior of the HBC-algorithm on special inputs. In the construction of these inputs we use the following notation. Given simple hypergraphs $\mathcal{H} = \{e_1, e_2, \ldots, e_m\}$ and $\mathcal{H}' = \{e'_1, e'_2, \ldots, e'_{m'}\}$, there are the following two different "unions."

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

Takata [17] presented the first nontrivial lower bound for any transversal hypergraph generation algorithm, namely for the sequential method [3], using the following inductively defined family of hypergraphs.

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

These hypergraphs were also valuable in the analysis of Hagen [12] who showed the DL- and the KS-algorithm not to be output-polynomial on the $\mathcal{G}_i$'s. As we will also use these hypergraphs in our analysis of the HBC-algorithm, we will need the following observations by Takata [17].

Lemma 3.1. (Proofs in [17])
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 $|Tr(\mathcal{G}_i \setminus \{e\}) \setminus Tr(\mathcal{G}_i)| \geq 2^{(i-2)2^i+2}$.

The idea that we use in our analysis is probably best described by an example. Consider the hypergraph $\mathcal{G}_1 = \{\{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_4\}\}$ as input and note that $Tr(\mathcal{G}_1) = \{\{v_1, v_2\}, \{v_3, v_4\}\}$. What may have led to the running time claim in [13] is the impression that only subsets of sets that are finally part of the output (minimal transversals in our case) are processed by the algorithm as this is true in other scenarios where the Apriori technique is used. However, consider the set $\{v_1, v_3\}$ in our example. It is an element of the candidate set C_2 in the HBC-algorithm as it hits more edges of $\mathcal{G}_1$ than its one element subsets which all are contained in C_1. But $\{v_1, v_3\}$ is not subset of any minimal transversal. The same holds for $\{v_1, v_4\}$, $\{v_2, v_3\}$, and $\{v_2, v_4\}$. All are candidates in C_2 but not contained in any minimal transversal. As for $\mathcal{G}_1$, the O-notation assures that the claimed bound of [13] holds. However, for large enough i we show that the HBC-algorithm on input $\mathcal{G}_i$ generates too many candidates to run in $O(2^{t(\mathcal{H})}|Tr(\mathcal{H})|)$ time. Thereby, we also show the HBC-algorithm not to be output-polynomial. A lower bound for the number of candidates produced is given in the following lemma.

Lemma 3.2. The number of candidates generated by the HBC-algorithm on input $\mathcal{G}_i$ is at least $2^{(i-2)2^i+2}$ for $i \geq 2$.

Proof:
From Lemma 3.1 we have $|Tr(\mathcal{G}_i \setminus \{e\}) \setminus Tr(\mathcal{G}_i)| \geq 2^{(i-2)2^i+2}$ independent of the choice of e. By fixing any e we note that each element of $Tr(\mathcal{G}_i \setminus \{e\}) \setminus Tr(\mathcal{G}_i)$ is generated as a candidate by the HBC-algorithm. The reasons are the following. Each element of $Tr(\mathcal{G}_i \setminus \{e\}) \setminus Tr(\mathcal{G}_i)$ is a minimal transversal of $\mathcal{G}_i \setminus \{e\}$ and thus the HBC-algorithm would have produced them on input $\mathcal{G}_i \setminus \{e\}$. Hence, each element of $Tr(\mathcal{G}_i \setminus \{e\}) \setminus Tr(\mathcal{G}_i)$ hits more edges of $\mathcal{G}_i \setminus \{e\}$ (and thus of $\mathcal{G}_i$) than all its subsets and all these subsets also are candidates at a lower level. $\square$

In order to show that the bound given in [13] is wrong, we need the following observation on the size of a largest minimal transversal of $\mathcal{G}_i$.

Lemma 3.3. The size $t(\mathcal{G}_i)$ of a largest minimal transversal of $\mathcal{G}_i$ is 2^i.

Proof:
Note that by the definition of $\mathcal{G}_i$ we have $t(\mathcal{G}_i) = 2 \cdot t(\mathcal{G}_{i-1})$ and with the initial condition $t(\mathcal{G}_0) = 1$ we get $t(\mathcal{G}_i) = 2^i$ by iteration. $\square$

We can now prove a superpolynomial lower bound for the HBC-algorithm that also falsifies the originally claimed upper bound on the running time.

Theorem 3.4. The HBC-algorithm is not output-polynomial. Its running time is at least $n^{\Omega(\log\log n)}$, where n denotes the combined size of input and output. Furthermore, the $O(2^{t(\mathcal{H})}|Tr(\mathcal{H})|)$ upper time bound stated in [13] is wrong.

Proof:
We consider the HBC-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 3.1 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 $\gamma(i)$ denote the number of candidates generated by the HBC-algorithm on input $\mathcal{G}_i$. The time, the HBC-algorithm needs to compute $Tr(\mathcal{G}_i)$, is at least the number of candidates generated. With Lemma 3.2 we have $\gamma(i) \geq 2^{(i-2)2^i+1}$ for $i \geq 2$. Thus, to analyze the running time we will show that $\gamma(i)$ 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 $\gamma(i)$ is superpolynomial in m_i, namely $\gamma(i) = m_i^{\Omega(\log\log m_i)}$ which gives the stated lower bound.

To prove the second part, namely that $O(2^{t(\mathcal{H})}|Tr(\mathcal{H})|)$ is not an upper time bound for the HBC-algorithm we shall show that$\gamma(i)$ is superpolynomial in $m'_i = 2^{t(\mathcal{G}_i)}|Tr(\mathcal{G}_i)|$ for large enough i. From Lemmata 3.1 and 3.3 we have $m'_i = 2^{2^i} 2^{2^i-1} = 2^{2\cdot 2^i - 1}$. Hence, it suffices to show that $2^{(i-2)2^i} > (2^{2\cdot 2^i})^c$, for any constant c. This is equivalent to $i-2 > 2c$, for any constant c. Since this obviously holds for large enough i, we have proven that $\gamma(i)$ is superpolynomial in m'_i and hence the $O(2^{t(\mathcal{H})}|Tr(\mathcal{H})|)$ upper time bound claimed in [13] is wrong. $\square$

4. Concluding Remarks

In this short addition to a paper by Hagen [12], we have proven a superpolynomial lower bound for the HBC-algorithm in terms of the combined size of input and output. Thus, like the sequential method, the DL-, the BMR-, and the KS-algorithm, the HBC-algorithm is not output-polynomial. At the same time, the proven lower bound also falsifies the originally claimed upper bound on the HBC-algorithm's running time.

We are not aware of any other nontrivial lower bounds for algorithms generating the transversal hypergraph although we hypothesize 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 [17]. Elbassioni proved a quasi-polynomial upper bound on the running time [6]. But giving a superpolynomial lower bound for the multiplication method requires the construction of new hypergraphs. Takata's hypergraphs $\mathcal{G}_i$ are solved too fast by the multiplication method.

There are also no known nontrivial lower bounds for the Fredman-Khachiyan-Algorithm A and its improved version Algorithm B [8]. Though Gurvich and Khachiyan [9] 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.

References

[1] Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases, *Proceedings of VLDB 1994*.

[2] Bailey, J., Manoukian, T., Ramamohanarao, K.: A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns, *Proceedings of ICDM 2003*.

[3] Berge, C.: *Hypergraphs*, vol. 45 of *North-Holland Mathematical Library*, North-Holland, 1989.

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

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

[6] Elbassioni, K. M.: On the complexity of the multiplication method for monotone CNF/DNF dualization, *Proceedings of ESA 2006*.

[7] Elbassioni, K. M., Hagen, M., Rauf, I.: Some fixed-parameter tractable classes of hypergraph duality and related problems, *Proceedings of IWPEC 2008*.

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

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

[10] Hagen, M.: On the fixed-parameter tractability of the equivalence test of monotone normal forms, *Information Processing Letters*, **103**(4), 2007, 163–167.

[11] Hagen, M.: *Algorithmic and Computational Complexity Issues of* MONET, Ph.D. Thesis, Friedrich-Schiller-Universität Jena, 2008.

[12] Hagen, M.: Lower bounds for three algorithms for transversal hypergraph generation, *Discrete Applied Mathematics*, **157**(7), 2009, 1460–1469.

[13] Hébert, C., Bretto, A., Crémilleux, B.: A data mining formalization to improve hypergraph minimal transversal computation, *Fundamenta Informaticae*, **80**(4), 2007, 415–433.

[14] Johnson, D. S., Papadimitriou, C. H., Yannakakis, M.: On generating all maximal independent sets, *Information Processing Letters*, **27**(3), 1988, 119–123.

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

[16] Papadimitriou, C. H.: NP-Completeness: A Retrospective, *Proceedings of ICALP 1997*.

[17] Takata, K.: A worst-case analysis of the sequential method to list the minimal hitting sets of a hypergraph, *SIAM Journal on Discrete Mathematics*, **21**(4), 2007, 936–946.