Experimental comparison of the two Fredman-Khachiyan-algorithms

Matthias Hagen[*]   Peter Horatschek[†]   Martin Mundhenk[†]

Abstract

We experimentally compare the two algorithms A and B by Fredman and Khachiyan [FK96] for the problem MONET—given two monotone Boolean formulas φ in DNF and ψ in CNF, decide whether they are equivalent. Currently, algorithm B is the MONET algorithm with the best known worst-case performance. However, there is no experimental evaluation of its practical performance yet, mainly due to the following two reasons. Firstly, implementation of algorithm B is usually considered to be more involved than for algorithm A. Secondly and probably more importantly, there is the assumption that the operations performed by algorithm B to ensure recursion on smaller sub-problems do only pay off theoretically.

In this paper, we contrast this assumption by experimentally showing algorithm B to be competitive and even superior to algorithm A on many instances.

1   Introduction

The problem MONET—MO(notone) N(ormal form) E(quivalence) T(est)—asks for the equivalence of two monotone (a synonym for positive) Boolean formulas φ in DNF and ψ in CNF. Algorithms solving MONET can be easily transformed to solve the computational variant MONET'—given a monotone DNF, compute the equivalent CNF—and vice versa. Hence, MONET and MONET' are equivalent in the sense of solvability in appropriate terms of polynomial time [BI95]. Furthermore, MONET' is equivalent to the problems DUALIZATION of monotone CNFs and TRANSVERSAL HYPERGRAPH GENERATION. This means that any MONET or MONET' algorithm is applicable to many fundamental problems 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], distributed systems [GB85], and graph theory [JPY88, LLK80]. See [Hag08, Chapter 3] for a more detailed list of equivalent problems and possible applications. The currently best known MONET algorithms run in quasi-polynomial $n^{o(\log n)}$ time or use $O(\log^2 n)$ nondeterministic bits [EGM03, FK96, KS03]. Thus, on the one hand, MONET is probably not coNP-complete, but on the other hand a polynomial time algorithm is not yet known. This situation turns MONET into one of the very few problems "between" P and NP- resp. coNP-hard—the other famous such problem being GRAPH ISOMORPHISM. Actually, there are polynomial time algorithms for many special MONET classes— e.g. when φ is a k-DNF, 2-monotonic, μ-equivalent, or acyclic [BHIK97, Eit94, EG95]—but the exact complexity of the general problem MONET is a long standing famous open question [Pap97].

As for evaluating the practical performance, there have been several experimental studies on known algorithms for MONET or equivalent problems [BMR03, DL05, KBEG06, KS05, LJ03, TT02, US03]. Unfortunately, all have some lack of coverage. None of the published studies include the algorithm B by Fredman and Khachiyan [FK96], the MONET algorithm with the best known worst-case performance. Actually, the usual assumption in the literature is that the operations performed by algorithm B to ensure recursion on smaller sub-problems compared to the less involved algorithm A just complicate the implementation and do only pay off theoretically [BMR03, KBEG06]. Thus, the practical performance of algorithm B has not been systematically examined. Hence, it is not clear at all, which of the currently known algorithms is the best choice on which kind of instances.

In this paper we start working on closing this gap. Contrasting the somehow folklore assumption, we experimentally show algorithm B to be competitive and even superior to algorithm A on many instances.

2   Preliminaries

Two Boolean formulas are *equivalent* if they have the same truth table. *Monotone formulas* are Boolean formulas with $\wedge$ and $\vee$ as only connectives. No negation signs are allowed. A monomial (resp. clause) is the conjunction (disjunction) of variables. We often refer to monomials or clauses simply as *terms*. A monotone

---

[*]Institute of Computer Science, Friedrich Schiller University Jena, D–07737 Jena, Germany; Research Group Programming Languages / Methodologies, University Kassel, D–34121 Kassel, Germany; Research Group Web Technology & Information Systems, Bauhaus University Weimar, D–99423 Weimar, Germany, matthias.hagen@uni-weimar.de

[†]Institute of Computer Science, Friedrich Schiller University Jena, D–07737 Jena, Germany, {hopet|mundhenk}@cs.uni-jena.de

DNF (resp. *CNF*) is the disjunction (conjunction) of monomials (clauses). A monotone DNF or CNF α is said to be *irredundant* if there are no two terms in α such that one is contained in the other. The irredundant DNF and CNF of monotone formulas are unique [Qui53] and can be obtained from respective redundant normal forms in quadratic time (by deleting the superset terms). Since terms that contain other terms are "useless" in case of equivalence testing (absorption rule!), we only concentrate on irredundant inputs yielding the following formal definition.

> MONET
> instance: irredundant, monotone DNF φ and CNF ψ
> question: are φ and ψ equivalent?

The *size* of the MONET-instance (φ, ψ) is the number of variable occurrences in φ and ψ. An *assignment* for φ and ψ is a subset $\mathcal{A} \subseteq V$, where V is the set of variables of φ and ψ. We write $\mathcal{A}(\varphi)$ for the evaluation of formula φ with respect to the assignment $\mathcal{A}$. Thereby, the notion is that variable x is set to `true` iff $x \in \mathcal{A}$. This means that the powerset $\mathcal{P}(V)$ can also be seen as the set of all assignments for φ and ψ. In the same way we consider the terms of φ and ψ to be sets of variables. Hence, they can also be viewed as assignments.

Note that throughout the paper φ always denotes a monotone DNF and ψ always denotes a monotone CNF.

3 The FK-algorithms

In 1996 Fredman and Khachiyan developed two MONET algorithms, FK-algorithm A and the improved version FK-algorithm B [FK96]. Both algorithms exploit the self-reducibility of MONET. Namely, φ and ψ are equivalent iff setting any variable x to `false` resp. `true` yields two respectively equivalent DNF/CNF-pairs. In case of non-equivalence, both FK-algorithms return an assignment $\mathcal{A}$ with $\mathcal{A}(\varphi) \neq \mathcal{A}(\psi)$ as a witness for non-equivalence. Furthermore, both algorithms work recursively, but before exploiting the self-reducibility they check some basic conditions that can easily guarantee non-equivalence in case of monotone normal forms.

3.1 Preconditions Let (φ, ψ) be a pair of an irredundant, monotone DNF φ and an irredundant, monotone CNF ψ that are equivalent. Then the following three conditions hold. Firstly,

(3.1)   any term of φ and any term of ψ intersect.

Assume there exists a monomial $m \in \varphi$ and a clause $c \in \psi$ with $m \cap c = \emptyset$. Now consider the assignment $\mathcal{A} = m$ and note that $\mathcal{A}(\varphi) = 1$ and $\mathcal{A}(\psi) = 0$.

Secondly,

(3.2) φ and ψ contain exactly the same variables.

Assume that there is a variable x in φ that is not present in ψ (the argumentation is similar if ψ contains a "new" variable). Let m be a monomial of φ containing x and consider the assignment $\mathcal{A} = m \setminus \{x\}$. We have $\mathcal{A}(\varphi) = 0$ and $\mathcal{A}(\psi) = 1$ as $\mathcal{A}$ has a non-empty intersection with every clause of ψ due to condition (3.1).

Thirdly, denote by $|\varphi|$ and $|\psi|$ the number of terms of φ and ψ, and let $\max(\varphi)$ and $\max(\psi)$ be the size of a largest term in φ resp. ψ, then

(3.3) $\max(\varphi) \leq |\psi|, \quad \max(\psi) \leq |\varphi|.$

Assume that there is a monomial $m \in \varphi$ that contains more variables than clauses are contained in ψ (the argumentation is similar if ψ contains a clause that is "too large"). Now let $m' \subset m$ be a proper subset of m satisfying $m' \cap c \neq \emptyset$ for any clause c of ψ. Consider the assignment $\mathcal{A} = m'$ and note that $\mathcal{A}(\varphi) = 0$ and $\mathcal{A}(\psi) = 1$.

Conditions (3.1)–(3.3) can be easily tested in linear resp. quadratic time which is done by both FK-algorithms as a preprocessing step. We now come to the description of FK-algorithm A. In Section 3.3 we then discuss the improvements of FK-algorithm B.

3.2 FK-algorithm A FK-algorithm A uses an additional precondition that holds for any equivalent (φ, ψ) pair. For the number v of variables we have

$$(3.4) \qquad \sum_{m \in \varphi} 2^{v-|m|} + \sum_{c \in \psi} 2^{v-|c|} \geq 2^v.$$

The left hand side of condition (3.4) sums up the assignments that satisfy φ and the assignments that do not satisfy ψ. Hence, if the left hand side of condition (3.4) is smaller than 2^v, there must be an assignment $\mathcal{A}^*$ that does not satisfy φ but ψ. Such an assignment $\mathcal{A}^*$ can be found iteratively as follows. Start with the empty assignment and at step i include variable x_i iff $\ell(\mathcal{A}_{i-1}^* \cup \{x_i\}) \leq \ell(\mathcal{A}_{i-1}^*)$, where $\mathcal{A}_{i-1}^*$ is the partial result from step $i-1$ and $\ell(\mathcal{A})$ gives the number of monomials of φ satisfied by $\mathcal{A}$ plus the the number of clauses of ψ not satisfied by $\mathcal{A}$. Hence, $\mathcal{A}_i^*$ is computed in a way as to minimize the value of ℓ where ℓ is similar to the left hand side of condition (3.4). Fredman and Khachiyan [FK96] mainly include condition (3.4) in their algorithm A to ensure the existence of a sufficiently frequent variable that then guarantees the validity of estimations made in their worst-case analysis.

A pseudocode listing of FK-algorithm A is given as Algorithm 1. We give some further brief remarks. As

for the initial call of FK-A, the global variable $\mathcal{A}$ is the empty set. Note that we have to check irredundancy of the input as this property of the original input might get lost during the recursion process. We already discussed how appropriate assignments are found in case of violation of conditions (3.1)–(3.4).

In case of small inputs consisting of at most one term each (line 4 of FK-algorithm A), there are only very few possibilities. If there are no variables at all, φ is the empty DNF (which is unsatisfiable) or contains the empty monomial (which is valid). The equivalent CNF of the empty DNF contains the empty clause only, and the equivalent CNF of the empty monomial is the empty CNF. Hence, if the inputs are not equivalent but contain no variables, any assignment serves as a witness. If the formulas contain variables but only one term each, the previous check of condition (3.2) already ensures equivalence as then the formulas must be identical!

As for the recursive process, the FK-algorithm A decomposes the original input instance (φ, ψ) as follows. It selects a "splitting" variable x that appears with frequency at least $1/\log(|\varphi| + |\psi|)$ in either φ or ψ. The existence of such a variable is ensured by condition (3.4) [FK96]. Then φ and ψ can be rewritten as

$$\begin{aligned} \varphi &\equiv (x \wedge \varphi_0) \vee \varphi_1, \\ \psi &\equiv (x \vee \psi_0) \wedge \psi_1, \end{aligned}$$

where φ_1 (resp. ψ_1) contains the monomials (resp. clauses) of φ (resp. ψ) that do not contain x and φ_0 (resp. ψ_0) are the other monomials (resp. clauses) from which x was excluded. Now deciding equivalence of (φ, ψ) is equivalent to deciding equivalence of the two smaller problems

$$(3.5) \qquad (\varphi_1, \psi_0 \wedge \psi_1) \quad \text{and}$$
$$(3.6) \qquad (\varphi_0 \vee \varphi_1, \psi_1).$$

Note that (3.5) corresponds to setting x to `false` in the original instance (φ, ψ) whereas (3.6) corresponds to setting x to `true`. Hence, if the call on subproblem (3.5) returns an assignment showing non-equivalence, the original call can return exactly this assignment (remember the set notion of assignments). In case that the second subproblem (3.6) is not equivalent, the algorithm adds the splitting variable x to the assignment.

Fredman and Khachiyan showed the following worst-case performance.

PROPOSITION 3.1. ([FK96]) *FK-algorithm A runs in time* $n^{O(\log^2 n)}$.

The main tool in their analysis of FK-algorithm A is to base the estimation of the number of recursive calls on

---

Algorithm 1 The FK-algorithm A (FK-A)

Input: irredundant, monotone DNF φ and CNF ψ
Output: $\emptyset$ in case of equivalence; otherwise, assignment $\mathcal{A}$ with $\mathcal{A}(\varphi) \neq \mathcal{A}(\psi)$

1: make φ and ψ irredundant
2: **if** one of conditions (3.1)–(3.4) is violated **then**
3:     **return** appropriate $\mathcal{A}$
4: **if** $|\varphi| \cdot |\psi| \leq 1$ **then**
5:     **return** appropriate $\mathcal{A}$ found by a trivial check
6: find a variable x appearing with frequency
                      $\geq 1/\log(|\varphi| + |\psi|)$ in either φ or ψ
7: $\mathcal{A} \leftarrow$ FK-A$(\varphi_1, \psi_0 \wedge \psi_1)$
8: **if** $\mathcal{A} = \emptyset$ **then**
9:     $\mathcal{A} \leftarrow$ FK-A$(\varphi_0 \vee \varphi_1, \psi_1)$
10:     **if** $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A} \cup \{x\}$
11: **return** $\mathcal{A}$

---

the fact that for equivalent normal forms condition (3.4) holds and hence the splitting variable is sufficiently frequent.

As for the practical performance, an experimental study of a randomized version of FK-algorithm A showed it to be quite efficient [KBEG06]. Furthermore, there is also a version by Tamaki designed to run with polynomial space [Tam00].

3.3 FK-algorithm B FK-algorithm A does not exploit the fact that the second recursive call is only performed if the first call did not yield a witness for non-equivalence, but FK-algorithm B does. Assume that the input (3.5) of the first recursive call is an equivalent pair. Now in the second call, we try to find an assignment $\mathcal{A}$ with $\mathcal{A}(\varphi_0 \vee \varphi_1) \neq \mathcal{A}(\psi_1)$ (a witness for the non-equivalence of the second pair (3.6)). As φ_1 is equivalent to $\psi_0 \wedge \psi_1$ this then gives $\mathcal{A}(\varphi_0) \vee \mathcal{A}(\psi_0 \wedge \psi_1) \neq \mathcal{A}(\psi_1)$. If now $\mathcal{A}(\psi_0) = 1$, we have $\mathcal{A}(\varphi_0) \vee \mathcal{A}(\psi_1) \neq \mathcal{A}(\psi_1)$, which implies $\mathcal{A}(\psi_1) = 1$ and $\mathcal{A}(\varphi_0)$. However, this is a contradiction to condition (3.1). Hence, actually, for the second recursive call on (3.6) it suffices to find an assignment $\mathcal{A}$ with $\mathcal{A}(\psi_0) = 0$ and $\mathcal{A}(\psi_1) \neq \mathcal{A}(\varphi_0)$. As for $\mathcal{A}(\psi_0) = 0$, note that we only have to check the maximal assignments (with respect to set inclusion) not satisfying ψ_0, of which there are exactly $|\psi_0|$ (for each clause $c \in \psi_0$ the assignment that does not contain exactly the variables of c). Hence, for each clause c of ψ_0, FK-algorithm B is recursively called on an adjusted pair (φ_0^c, ψ_1^c), where the superscript c denotes that all variables from c are set to `false` in the respective formula.

Note that in case we started testing equivalence of φ and ψ by first examining the second pair (3.6), an analogous argumentation yields that the second call

Algorithm 2 The FK-algorithm B (FK-B)
___

Input: irredundant, monotone DNF φ and CNF ψ
Output: $\emptyset$ in case of equivalence; otherwise, assignment $\mathcal{A}$ with $\mathcal{A}(\varphi) \neq \mathcal{A}(\psi)$
1: make φ and ψ irredundant; $\quad \nu = |\varphi| \cdot |\psi|$;
2: **if** one of conditions (3.1)–(3.3) is violated **then**
3: $\quad$ **return** appropriate $\mathcal{A}$
4: **if** $\min\{|\varphi|, |\psi|\} \leq 2$ **then**
5: $\quad$ **return** appropriate $\mathcal{A}$ found by a trivial check
6: choose some variable x from the formulas
7: $\varepsilon(\nu) \leftarrow 1/\chi(\nu)$
8: $\varepsilon_x^\varphi \leftarrow |\{m \in \varphi : x \in m\}|/|\varphi|$
9: $\varepsilon_x^\psi \leftarrow |\{c \in \psi : x \in c\}|/|\psi|$;
10: **if** $\varepsilon_x^\varphi \leq \varepsilon(\nu)$ **then**
11: $\quad \mathcal{A} \leftarrow$ FK-B$(\varphi_1, \psi_0 \wedge \psi_1)$
12: $\quad$ **if** $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A}$
13: $\quad$ **for all** clauses $c \in \psi_0$ **do**
14: $\quad\quad \mathcal{A} \leftarrow$ FK-B(φ_0^c, ψ_1^c)
15: $\quad\quad$ **if** $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A} \cup \{x\}$
16: **else if** $\varepsilon_x^\psi \leq \varepsilon(\nu)$ **then**
17: $\quad \mathcal{A} \leftarrow$ FK-B$(\varphi_0 \vee \varphi_1, \psi_1)$
18: $\quad$ **if** $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A} \cup \{x\}$
19: $\quad$ **for all** monomials $m \in \varphi_0$ **do**
20: $\quad\quad \mathcal{A} \leftarrow$ FK-B(φ_1^m, ψ_0^m)
21: $\quad\quad$ **if** $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A} \cup m$
22: **else**
23: $\quad \mathcal{A} \leftarrow$ FK-B$(\varphi_1, \psi_0 \wedge \psi_1)$
24: $\quad$ **if** $\mathcal{A} = \emptyset$ **then**
25: $\quad\quad \mathcal{A} \leftarrow$ FK-B$(\varphi_0 \vee \varphi_1, \psi_1)$
26: $\quad\quad$ **if** $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A} \cup \{x\}$
27: **return** $\mathcal{A}$

___

then is equivalent to finding an assignment $\mathcal{A}$ with $\mathcal{A}(\varphi_0) = 1$ and $\mathcal{A}(\varphi_1) \neq \mathcal{A}(\psi_0)$. Hence, for each monomial m of φ_0, FK-algorithm B is recursively called on an adjusted pair (φ_1^m, ψ_0^m), where the superscript m in this case denotes that all variables from m are set to true in the respective formula. Note that in case of non-equivalence we now also have to include the corresponding monomial m in the respective witness (remember our set notion of assignments).

A pseudocode listing of FK-algorithm B is given as Algorithm 2. The algorithm exploits the above described decomposition of the second recursive call whenever useful. The decision, if it is useful and which of the two recursive calls is performed first, is done according to the frequency of the "splitting" variable x (chosen in line 6). Therefore, in line 7 the algorithm computes a "threshold" frequency $\varepsilon(\nu) = 1/\chi(\nu)$, where $\nu = |\varphi| \cdot |\psi|$ is the *volume* of φ and ψ and χ is the function defined by $\chi(n)^{\chi(n)} = n$. Note that

$\chi(n) \sim \log n / \log \log n = o(\log n)$.

If the frequency of the splitting variable in φ is less than $\varepsilon(\nu)$, the FK-algorithm B uses (3.5) as input of the first recursive call and uses the more sophisticated version of the second call to solve (3.6). If elsewise the frequency of the splitting variable in ψ is less than $\varepsilon(\nu)$, the FK-algorithm B uses (3.6) as input of the first recursive call and then solves (3.5) using the improved decomposition. If otherwise the splitting variable is more frequent than $\varepsilon(\nu)$ in both, φ and ψ, the FK-algorithm B just branches as FK-algorithm A. Note that condition (3.4) is not necessary any more as we do not have to guarantee a sufficiently frequent splitting variable.

As for the easy cases in line 4, note that if $\min\{|\varphi|, |\psi|\} \leq 1$ we have a similar argumentation as for FK-algorithm A. If the formulas are not empty, one contains just one term and the other then has to contain singleton terms for each variable. If not, it is easy to give a witness for non-equivalence according to the situation. Similarly, if the minimum is 2, a brute force multiplication of the two terms and a following comparison to the other normal form is sufficiently efficient.

As for the worst-case analysis, Fredman and Khachiyan give a better upper bound on the runtime than for FK-algorithm A.

PROPOSITION 3.2. ([FK96]) *FK-algorithm B runs in time $n^{o(\log n)}$.*

The main tool in the analysis is that the new branching guarantees better bounds on the size of the inputs of recursive calls than in the analysis of FK-algorithm A.

As for the practical performance, none of the so far published experimental studies of MONET algorithms include FK-algorithm B as the assumption usually is that, despite the theoretically worse runtime, FK-algorithm A will perform better than FK-algorithm B in experiments [BMR03, KBEG06]. However, we will show in Section 5 that this assumption has to be adjusted, as in fact FK-algorithm B turns out to be competitive in practical experimentation.

4 A few implementation details

The algorithms were implemented using Java. We decided to represent variable sets—like terms and assignments—as *bitmaps* which is just a sequence of bits where bit i is set iff x_i is contained in the corresponding term. This allows us to process operations on formulas as logical operations on bitmaps, which can be performed very fast if using some Java predefined data type. Hence, our choice for internally representing bitmaps is the `long` data type—one of the primitive Java data types. Note that we can use only 63 of the

64 bits of a `long` variable (the remaining bit being the reserved sign bit). This restricts us to formulas with at most 63 variables. Hence, we compared several other possibilities of representing bitmaps. Namely, we tried using `BigInteger`, `BitSet`, and an own structure composed of an array of sufficiently many `long`'s. Somehow surprisingly, the best overall performance for formulas with more than 63 variables was achieved by our own array of `long`'s structure that beats the Java proprietary data types in our pretests. However, not that surprisingly, for formulas with less than 63 variables, a single `long` turns out to be the best choice.

Now that we know how to represent variable sets, we still have to internally represent complete normal forms. In our pretests we compared implementations of the FK-algorithms using the classes `Array` and `Vector` to store a set of bitmaps. An advantage of `Array` is the faster access compared to `Vector`, whereas `Vector` might have advantages in the process of making inputs irredundant as in an `Array` implementation we have to manually close "gaps" in the array due to redundant terms. However, our pretests favored the `Array` implementation.

Hence, in our implementations a formula is stored as an `Array` of bitmaps—that itself are stored as `long` resp. `Array` of `long` according to the number of variables.

Both FK-algorithms use an irredundancy procedure in line 1 (as in recursive calls irredundancy of the original inputs might get lost). But note that when making the DNF/CNF of a recursive call irredundant not the whole DNF/CNF have to be considered as the terms that included the splitting variable cannot be redundant in the resulting formulas. Redundancy can only appear in the formulas $\varphi_0 \vee \varphi_1$ and $\psi_0 \wedge \psi_1$ and there only terms in φ_1 and ψ_1 have to be checked for redundancy. Hence, in our implementations, the process of making formulas irredundant is always carried out *before* a recursive call. This significantly speeds up computation. Note that in case of FK-algorithm B we can analogously save some processing time when making the inputs of the many calls replacing the second call irredundant. If we set the variables of a monomial m of φ_0 to `true` only monomials of φ_1^m may be redundant. Analogously, in case of setting the variables of a clause c of ψ_0 to `false` only clauses of ψ_1^c may be redundant.

In our implementations we also slightly adopt the choice of the splitting variable to speed up computation but not affecting the theoretical runtime guarantees. As for FK-algorithm A we do not check which variables are sufficiently frequent but choose a variable with the highest frequency in either φ or ψ. As for FK-algorithm B we always choose a variable with the smallest frequency in either φ or ψ to reach the improved branching when-

ever possible. Furthermore, we do not really compute the "threshold" frequency as there is no closed form for the function χ. Hence, instead of computing $\varepsilon(\nu)$ via χ, we compute the frequencies of our splitting variable x_i (defined in lines of Algorithm 2) and set $y_\varphi = 1/\varepsilon_i^\varphi$. When we now have to check whether $\varepsilon_i^\varphi \leq \varepsilon(\nu)$ we instead perform the equivalent check $y_\varphi^{y_\varphi} \geq \nu$ that can be implemented more easily. An analogous check is performed for ε_i^ψ.

5 Experimental results

To ensure comparability, we use test instances that were also used in previous studies [KBEG06, KS05]. We only slightly changed the known test bed in the sense that we added some additional instances not used before. Namely, we have so-called DTH instances that we derive by a role exchange from the known TH instances. Furthermore, we use a class of instances that are "hard" for several other MONET algorithms in the sense of theoretical lower bound analysis [Tak07, Hag07]. The DNFs of our test instances are defined as follows (equivalent CNFs were previously computed by a brute force multiplication using the DL-algorithm [DL05] if necessary):

Matching (M(v)): v variables (v is even) $x_1, \ldots, x_v$ and the monomial set $\{\{x_{i-1}, x_i\} : 2 \leq i \leq v, i \text{ is even}\}$ (in a graph this would form an induced matching). Hence, the DNF has $v/2$ monomials and the CNF has $2^{v/2}$ clauses.

Dual Matching (DM(v)): roles of DNF and CNF of the respective M(v) instance are exchanged. Hence, the CNF is very small.

Threshold (TH(v)): v variables (v is even) $x_1, \ldots, x_v$ and the monomial set $\{\{x_i, x_j\} : 1 \leq i < j \leq v, j \text{ is even}\}$. This yields $v^2/4$ monomials and $v/2 + 1$ clauses.

Dual Threshold (DTH(v)): roles of DNF and CNF of the respective TH(v) instance are exchanged.

Self-Dual Threshold (SDTH(v)): the monomial set of SDTH(v) is obtained from the TH and DTH instances as follows: $\{\{x_{v-1}, x_v\}\} \cup \{\{x_{v-1}\} \cup m : m \in \text{TH}(v-2)\} \cup \{\{x_v\} \cup m : m \in \text{DTH}(v-2)\}$. The effect is that the equivalent CNF has the same set of terms. The number of terms is $(v-2)^2/4 + v/2 + 1$.

Self-Dual Fano-Plane (SDFP(v)): the DNF contains v variables and $(k-2)^2/4 + k/2 + 1$ monomials, where $k = (v-2)/7$. The construction starts with the DNF φ_0 that contains the monomials $\{x_1, x_2, x_3\}, \{x_1, x_5, x_6\}, \{x_1, x_7, x_4\}, \{x_2, x_4, x_5\},$

		$v = 20$	$v = 24$	$v = 28$	$v = 30$	$v = 32$	$v = 34$	$v = 36$	$v = 38$	$v = 40$
M	FK-A	0.94	2.25	14.14	45.28	108.58	264.68	677.69	1888.78	5396.90
	FK-B	0.54	1.36	6.25	19.94	68.43	245.36	944.16	3538.28	15107.78
		$v = 20$	$v = 24$	$v = 28$	$v = 30$	$v = 32$	$v = 34$	$v = 36$	$v = 38$	$v = 40$
DM	FK-A	0.88	3.78	19.36	45.10	107.20	267.32	684.25	1970.00	5962.20
	FK-B	0.53	1.45	6.46	19.67	69.70	256.71	905.17	3575.83	14130.80
		$v = 40$	$v = 60$	$v = 80$	$v = 100$	$v = 120$	$v = 140$	$v = 160$	$v = 180$	$v = 200$
TH	FK-A	0.68	1.73	1.47	1.93	2.89	4.52	6.85	10.64	16.79
	FK-B	0.04	0.30	0.51	0.61	0.94	0.90	1.19	1.71	2.00
		$v = 40$	$v = 60$	$v = 80$	$v = 100$	$v = 120$	$v = 140$	$v = 160$	$v = 180$	$v = 200$
DTH	FK-A	0.48	1.29	1.48	2.09	3.06	4.66	6.67	10.23	13.92
	FK-B	0.04	0.30	0.48	0.65	0.96	0.93	1.47	1.56	2.04
		$v = 42$	$v = 62$	$v = 82$	$v = 102$	$v = 122$	$v = 142$	$v = 162$	$v = 182$	$v = 202$
SDTH	FK-A	1.28	1.39	1.72	3.06	5.13	9.24	16.84	27.08	41.33
	FK-B	0.30	0.51	0.83	1.16	1.41	2.49	4.81	7.35	10.81
		$v = 16$		$v = 23$		$v = 30$		$v = 37$		
SDFP	FK-A	0.47		2.42		15.40		801.17		
	FK-B	0.10		1.44		6.52		113.68		

Table 1: Performance of the FK-algorithms. Runtime in seconds.

$\{x_2, x_6, x_7\}$, $\{x_3, x_4, x_6\}$, and $\{x_3, x_5, x_7\}$, representing the set of lines in a Fano plane. Now let $\varphi = \varphi_1 \vee \cdots \vee \varphi_k$, where $\varphi_1, \ldots, \varphi_k$ are k disjoint copies of φ_0. Furthermore, let ψ be the equivalent CNF of φ; its 7^k clauses are obtained by taking one monomial from each of the k copies of φ_0. We obtain the monomial set of SDFP(v) as $\{\{x_{v-1}, x_v\}\} \cup \{\{x_{v-1}\} \cup m : m \in \varphi\} \cup \{\{x_v\} \cup c : c \in \psi\}$.

Takata: these DNFs are "hard" for several MONET algorithms and are used in proving lower bounds [Tak07, Hag07]. The starting point is $\varphi_1 = x_1$. The DNF φ_i is then obtained by multiplying out $\varphi_i = (\alpha \vee \beta) \wedge (\gamma \vee \delta)$, where α, β, γ, and δ are disjoint copies of φ_{i-1}. This gives $2^{2(2^i-1)}$ monomials in the DNF and 2^{2^i-1} clauses in the equivalent CNF.

The experimentation was done on an AMD Athlon 64 3700+ with 2,2 GHz and 1GB RAM running a Debian/GNU Linux 4.0 with kernel version 2.6.18. As for compiling and interpreting the bytecode we used the JDK and JRE by Sun in version 1.5.0.10 in the 64 bit variant. Table 1 summarizes our experimental results on equivalent input instances. In the table, we show the total CPU time, in seconds. Times are normalized over five runs for each instance. Furthermore, Figures 1 to 4 graphically show our results on several of the test instance classes. Note the the time axes are scaled logarithmically.

In Table 1 there are no results for the Takata instances or for non-equivalent inputs. As for the Takata instances, the reason is their exponential growing term set. The Takata DNFs φ_1 or φ_2 and their equivalent CNFs are just too small to give meaningful runtimes. Furthermore, storing φ_4 would require more than 1GB so that we decided to only test φ_3 and its equivalent CNF. This instance is solved by FK-algorithm B in 753.62 seconds whereas FK-algorithm A did not finish within 5 hours (18,000 seconds).

As for non-equivalent inputs, we tested both FK-algorithms on non-equivalent inputs that we consider to be "hardest". Namely, leaving out just one term of the DNF or CNF results in "nearly" equivalent instances. Consequently, the runtimes of our implementations then are just a little faster than the ones we report for the respective equivalent inputs. Not surprisingly, leaving out more terms or using some completely different CNFs speeds up computation as then larger and larger parts of the recursion tree are not traversed. Hence, the runtimes in Table 1 are somehow the "worst" for the respective instance classes with the FK-algorithms traversing the whole recursion tree.

6 Conclusion

Comparing our results for FK-algorithm A and FK-algorithm B, we can conclude that FK-algorithm B is competitive on all classes, except for large Matching or Dual Matching instances. What exactly happens on these instances is an interesting issue to be addressed in future research. Utz-Uwe Haus mentioned that one
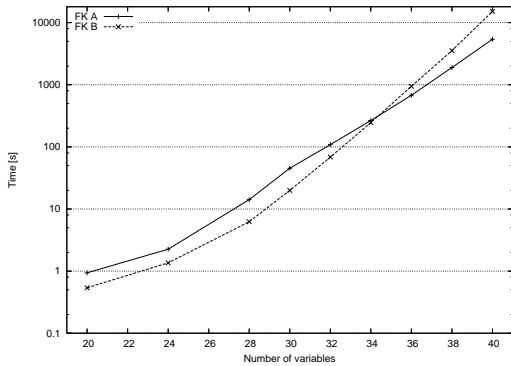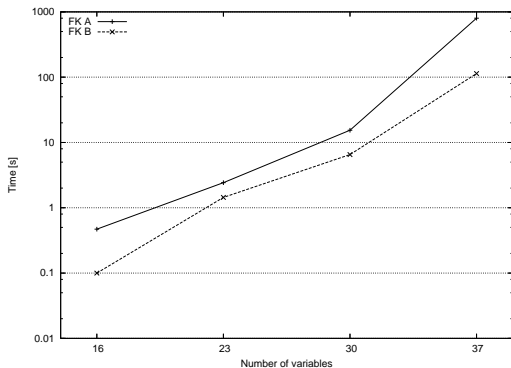
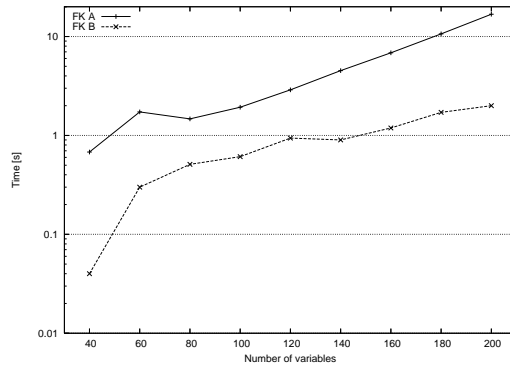Figure 1: Runtimes on M(v)

Figure 2: Runtimes on TH(v)
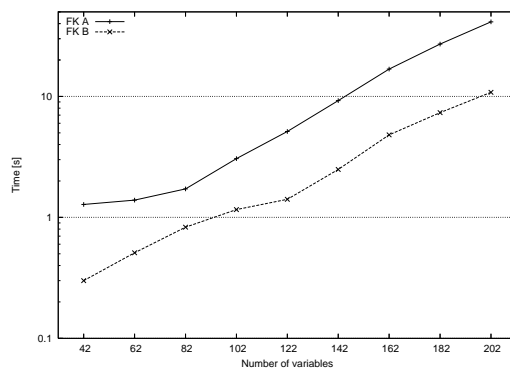

Figure 3: Runtimes on SDFP(v)

Figure 4: Runtimes on SDTH(v)

reason might be internal sorting of the terms [Hau08]. Anyway, our experiments show that FK-algorithm B should not *a priori* be excluded from experimental studies of MONET algorithms any more.

As for the computational variants of the FK-algorithms—given the DNF, compute the CNF—the relative behavior stays the same. However, runtime increases dramatically and, compared to a Java implementation of the DL-algorithm [DL05], our implemented computational variants of the FK-algorithms currently are rather slow.

A promising future research task would be the development of an unbiased, comprehensive, systematic experimental evaluation of all the known algorithms for MONET and the computational variant MONET′. Unfortunately, the existing studies mostly just show the potential of a single algorithm (and are often authored by the algorithm's developers), usually implemented on different platforms. From a more theoretical perspective it would be really interesting to have a theoretical lower bound for the FK-algorithms. Though Gurvich and Khachiyan [GK97] note that it should be possible to give a superpolynomial lower bound for FK-algorithm A using Takata-like instances, the proof is still open.

Giving a lower bound for FK-algorithm B seems to be even more involved.

References

[BHIK97] Endre Boros, Peter L. Hammer, Toshihide Ibaraki, and Kazuhiko Kawakami. Polynomial-time recognition of 2-monotonic positive Boolean functions given by an oracle. *SIAM Journal on Computing*, 26(1):93–109, 1997.

[BI95] Jan C. Bioch and Toshihide Ibaraki. Complexity of identification and dualization of positive Boolean functions. *Information and Computation*, 123(1):50–63, 1995.

[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 descrip-

tions 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.

[EGM03] Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003.

[Eit94] Thomas Eiter. Exact transversal hypergraphs and application to Boolean μ-functions. *Journal of Symbolic Computation*, 17(3):215–225, 1994.

[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.

[Hag07] Matthias Hagen. Lower bounds for three algorithms for the transversal hypergraph generation. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science, 33rd International Workshop, WG 2007, Dornburg, Germany, June 21-23, 2007. Revised Papers*, volume 4769 of *Lecture Notes in Computer Science*, pages 316–327. Springer, 2007.

[Hag08] Matthias Hagen. *Algorithmic and Computational Complexity Issues of* MONET. PhD thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena, December 2008.

[Hau08] Utz-Uwe Haus. Personal communication, June 2008.

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

[KBEG06] Leonid Khachiyan, Endre Boros, Khaled M. El-bassioni, and Vladimir Gurvich. An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation. *Discrete Applied Mathematics*, 154(16):2350–2372, 2006.

[KS03] Dimitris J. Kavvadias and Elias C. Stavropoulos. Monotone Boolean dualization is in coNP[$\log^2 n$]. *Information Processing Letters*, 85(1):1–6, 2003.

[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.

[LJ03] Li Lin and Yunfei Jiang. The computation of hitting sets: Review and new algorithms. *Information Processing Letters*, 86(4):177–184, 2003.

[LLK80] Eugene L. Lawler, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.

[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.

[Qui53] Willard van Orman Quine. Two theorems about truth functions. *Boletin de la Sociedad Matemática Mexicana*, 10:64–70, 1953.

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

[Tak07] Ken Takata. A worst-case analysis of the sequential method to list the minimal hitting sets of a hypergraph. *SIAM Journal on Discrete Mathematics*, 21(4):936–946, 2007.

[Tam00] Hisao Tamaki. Space-efficient enumeration of minimal transversals of a hypergraph. In *Proceedings 75th SIGAL Conference of the Information Processing Society of Japan (IPSJ-AL 75)*, pages 29–36, 2000. Extended paper available from the author.

[TT02] Vetle I. Torvik and Evangelos Triantaphyllou. Minimizing the average query complexity of learning monotone Boolean functions. *INFORMS Journal on Computing*, 14(2):144–174, 2002.

[US03] Takeaki Uno and Ken Satoh. Detailed description of an algorithm for enumeration of maximal frequent sets with irredundant dualization. In Bart Goethals and Mohammed Javeed Zaki, editors, *FIMI'03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.