Vol.2015-AL-155 No.15
2015/11/21

One-line hack of Knuth's algorithm for minimal hitting set computation with ZDDs

Takeo Imai[1,a)]

Abstract: We show a "one-line hack" of Knuth's algorithm for minimal hitting set computation based on Zero-suppressed binary Decision Diagrams (ZDDs). This modification provides a major performance gain (up to 17.0x in our experimental evaluation), making the algorithm nearly competitive with state-of-the-art algorithms.

1. Introduction

In this paper we study the minimal hitting set problem (a.k.a. the hypergraph transversal problem) that requires enumeration of all minimal hitting sets of a family of finite sets f, denoted as $f^{\#}$. Formally, the problem is to compute $f^{\#} = \{\alpha \mid \beta \in f \text{ implies } \alpha \cap \beta \neq \emptyset\}^{\downarrow}$, where a family of minimal sets $f^{\downarrow} = \{\alpha \in f \mid \beta \in f \text{ and } \alpha \supseteq \beta \text{ implies } \alpha = \beta\}$.

Whilst there have been various algorithms for solving the problem, Knuth suggested a novel one in his book "The art of computer programming" [1] using ZDDs (Zero-suppressed binary Decision Diagrams) [2]. A ZDD is, as shown in Fig. 1, a data structure for representing a family of sets in a space-efficient manner so that it is suitable for many combinatorial problems. Knuth provided a detailed introduction to ZDD in his book, and proposed a novel algorithm for computing minimal hitting set computation as an answer to an exercise.

The performance of the algorithm had been unknown, until Toda evaluated it with experiments [3]. The experiments showed the algorithm performed better than other existing algorithms in many cases, and Toda's own algorithm proposed in [3] is basically better than Knuth's. (So Toda's gave the best performance in many cases.)

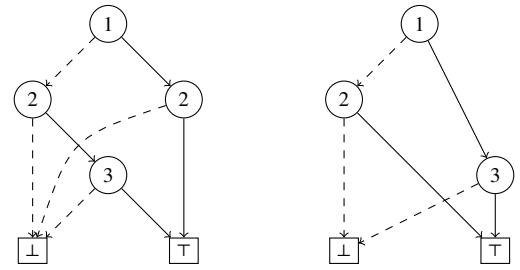
In this paper we present a new algorithm. Our algorithm is a slight modification—actually only one-line replacement ("one-line hack")—of Knuth's algorithm. Despite the small revision, the new algorithm can produce a significant performance gain compared to Knuth's, approaching the performance of Toda's.

2. Zero-suppressed binary decision diagram (ZDD)

A *zero-suppressed binary decision diagram (ZDD)* is a variant of an ordered binary decision diagram (BDD), a graph representation of a family of sets.

Fig. 1 shows examples of ZDDs. A node inside has its index v, and two branches HI and LO; intuitively, v represents the smallest element in the family, and the HI and LO branches represent the residual subfamilies that do and do not contain that root element. In addition, there are two terminal nodes $\bot$ and $\top$, where $\bot$ represents $\emptyset$, and $\top$ represents $\{\emptyset\}$. Another interpretation of a ZDD is that a path from the root to $\top$ represents a set contained in the family, and a path to $\bot$ represents a set not contained in the family.

Every ZDD must be *ordered*, i.e. if a node v_1 points to v_2, there must be an order $v_1 < v_2$. Moreover, in the construction of



(a) $f = \{\{1,2\},\{2,3\}\}$ (b) $f^{\#} = \{\{1,3\},\{2\}\}$

Fig. 1: An example of ZDDs and minimal hitting sets. Each circle represents a node. HI branches are drawn as solid lines, and LO branches are drawn as dashed lines. (a) represents a family f and (b) represents $f^{\#}$, a family of the minimal hitting sets of f.

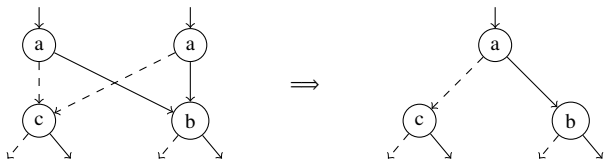


Fig. 2: Node-sharing rule on ZDDs.

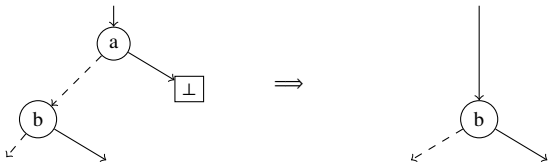


Fig. 3: Node-elimination rule on ZDDs.

[1] Graduate School of Information Science and Technology, the University of Tokyo

[a)] bono@is.s.u-tokyo.ac.jp

Algorithm 1 Knuth's algorithm [1]

```
1: function MINHIT(f)
2:     if f = ⊥ then return ⊤
3:     if f = ⊤ then return ⊥
4:     r ← UNION(f_l, f_h)
5:     r_l ← MINHIT(r)
6:     r ← MINHIT(f_l)
7:     r_h ← NONSUP(r, r_l)
8:     return ZUNIQUE(f_v, r_l, r_h)
9: end function
```

ZDD, every node that represents the same family is maintained to be unique and shared (according to the "node-sharing rule", see Fig. 2). Given the triple of an index v and two nodes l, h, the function ZUNIQUE(v, l, h) returns an existing node associated with the triple (v, l, h), or otherwise a newly created node that has the index v, LO branch l, and HI branch h. In addition, there is "a node elimination rule" (Fig. 3) adopted in the computation of ZUNIQUE: ZUNIQUE$(v, l, \bot) = l$. Hence every ZDD must be *reduced*, i.e. the two rules cannot be applied any more.

3. Knuth's algorithm and the hack

Algorithm 1 is Knuth's algorithm for minimal hitting set computation (from [1], except that statements handling cache are omitted), where f_v is the root node of f, and f_l, f_h are the LO and HI branch of f, respectively. For the new LO branch r_l, it puts $(f_l \cup f_h)^{\#}$. For the new HI branch r_h, it computes $f_l^{\#} \searrow r_l$, where $\searrow$ is the *nonsuperset* operator: $f \searrow g = \{\alpha \in f \mid \beta \in g \text{ implies } \alpha \not\supseteq \beta\}$.

A brief and intuitive interpretation of the code is as follows: when f_v does not hit f (i.e. when the hitting set does not contain f_v), the subsequent nodes need to hit both f_l and f_h, so $r_l = (f_l \cup f_h)^{\#}$. On the other hand, when f_v hits f (i.e. when the hitting set contains f_v), the subsequent nodes have only to hit f_l. The resulting set, however, cannot be a superset of any element of r_l; hence r_h is $f_l^{\#} \searrow r_l$.

This nonsuperset computation (line 7 in Algorithm 1) is an expensive operation. NONSUP is, however, actually redundant here; we found that $f_l^{\#}$ never contains proper supersets of elements of r_l. So the operation can be replaced by *difference* operation DIFF(f, g), or $f \setminus g$, which can be computed in much less time. This replacement is what we call "the one-line hack".

To illustrate our hack has a reliable effect, we show the algorithms of NONSUP and DIFF in Algorithm 2 and Algorithm 3, respectively (both in simple versions). These two algorithms have almost the same structures, but the main differences are at assignments to r_h (where INTSEC(f, g) computes an intersection $f \cap g$, which can be implemented in quite a similar way to DIFF). Although the precise time complexity of NONSUP is unknown, obviously this operation employs more recursive calls than DIFF.

3.1 Correctness of the hack

Here we show the correctness of the hack. What is to be proved is $f_l^{\#} \searrow (f_l \cup f_h)^{\#} = f_l^{\#} \setminus (f_l \cup f_h)^{\#}$, or more generally, $f^{\#} \searrow (f \cup g)^{\#} = f^{\#} \setminus (f \cup g)^{\#}$ for any f and g.

The proof is as follows. Hereinafter, we use symbols f and g for an arbitrary family of sets.

Algorithm 2 Nonsuperset operation

```
1: function NONSUP(f, g)
2:     if g = ⊥ then return f
3:     if f = ⊥ or g = ⊤ or f = g then return ⊥
4:     if f_v > g_v then return NONSUP(f, g_l)
5:     if f_v < g_v then
6:         r_l ← NONSUP(f_l, g), r_h ← NONSUP(f_h, g)
7:     else
8:         r_l ← NONSUP(f_l, g_l),
9:         r_h ← INTSEC(NONSUP(f_h, g_h), NONSUP(f_h, g_l))
10:    return ZUNIQUE(f_v, r_l, r_h)
11: end function
```

Algorithm 3 Difference operation

```
1: function DIFF(f, g)
2:     if g = ⊥ then return f
3:     if f = ⊥ or g = ⊤ or f = g then return ⊥
4:     if f_v > g_v then return DIFF(f, g_l)
5:     if f_v < g_v then
6:         r_l ← DIFF(f_l, g), r_h ← f_h
7:     else
8:         r_l ← DIFF(f_l, g_l),
9:         r_h ← DIFF(f_h, g_h)
10:    return ZUNIQUE(f_v, r_l, r_h)
11: end function
```

Lemma 1. *For all $e \in (f \cup g)^{\#}$, there exists $e' \in f^{\#}$ s.t. $e' \subseteq e$.*

Proof. e minimally hits $f \cup g$, therefore e hits f. This means there is a subset $e' \subseteq e$ that minimally hits f, which should be a member of $f^{\#}$. □

Lemma 2. *For all $e \in (f \cup g)^{\#}$ and $e' \in f^{\#}$, $e' \supseteq e$ implies $e' = e$.*

Proof. From Lemma 1, there exists $e'' \in f^{\#}$ that satisfies $e \supseteq e''$, i.e. $e' \supseteq e \supseteq e''$.

However, because both e' and e'' are members of a family $f^{\#}$ of minimal sets, $e' \supseteq e''$ implies $e' = e''$. So $e' = e$ follows. □

Theorem 1. $f^{\#} \searrow (f \cup g)^{\#} = f^{\#} \setminus (f \cup g)^{\#}$.

Proof. From the definition, $f^{\#} \searrow (f \cup g)^{\#} = f^{\#} \setminus (f^{\#} \nwarrow (f \cup g)^{\#})$ where $\nwarrow$ is the *supersets* operator: $f \nwarrow g = \{e \in f \mid e \supseteq e' \text{ for some } e' \in g\}$.

From Lemma 2,

$$f^{\#} \nwarrow (f \cup g)^{\#} \subseteq \{e \in f^{\#} \mid e = e' \text{ for some } e' \in (f \cup g)^{\#}\}$$

$$= f^{\#} \cap (f \cup g)^{\#}$$

and trivially $f^{\#} \nwarrow (f \cup g)^{\#} \supseteq f^{\#} \cap (f \cup g)^{\#}$, which implies:

$$f^{\#} \nwarrow (f \cup g)^{\#} = f^{\#} \cap (f \cup g)^{\#}$$

Therefore,

$$f^{\#} \searrow (f \cup g)^{\#} = f^{\#} \setminus (f^{\#} \cap (f \cup g)^{\#}) = f^{\#} \setminus (f \cup g)^{\#}$$

from set operations. □

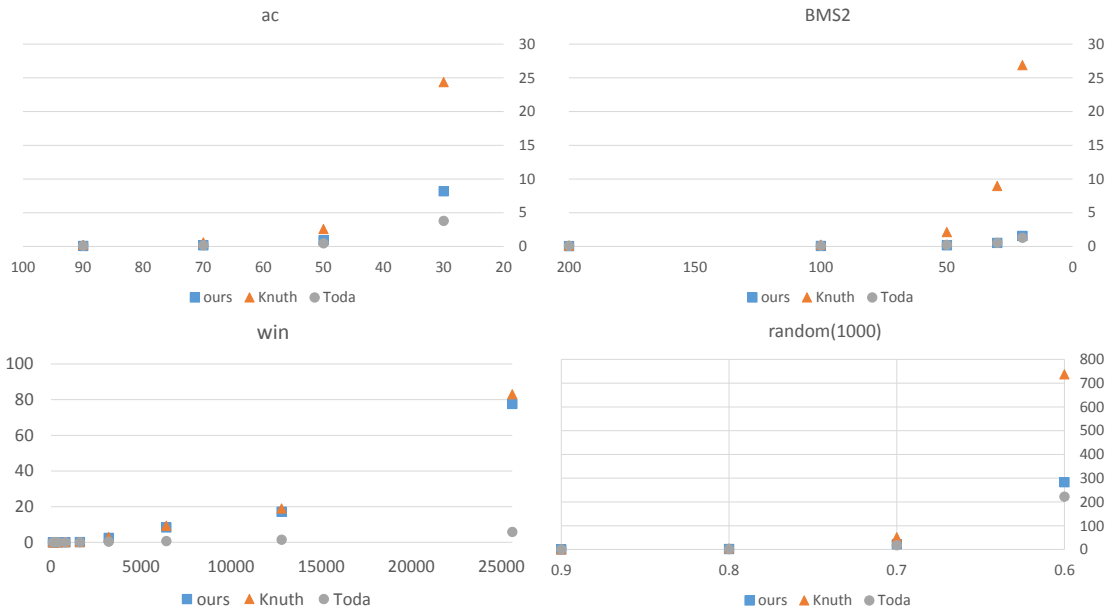


Fig. 4: Comparisons of running time: square/triangular/circular dots represent our/Knuth's/Toda's algorithm, respectively. All horizontal axes show times [s].

4. Experiment

4.1 Setup

For an evaluation of our algorithm, we used an implementation distributed in [4] that implements Knuth's and Toda's algorithms. We did literally a one-line hack to Knuth's part in the implementation, and then we performed experiments on data that can be obtained from [5].

The instances we used are classified into the following four types:

(1) accidents (**ac**): the complement of the set of maximal frequent itemsets with support threshold $n \times 10^3$, where $n \in \{30, 50, 70, 90, 110, 130, 150, 200\}$.

(2) BMS-Web-View-2 (**BMS2**): constructed in the same way as **ac**, where $n \in \{20, 30, 50, 100, 200\}$.

(3) Connect4-win (**win**): a hypergraph with n hyperedges corresponding to the set of minimal winning stages of a board game "connect-4", where $n \in \{100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600\}$.

(4) Uniform random(**random(1000)**): random instances; each element is included in a set in the probability $n/10$ ($n \in \{6, 7, 8, 9\}$). The number of sets is 1000.

The entire execution was carried out on Ubuntu Server 14.04 LTS, Intel Xeon E5-2670 v2 CPU with 30GB RAM.

4.2 Result and Discussion

Fig. 4 shows the comparison result. Our hack provided a significant speedup in **ac** (– 3.0x), **BMS2** (– 17.0x), and **random** (– 2.6x) compared with Knuth's original algorithm. Our algorithm was still slower than Toda's, but nearly competitive.

On the other hand, there was only a slight performance gain in **win** (–1.2x). We suppose the difference in the results stems from the fact that **win** had much smaller sets than the other three datasets did; all the sets in **win** had only 8 or 9 elements, whereas those in **BMS2** had more than 1000, for example. The characteristic of **win** made the input ZDD lopsided to LO–branch side, in which nearly every f_l was almost equal to $f_l \cup f_h$. This may result in fewer assignments to r_h in NONSUP/DIFF in MINHIT, resulting in little difference between NONSUP and DIFF.

5. Summary

In this paper, we proposed a new algorithm for the minimal hitting set computation. The algorithm is a variant—actually an only one-line modification ("one line hack")—of Knuth's [1]. We have presented a proof that shows the correctness of the hack. Moreover, in our experimental evaluation, the hack provided huge performance gains (up to 17.0x) so that it became nearly competitive to Toda's algorithm [3], one of the fastest in state-of-the-art algorithms.

References

[1] Knuth, D. E.: *The Art of Computer Programming vol.4A, Combinatorial Algorithms: Part 1*, Addison-Wesley Professional (2011).

[2] Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, *Proceedings of the 30th International Design Automation Conference*, pp. 272–277 (1993).

[3] Toda, T.: Hypergraph transversal computation with binary decision diagrams, *Experimental Algorithms*, pp. 91–102 (2013).

[4] Toda, T.: HTC-BDD: Hypergraph Transversal Computation with Binary Decision Diagrams, `http://www.sd.is.uec.ac.jp/toda/htcbdd.html`.

[5] Murakami, K. and Uno, T.: hypergraph dualization repository, `http://research.nii.ac.jp/~uno/dualization.html`.