Speeding up Dualization in the Fredman-Khachiyan Algorithm B

Nafiseh Sedaghat
School of Computing Science, Simon Fraser University
nf_sedaghat@sfu.ca

Tamon Stephen[1]
Department of Mathematics, Simon Fraser University
tamon@sfu.ca

Leonid Chindelevitch[2]
School of Computing Science, Simon Fraser University
leonid@sfu.ca

Abstract

The problem of computing the dual of a monotone Boolean function f is a fundamental problem in theoretical computer science with numerous applications. The related problem of duality testing (given two monotone Boolean functions f and g, declare that they are dual or provide a certificate that shows they are not) has a complexity that is not yet known. However, two quasi-polynomial time algorithms for it, often referred to as FK-A and FK-B, were proposed by Fredman and Khachiyan in 1996, with the latter having a better complexity guarantee. These can be naturally used as a subroutine in computing the dual of f.

In this paper, we investigate this use of the FK-B algorithm for the computation of the dual of a monotone Boolean function, and present practical improvements to its performance. First, we show how FK-B can be modified to produce multiple certificates (Boolean vectors on which the functions defined by the original f and the current dual g do not provide outputs consistent with duality). Second, we show how the number of redundancy tests - one of the more costly and time-consuming steps of FK-B - can be substantially reduced in this context. Lastly, we describe a simple memoization technique that avoids the solution of multiple identical subproblems.

We test our approach on a number of inputs coming from computational biology as well as combinatorics. These modifications provide a substantial speed-up, as much as an order of magnitude, for FK-B dualization relative to a naive implementation. Although other methods may end up being faster in practice, our work paves the way for a principled optimization process for the generation of monotone Boolean functions and their duals from an oracle.

2012 ACM Subject Classification Computing methodologies → Boolean algebra algorithms

Keywords and phrases Monotone boolean functions, dualization, Fredman-Khachiyan algorithm, algorithm engineering, metabolic networks

Digital Object Identifier 10.4230/LIPIcs.SEA.2018.6

1 Introduction

Boolean functions are a powerful modeling tool. In many applications, such as those described in [9], the relevant Boolean functions have a natural monotone structure, and can thus be understood and manipulated in terms of their minimal true and maximal false settings.

[1] Partially supported by an NSERC Discovery grant.
[2] Partially supported by an NSERC Discovery grant and a Sloan Foundation fellowship.

Obtaining and translating between these representations is a challenging and deep theoretical question. There are numerous applications to do this in areas such as graph theory (generating the transversal of a hypergraph), combinatorics (finding minimal hitting sets), and machine learning (model-based fault diagnosis). It is also of interest in more applied fields from security to networking and from distributed systems to computational biology. Our interest in the problem stems begins from computational biology, notably in computing elementary flux modes (EFMs) and minimal cut sets (MCSs) following [13].

Given the list of minimal true settings, the problem of generating the list of maximal false settings is the classical problem of hypergraph dualization, which has arisen in several contexts, see for example [9] for a survey, but is not well understood theoretically. The problem of jointly generating both lists when the function is presented as an oracle is an attractive problem. Notably Fredman and Khachiyan [7] found two novel algorithms for dualization, that, unlike other known algorithms for the problem, extend to oracle-based generation [10]. These algorithms, which we refer to as FK-A and FK-B, verify duality or generate new clauses in incremental quasi-polynomial time $N^{o(\log N^2)}$ and $N^{o(\log N)}$ respectively, where N is the current joint size of the input and output lists. These algorithms are poorly understood in theory and in practice. The only available open-source code for oracle-based generation is cl-jointgen [12] for FK-A, though some experiments on FK-A and FK-B are described in [11], and an FK-A based dualization algorithm by Elbassioni is also now available [5]. Highly parallel FK-type algorithms were proposed by Elbassioni [6] and Boros and Makino [1].

In this paper we address some of the computational challenges of using the FK-B algorithm for dualizing a monotone Boolean function, although our techniques can also be directly applied to the setting of jointly generating the minimal true and maximal false settings of a monotone Boolean function given as an oracle. Our main contribution is an extension of the basic FK-B algorithm to produce multiple conflicting assignments (abbreviated as CA) in a single iteration. Our second contribution is a substantial reduction in the number of redundancy tests (namely, a test to make sure that no clause is a strict superset of another one, required to maintain correctness) during the execution of FK-B. Lastly, we find that a number of the subproblems arising during the different calls to FK-B are identical, and this leads to our final contribution - the use of a memoization technique to speed up dualization. Each improvement alone produces a substantial speed-up, and in combination, they result in an order of magnitude speed gain relative to a naive (unoptimized) implementation.

2 Definitions

Let n be fixed. We write $\mathcal{B}$ to denote the set $\{0,1\}$. A Boolean function $f : \mathcal{B}^n \to \mathcal{B}$ is *monotone* if $f(s) \leq f(t)$ for any two vectors $s \leq t \in \mathcal{B}^n$, where the inequality is interpreted component-wise. In other words, replacing a 0 with a 1 in the input cannot decrease f's value. Monotone functions are precisely those that can be constructed using the OR and AND operations, without any NOT gates.

The dual of a Boolean function f is the function f^d defined by:

$$f^d(x) = \overline{f(\overline{x})} \tag{1}$$

for all $x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$, where $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$.

A monotone Boolean function f is said to be in Disjunctive Normal Form (DNF) if it is represented as an OR of ANDs, i.e. as

$$f = \bigvee_{j=1}^{m} M_j, \text{ where } M_j = \bigwedge_{i \in T_j} x_i.$$

Here, the monomials M_j are called *implicants* of f. If the underlying sets T_j satisfy the *Sperner property*, i.e. $T_j \not\subset T_k$ whenever $j \neq k$, then each M_j is also called a *prime implicant* of f and m is called the *size* of f. In this case, the point x defined by

$$x_i = \begin{cases} 1 \text{ if } i \in T_j \\ 0 \text{ otherwise} \end{cases}$$

is a *minimal true point* of f; indeed, for this x we have $f(x) = 1$ and $f(y) = 0$ for any $y < x$, where $y < x$ if and only if $y \leq x$ and $y \neq x$.

Similarly, a monotone Boolean function f is said to be in Conjunctive Normal Form (CNF) if it is represented as an AND of ORs, i.e. as

$$f = \bigwedge_{j=1}^{m} C_j, \text{ where } C_j = \bigvee_{i \in S_j} x_i.$$

Here, the clauses C_j are called *implicates* of f. Once again, if the underlying sets S_j satisfy the Sperner property, then each C_j is also called a *prime implicate* of f and m is called the *size* of f. In this case, the point x defined by

$$x_i = \begin{cases} 0 \text{ if } i \in S_j \\ 1 \text{ otherwise} \end{cases}$$

is a *maximal false point* of f; indeed, for this x we have $f(x) = 0$ and $f(y) = 1$ for any $y > x$.

Lastly, we define the *support* of x, denoted *supp*(x), as the set $\{i \in \{1, 2, \ldots, n\} | x_i = 1\}$.

In the context of a metabolic network model M, which we use as one of the test cases here, the monotone Boolean function f is defined on subsets of reactions via $f(x) = 1$ if and only if the support of x enables biomass production. In this setting the minimal true points of f are called elementary flux modes (EFMs) and the maximal false points of f are called minimal cut sets (MCSs), see for example [20, 14].

We can define two related problems for monotone Boolean functions:

- testing the equivalence of two monotone Boolean functions defined by a DNF and a CNF;
- the dualization problem: computing the equivalent CNF when a monotone DNF is given.

These two problems can be easily transformed into one another. The dualization problem is equivalent to *Transversal Hypergraph Generation*, also called the *Minimal Hitting Set Enumeration* problem. For background on these problems and applications, we refer the reader to [4, 11, 3, 9] and references therein.

The study of monotone Boolean functions is a vibrant area of ongoing research. The algorithm with the best known worst-case performance guarantee for the dualization of a monotone Boolean function f has incremental quasi-polynomial running time. More precisely, starting from a description of f in DNF, each iteration obtains an additional clause of the equivalent CNF, in $N^{O(logN)}$ time, where N is the total size of the DNF and the current, possibly incomplete, CNF [7].

2.1 The FK-Dualization Algorithm

Algorithms 1 and 2 show the FK dualization and FK-B duality checking procedure respectively, following the presentations of [3] and [15].

Let ϕ be a DNF or CNF, and let x be a splitting variable. Then ϕ_0^x denotes the formula that consists of the terms of ϕ from which x is removed: $\phi_0^x = \{t - \{x\} : t \in \phi\}$. Analogously, ϕ_1^x denotes the formula that consists of all terms of ϕ that do not contain x: $\phi_1^x = \{t : t \in \phi \text{ and } x \notin t\}$.

Algorithm 1 Fredman-Khachiyan Dualization.

Input: A positive Boolean function f on B^n expressed by its complete DNF.
Output: The complete DNF of f^d.

```
1: function $FK\_$DUALIZATION($f$)

2:     $g = 0$;
3:     Call $FK_B$ on the pair $(f, g)$;
4:     if the returned value is "Yes" then
5:         halt;
6:     else
7:         let $X^* \in \mathcal{B}^n$ be the point returned by $FK_B$;
8:         compute a maximal false point of $f$ , say $Y^*$, such that $X^* \leq Y^*$;
9:         $g = g \vee \bigwedge_{j \in supp(\overline{Y^*})} x_j$ ;
10:        return to Step 3.
```

A variable x is called at most μ-frequent in D if its frequency in D is at most $1/\mu(|D| \cdot |C|)$, i.e. $|\{m \in D : x \in m\}|/|D| \leq 1/\mu(|D| \cdot |C|)$ where $\mu(n) \sim \log n / \log \log n$. A similar definition applies to C.

The original FK-B algorithm returns the first conflicting assignment (CA) that it finds between the given CNF and DNF. Hence computing the dual of a given DNF requires $N_{CNF} + 1$ iterations, where N_{CNF} is the size of the CNF that is dual to the given DNF.

3 Methods

3.1 Pre-processing and Post-processing Steps

When analyzing metabolic networks to obtain the MCSs from the EFMs, it is beneficial to pre-process the given EFMs before starting the dualization procedure. The preprocessing involves three steps:

- removing any reactions that are not part of any EFMs (also known as blocked reactions [2, 8]), which correspond to unused variables;
- removing any reactions involved in all the EFMs (also referred to as essential reactions [2, 8]), adding them as singleton MCSs in post-processing;
- collapsing any set of k reactions whose presence/absence patterns in clauses are identical (a special case of this is referred to as enzyme subsets [2, 8]) into a single reaction, expanding each of the final MCSs involving this reaction into k copies in post-processing.

The pre-processing and post-processing steps are not necessary and can be ignored. However, they reduce the original problem and make the dualization procedure faster, so we routinely perform these steps.

3.2 Finding Multiple Conflicting Assignments

Given that we can use any conflicting assignment between the current CNF and DNF to compute a new clause in CNF, we can find Multiple Conflicting Assignments (MCAs) at the same time to generate more than one clause per iteration of the dualization procedure, and reduce the running time of the algorithm by reducing the total number of required iterations.

Algorithm 2 The Fredman-Khachiyan Algorithm B (FK-B).

Input: irredundant, monotone DNF D and CNF C.
Output: $\emptyset$ in case of equivalence; otherwise, assignment $\mathcal{A}$ with $\mathcal{A}(D) \neq \mathcal{A}(C)$.

```
1: function FK-B($C, D$)
2:     make $D$ and $C$ irredundant;
3:     if a necessary condition is violated then return conflicting assignment;
4:     if $\min\{|D|, |C|\} \leq 2$ then return conflicting assignment found by a trivial check;
5:     else
6:         choose a splitting variable $x$ from the formulae
7:         if $x$ is at most $\mu$-frequent in $D$ then
8:             $\mathcal{A} \leftarrow$ FK-B($D_1^x, C_0^x \wedge C_1^x$) // recursive call for $x$ set to $false$
9:             if $\mathcal{A} \neq \emptyset$ then return $\mathcal{A}$
10:            for all clauses $c \in C_0^x$ do do
11:                $\mathcal{A} \leftarrow$ FK-B($D_0^{c,x}, C_1^{c,x}$) // see $\langle 1 \rangle$
12:                if $\mathcal{A} \neq \emptyset$ then return $\mathcal{A} \cup \{x\}$
13:        else if $x$ is at most $\mu$-frequent in $C$ then
14:            $A \leftarrow$ FK-B($D_0^x \vee D_1^x, C_1^x$) // recursive call for $x$ set to $true$
15:            if $\mathcal{A} \neq \emptyset$ then return $\mathcal{A} \cup \{x\}$
16:            for all monomials $m \in D_0^x$ do do
17:                $\mathcal{A} \leftarrow$ FK-B($D_1^{m,x}, C_0^{m,x}$) // see $\langle 2 \rangle$
18:                if $\mathcal{A} \neq \emptyset$ then return $\mathcal{A} \cup \{m\}$
19:        else
20:            $\mathcal{A} \leftarrow$ FK-B($D_1^x, C_0^x \wedge C_1^x$) // recursive call for $x$ set to $false$
21:            if $\mathcal{A} = \emptyset$ then
22:                $A \leftarrow$ FK-B($D_0^x \vee D_1^x, C_1^x$) // recursive call for $x$ set to $true$
23:                if $\mathcal{A} \neq \emptyset$ then return $\mathcal{A} \cup \{x\}$
24:    return $\mathcal{A}$
```

$\langle 1 \rangle$: $D_1^x \equiv C_0^x \wedge C_1^x$: recursive call for all maximal non-satisfying assignments of C_0^x for x set to $true$. $D_0^{c,x}$ and $C_1^{c,x}$ denote the formulae we obtain if we set all variables in c to $false$.

$\langle 2 \rangle$: $D_0^x \vee D_1^x \equiv C_1^x$: recursive call for all minimal satisfying assignments of D_0^x for x set to $false$. $D_1^{m,x}$ and $C_0^{m,x}$ denote the formula we obtain if we set all variables in m to $true$.

To this end, MCAs can be computed in the situations below without significant increase of computational effort. The first two situations arise during the assessment of the first two conditions necessary for equivalence in FK-B.

The first condition that we assess in the FK-B algorithm is the existence of a non-empty intersection between every clause in CNF and every monomial in DNF. If there is no intersection between monomial $m \in DNF$ and clause $c \in CNF$, then m makes the DNF *true* and the CNF *false*, so it is a CA. During the dualization procedure, especially early on, many of the monomials and clauses have no intersection. We thus consider intersections between every clause in the CNF and every monomial in the DNF at once and can return more than one CA.

The second condition that we assess in the FK-B algorithm is the presence of exactly the same variables in the CNF and the DNF. If this condition is not met, a CA is determined from the extra variable(s) in the CNF or the DNF. If multiple variables are present in exactly one of the CNF and the DNF, we consider all possible conflicting assignments instead of returning only one.

The third situation in which we compute MCAs is in the case where $\min(|C|, |D|) \leq 2$. In such cases, the conflicting assignment is directly derived from Boolean algebra. We only consider the case $|C| \leq 2$ here; the case $|D| \leq 2$ is symmetric and is processed analogously.

The following cases may happen during this step:

- $|C| = 1$
  Here, we look for a variable x in the unique CNF clause, denoted $C[1]$, such that D does not contain the singleton monomial x, in which case $\{x\}$ is a conflicting assignment.
- $|C| = 2$
  In this case, we denote the two clauses by $C[1]$ and $C[2]$. There are three sub-cases:
  - Let $A_0 := C[1] \cap C[2]$. If $x \in A_0$ is a variable such that D does not contain the singleton monomial x, then $\{x\}$ is a conflicting assignment.
  - Let $A_1 := C[1] - C[2]$ and $A_2 := C[2] - C[1]$. Note that $A_1, A_2 \neq \emptyset$. If some monomial m in D is a subset of one of the A_i's, then $\{x | x \in m\}$ is a conflicting assignment.
  - Let $(x, y) \in A_1 \times A_2$ (defined in the previous case). If no monomial in D is a subset of $\{x, y\}$ then $\{x, y\}$ is a conflicting assignment.

In all the aforementioned cases, whenever there is more than one conflicting assignment we return all of them. An issue regarding MCAs is that sometimes more than one CA can be mapped to a single clause in the CNF.

In the following sections, we refer to this version of FK-B as FK_M.

3.3 Reducing the Number of Redundancy Tests in the FK-B Algorithm

In the original FK-B algorithm, Algorithm 2, the first two instructions (line 2) remove redundancy in both the CNF and the DNF. During redundancy removal one performs an all-pairs comparison of the clauses (the monomials) in the CNF (the DNF) and remove any supersets found. In logic, removing redundancy is equivalent to applying the absorption rule to simplify the Boolean function.

In the FK-B algorithm, this procedure is a bottleneck due to the large number of pairwise comparisons that must be performed in each recursive call, and this is compounded by the fact that when we perform dualization, the FK-B algorithm is iterated many times to find the clauses of the CNF.

We reduce the number of redundancy tests performed in FK-B, and consequently in FK-dualization, by noting that when we set a variable to *true* ($false$) in the CNF (DNF), there is no need to check the redundancy of the CNF (DNF) in the next recursive call because such a setting results in one or more clauses (monomials) in the CNF (DNF) being removed, which cannot generate redundancy.

This can simply be implemented using two binary flags, which are respectively set if and only if the redundancy in the CNF (the DNF) should be checked, and cleared otherwise. For simplicity, we call this algorithm FK_R. It differs from the baseline, algorithm 2, in the following ways. First, the redundancy of the CNF (the DNF) is only checked if the corresponding flag is set. Second, in lines 8 and 20, where a variable x is set to $false$, the flag for the CNF is set and the flag for the DNF is cleared, since we only need to check the redundancy in the CNF, not the DNF. Conversely, in lines 14 and 22, the variable x is set to *true*, so the flag for the DNF is set and the flag for the CNF is cleared. Note that in lines

11 and 17, it is assumed that variable x is respectively set to $true$ and $false$, and then the variables in c and m are respectively set to $false$ and $true$. For this reason, redundancy can be produced in those lines, so the next call to FK_R needs to check the redundancy in both the CNF and the DNF.

3.4 Dealing with Repeated subproblems

Given that the FK-B algorithm is a recursive algorithm which is called multiple times during dualization, we encounter many subproblems solved in the previous recursive calls or past iterations. In this case, memoizing (storing for future retrieval) these subproblems and their solutions (in the form of CAs) is beneficial, as it can reduce the running time of both the FK-B algorithm as well as FK-dualization as a whole.

To this end, we use a hash table whose keys are combination of the CNF and the DNF and whose values are the CAs between them. To implement this idea, we compute the key for a given CNF and DNF prior to calling the FK algorithm. If it is already in the hash table, we retrieve the value, i.e. the corresponding CAs, bypassing a recursive call to FK. Otherwise, we call FK and store the computed CAs as a new record in the hash table.

As we experimented with different settings in implementation of the memoization idea, we realized that solving small subproblems, with $|C| < 3$ and $|D| < 3$, from scratch was faster than storing them in the hash table and retrieving the CAs. Thus, in our implementation we do not use the hashing technique for small subproblems. The method that uses hashing in the FK-B algorithm is referred to as FK_H.

4 Experimental Results

To assess the proposed algorithms, we run them on 12 problems including seven biological (metabolic network) models downloaded from the BioModels database[3] as well as 5 synthetic models. The code and examples that we used are available on GitHub [16].

We first parsed the biological models using the SBML parser in *MATLAB* to obtain a stoichiometric matrix containing the reactions and the metabolites, then applied EFMTool [18]/FluxModeCalculator [19] to extract the EFMs into a matrix. We converted this matrix into a binary one by setting all non-zero values to one, and used this as the input DNF, as is standard when looking for MCSs. We have chosen models with small to medium sizes.

The synthetic models included in our experiments are the 3×3 magic squares, called *'ms-33'*, 3×3 semi-magic squares, called *'sms-33'*, and three problems 'ac-200k', 'SDFP16', and 'SDFP23' taken from the Hypergraph Dualization Repository[4]. 'ac-200k' is the complement of the set of maximal frequent itemsets with support threshold $200,000$ from the "accident" dataset. 'SDFP16' and 'SDFP23' are the Self-Dual Fano Plane hypergraphs with respectively $n = 16$ and $n = 23$ vertices, and $(k_n - 2)^2/4 + k_n/2 + 1$ hyperedges, where $k_n = (n - 2)/7$.

4.1 Evaluating the Speed of the Algorithms

To measure the running time of the proposed algorithms we used the `timeit` function in *MATLAB*, which measures the typical running time of functions in seconds. `timeit` automatically repeats the input function in a timing loop a number of runs determined by a built-in heuristic method, and returns the median value across the runs.

[3] `http://www.ebi.ac.uk/biomodels-main/publmodels`
[4] `http://research.nii.ac.jp/~uno/dualization.html`

Table 1 Characteristics of models; n_{metab}: number of metabolites, n_{EFM}: number of elementary flux modes or monomials in DNF, $n_r^{<pre}$: Number of reactions/variables before preprocessing steps, $n_r^{>pre}$: Number of reactions/variables after preprocessing, $n_{MCS}^{<post}$: Number of minimal cut sets or clauses in CNF before postprocessing, $n_{MCS}^{>post}$: Number of minimal cut sets or clauses in CNF after post-processing.

Model	n_{metab}	n_{EFM}	$n_r^{<pre}$	$n_r^{>pre}$	$n_{MCS}^{<post}$	$n_{MCS}^{>post}$
BIOMD0000000048	23	63	25	14	320	12960
BIOMD0000000093	34	24	46	24	293	2001
BIOMD0000000094	34	23	45	23	293	667
BIOMD0000000089	16	20	36	28	192	15552
BIOMD0000000228	9	13	22	20	128	512
BIOMD0000000034	9	13	22	22	56	56
BIOMD0000000042	15	35	25	20	56	188
sms_33	-	48	9	9	81	81
ms_33	-	40	9	9	76	76
ac_200k	-	81	64	21	210	253
SDFP16	-	64	16	16	64	64
SDFP23	-	365	23	23	365	365

Table 2 Running time in seconds; FK_M: FK-dualization algorithm that returns multiple conflicting assignments, FK_R: Variant of FK with a reduction in the number of redundancy tests, FK_{MHR}: Variant of FK_M with a reduction in the number of redundancy tests that stores solved subproblems in a hash table, FK_{MHCR}: Variant of FK_{MHR} that stores solved subproblems in a hash table and uses the canonical form for small ones. In the last four columns, τ is the threshold such that if $|C| < \tau$ and $|D| < \tau$, the hash table is not used (when $\tau = 0$ it is always used).

	FK	FK_M	FK_R	FK_{MHR} ($\tau = 0$)	FK_{MHR} ($\tau = 3$)	FK_{MHCR} ($\tau = 0$)	FK_{MHCR} ($\tau = 3$)
BIOMD0000000048	119.06	62.71	86.13	12.82	**12.67**	17.29	12.80
BIOMD0000000093	136.68	88.69	117.54	**14.44**	15.84	25.24	15.92
BIOMD0000000094	140.37	78.81	122.28	**19.31**	21.69	37.71	21.81
BIOMD0000000089	31.27	26.6	25.96	9.59	**8.52**	15.62	8.57
BIOMD0000000228	9.30	6.41	7.82	**2.42**	2.81	5.57	3.06
BIOMD0000000034	1.56	1.34	1.36	0.45	**0.42**	1.05	0.43
BIOMD0000000042	7.74	3.72	7.24	**0.19**	**0.19**	0.20	0.22
sms_33	3.95	1.72	3.27	**0.46**	0.55	3.06	2.45
ms_33	2.90	1.26	2.28	**0.46**	0.64	2.79	2.17
ac_200k	311.57	35.22	305.69	**7.13**	7.85	13.44	11.65
SDFP16	6.67	2.80	5.86	**0.40**	0.58	1.90	0.56
SDFP23	648.59	251.05	504.30	**103.66**	108.98	130.81	108.67

Table 2 shows the running time of finding the dual of a given DNF. The time measurements only show the time required for dualization, exclusive of pre- and post-processing.

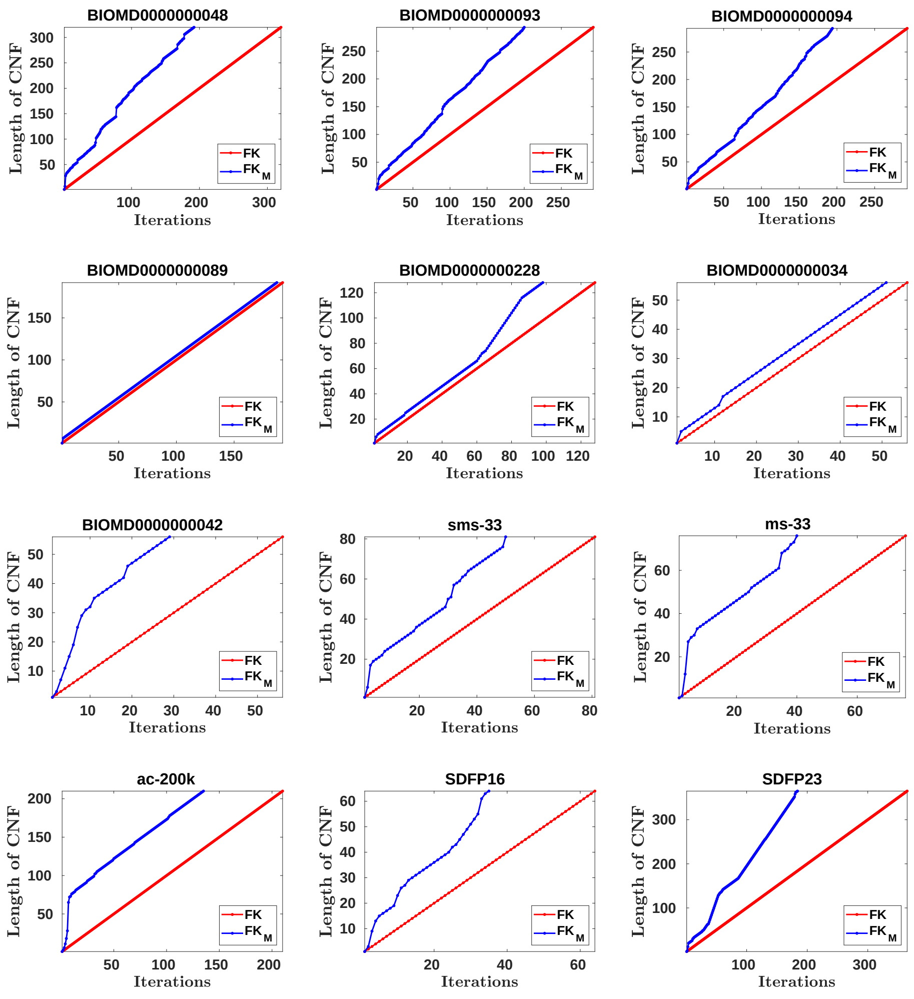

Figure 1 Size of the CNF versus number of iterations in FK-dualization and FK_M-dualization.

4.2 Reducing the Number of Iterations in FK-dualization Using FK_M

To show how effective FK_M-dualization is in comparison to FK-dualization we run both methods on our 12 problems and stored the size of the CNF in each iteration of the algorithms. Figure 1 shows the results. As can be seen in this figure, FK_M-dualization constructs the final CNF faster and requires as few as half the iterations in comparison to FK-dualization.

We have also counted the number of recursive calls to FK and FK_M in their corresponding dualization procedures. It turns out that by returning multiple conflicting assignments, FK_M requires fewer recursive calls than FK for every problem except ‘ac-200k’, as illustrated in Figure 2. Although finding multiple conflicting assignments always reduces the total number of iterations (except for one of the metabolic models, where it stays the same, as shown in Figure 1), the number of recursive calls to FK_M can occasionally exceed the number of recursive calls to FK, presumably due to changes in the traversal of the assignment tree.

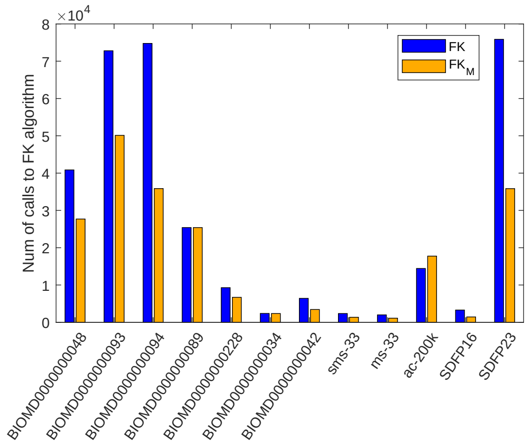

Figure 2 Comparing the number of recursive calls to FK and FK_M during dualization.

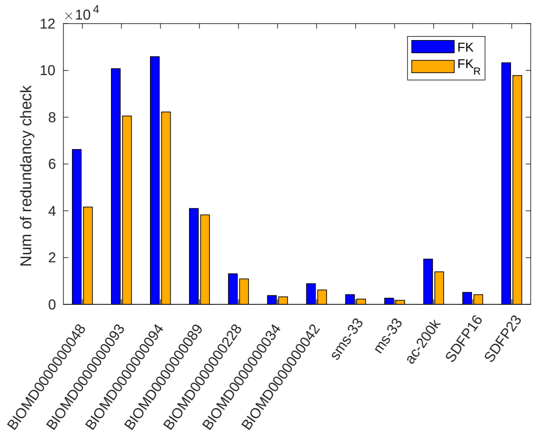

Figure 3 Comparing the number of redundancy tests in FK and FK_R dualization.

4.3 Reducing the Number of Redundancy Tests in the FK-B Algorithm

In this experiment, we show how using the two flags in the FK_R algorithm in comparison with FK reduces the number of redundancy tests in both the CNF and the DNF. To this end we count the number of calls to the redundancy testing function.

Figure 3 shows the results. As expected, using the two redundancy flags reduces the number of redundancy tests in the FK_R algorithm relative to the baseline FK algorithm.

In small problems, the reduction is not significant, however, in large problems like *BIOMD0000000048*, the number of redundancy tests in FK_R is about $\frac{2}{3}$ of what it is in FK. Given that the procedure of redundancy test is a bottleneck of the FK algorithm, fewer calls to this function reduces the time required to find the dual of a given monotone Boolean function.

4.4 Dealing with Repeated subproblems Using a Hash Table

In this experiment we show the efficiency of using a hash table for dualization. We count the number of successful key and value retrievals from the hash table. Figure 4 shows the characteristics of the subproblems, e.g. the frequency and the size of the subproblem, corresponding to the first five most popular keys in FK_{MH} dualization. For each model there are two figures, corresponding to $\tau = 0$ and $\tau = 3$, meaning that we use the hash table only for subproblems with $|C| > \tau$ and $|D| > \tau$.

Comparing the figures in columns with $\tau = 0$ to the figures in columns with $\tau = 3$ demonstrates that when we use hash table for every subproblem, i.e. $\tau = 0$, the size of frequent subproblems is very small while in the other case, i.e. $\tau = 3$, the size of the frequent subproblems is larger.

Since there is the possibility that some of the frequently occurring Boolean functions differ only by a permutation of the variables, we have implemented a simple scheme for reducing functions of up to seven variables to a canonical form as is done in Stephen and Yusun [17]. We found that the most popular keys in the implementation of the hash table with and without the canonical form are the same, thus, we only present the results without the canonical form here, as FK_{MH}.

5 Discussion

The FK-B algorithm was a theoretical breakthrough at the time of its discovery in 1996, showing for the first time that the problem of testing the duality of two monotone Boolean functions could be solved in quasi-polynomial time. Because of its ability to produce a certificate of non-duality, this also means that, given a Boolean function in explicit form, its dual can be generated in incremental quasi-polynomial time. In addition, the FK-B algorithm also applies to joint generation, namely, an explicit description of the function f and of its dual in the case where f is given implicitly by an oracle (an algorithm that, given an input x, returns the value $f(x)$) - with the same complexity guarantees provided that the oracle runs in polynomial time.

However, despite these exciting developments the FK-B algorithm is not easily usable for generating the dual of a large monotone Boolean function in practice. In this paper we provided several improvements that reduce the overall running time by an order of magnitude on most of the examples we considered. Although they do not change the worst-case algorithmic complexity of dualization via the FK-B algorithm, they take it closer to being usable in practice on medium-to-large-scale problems.

All our techniques also apply directly to the problem of joint generation. The generation of multiple conflicting assignments per iteration, the reduction in the number of redundancy tests, and the use of memoization could all turn out to be beneficial in this scenario as well. One of our future directions is to explore how helpful these techniques are in the context of joint generation (which, for metabolic network models, would amount to simultaneously generating the elementary modes and the minimal cut sets of the network).

In conclusion, our work paves the way for a systematic exploration of algorithmic improvements to Monotone boolean function dualization and joint generation algorithms that use a duality testing algorithm such as FK-B as a subroutine. We expect that, with additional algorithmic improvements, this approach will ultimately result in a practical method for solving this important problem.

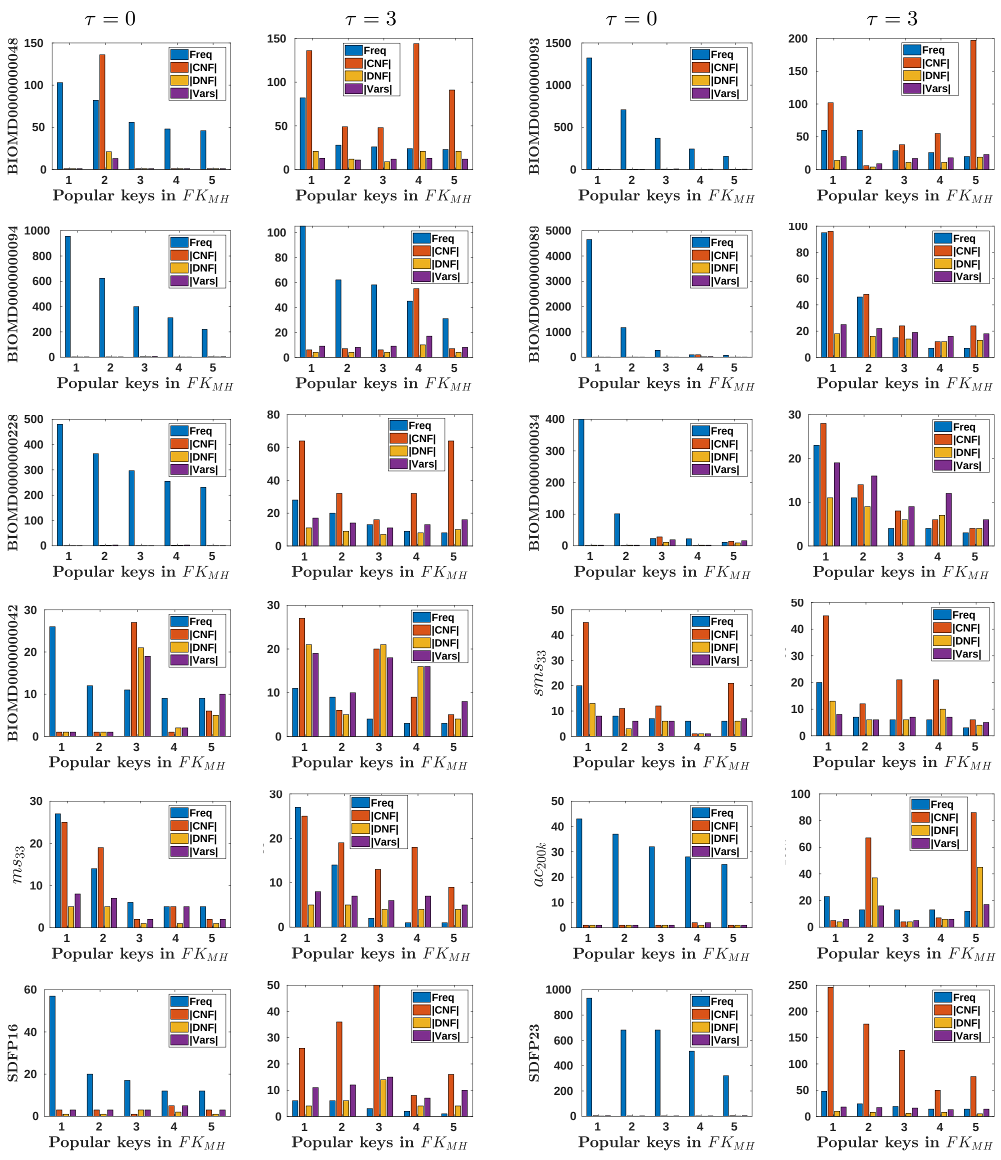

Figure 4 Characteristics of most frequent subproblems in the hash table in FK_{MH} dualization. In this set of experiments, hash table has been used for subproblems with $|C| > \tau$ and $|D| > \tau$.

References

1 Endre Boros and Kazuhisa Makino. A fast and simple parallel algorithm for the monotone duality problem. In *Automata, languages and programming. Part I*, volume 5555 of *Lecture Notes in Comput. Sci.*, pages 183–194. Springer, Berlin, 2009.

2 Leonid Chindelevitch, Jason Trigg, Aviv Regev, and Bonnie Berger. An exact arithmetic toolbox for a consistent and reproducible structural analysis of metabolic network models. *Nature Communications*, 5(4893):165–182, 2014.

3 Yves Crama and Peter L Hammer. *Boolean functions: Theory, algorithms, and applications.* Cambridge University Press, 2011.

4 Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational aspects of monotone dualization: a brief survey. *Discrete Appl. Math.*, 156(11):2035–2049, 2008.

5 Khaled Elbassioni. C implementation of Fredman and Khachiyan's algorithm A. Available from `https://github.com/VeraLiconaResearchGroup/MHSGenerationAlgorithms/blob/master/containers/fka-begk`.

6 Khaled M. Elbassioni. On the complexity of monotone dualization and generating minimal hypergraph transversals. *Discrete Appl. Math.*, 156(11):2109–2123, 2008.

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

8 Julien Gagneur and Steffen Klamt. Computation of elementary modes: a unifying framework and the new binary approach. *BMC Bioinformatics*, 5(175), 2004.

9 Andrew Gainer-Dewar and Paola Vera-Licona. The minimal hitting set generation problem: algorithms and computation. *SIAM J. Discrete Math.*, 31(1):63–100, 2017.

10 V. Gurvich and L. Khachiyan. On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions. *Discrete Appl. Math.*, 96/97:363–373, 1999. The satisfiability problem (Certosa di Pontignano, 1996); Boolean functions.

11 Matthias Hagen, Peter Horatschek, and Martin Mundhenk. Experimental comparison of the two Fredman-Khachiyan-algorithms. In *Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 154–161. SIAM, 2009.

12 Utz-Uwe Haus. `cl-jointgen`, a common lisp implementation of the joint-generation method. Available from `https://sourceforge.net/projects/cl-jointgen/` and in C at `https://sourceforge.net/projects/jointgen-c.cl-jointgen.p/`.

13 Utz-Uwe Haus, Steffen Klamt, and Tamon Stephen. Computing knock-out strategies in metabolic networks. *J. Comput. Biol.*, 15(3):259–268, 2008.

14 Steffen Klamt and Ernst Dieter Gilles. Minimal cut sets in biochemical reaction networks. *Bioinformatics*, 20(2):226–234, 2004.

15 Martin Mundhenk and Robert Zeranski. How to apply SAT-solving for the equivalence test of monotone normal forms. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011*, pages 105–119, Berlin, 2011. Springer.

16 Nafiseh Sedaghat. Matlab implementation of Fredman and Khachiyan's algorithm B. Available from `https://github.com/WGS-TB/FK/tree/master/Modified%20FK-B%20Algorithm`.

17 Tamon Stephen and Timothy Yusun. Counting inequivalent monotone Boolean functions. *Discrete Appl. Math.*, 167:15–24, 2014.

18 Marco Terzer and Jörg Stelling. Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics*, 24(19):2229–2235, 2008.

19 Jan Bert Van Klinken and Ko Willems van Dijk. FluxModeCalculator: an efficient tool for large-scale flux mode computation. *Bioinformatics*, 32(8):1265–1266, 2015.

20 Jürgen Zanghellini, David E. Ruckerbauer, Michael Hanscho, and Christian Jungreuthmayer. Elementary flux modes in a nutshell: Properties, calculation and applications. *Biotechnology Journal*, 8(9):1009–1016, 2013. `doi:10.1002/biot.201200269`.