Speeding up the structural analysis of metabolic network models using the Fredman-Khachiyan Algorithm B

Journal:	*Journal of Computational Biology*
Manuscript ID	JCB-2022-0319.R2
Manuscript Type:	Original Paper
Keyword:	algorithms, biochemical networks, combinatorial optimization, computational molecular biology
Manuscript Keywords (Search Terms):	metabolic flux analysis, algorithms, monotone Boolean functions, minimal cut sets, metabloic networks
Abstract:	The problem of computing the Elementary Flux Modes (EFM) and Minimal Cut Sets (MCS) of metabolic network is a fundamental one in metabolic networks. A key insight is that they can be understood as a dual pair of monotone Boolean functions. Using this insight, this computation reduces to the question of generating from an oracle a dual pair of monotone Boolean functions. If one the two sets (functions) is known, then the other can be computed via a process known as dualization. Fredman and Khachiyan provided two algorithms, which they called simply A and B, that can serve as an engine for oracle-based generation or dualization of monotone Boolean functions. We look at efficiencies available in implementing their algorithm B, which we will refer to as FK-B. Like their algorithm A, FK-B certifies whether two given monotone Boolean functions in the form of Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF) are dual or not, and in case of not being dual it returns a conflicting assignment (CA), i.e. an assignment that makes one of the given Boolean functions True and the other one False. The FK-B algorithm is a recursive algorithm that searches through the tree of assignments to find a CA. If it does not find any CA, it means that the given Boolean functions are dual. In this paper, we propose six techniques applicable to the FK-B and hence to the dualization process. Although these techniques do not reduce the time complexity, they considerably reduce the running time in practice. We evaluate the proposed improvements by applying them to compute the MCSs from the EMSs in the 19 small- and medium-sized models from the BioModels database along with 4 models of biomass synthesis in E. coli that were used in an earlier computational survey.

Speeding up the structural analysis of metabolic network models using the Fredman-Khachiyan Algorithm B

Nafiseh Sedaghat,[1], Tamon Stephen[2,*], Leonid Chindelevitch[3]

[1]School of Computing Science, Simon Fraser University, B.C., Canada,

[2]Department of Mathematics, Simon Fraser University, B.C., Canada,

[3]MRC Centre for Global Infectious Disease Analysis, School of Public Health, Imperial College London, United Kingdom

[*]To whom correspondence should be addressed;

E-mail: nf_sedaghat@sfu.ca, tamon@sfu.ca, l.chindelevitch@imperial.ac.uk.

March 27, 2023

Keywords: algorithms, biochemical networks, combinatorial optimization, computational molecular biology

Abstract: The problem of computing the Elementary Flux Modes (EFM) and Minimal Cut Sets (MCS) of metabolic network is a fundamental one in metabolic networks. A key insight is that they can be understood as a dual pair of monotone Boolean functions. Using this insight, this computation reduces to the question of generating from an oracle a dual pair of monotone Boolean functions. If one

the two sets (functions) is known, then the other can be computed via a process known as dualization.

Fredman and Khachiyan provided two algorithms, which they called simply A and B, that can serve as an engine for oracle-based generation or dualization of monotone Boolean functions. We look at efficiencies available in implementing their algorithm B, which we will refer to as FK-B. Like their algorithm A, FK-B certifies whether two given monotone Boolean functions in the form of Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF) are dual or not, and in case of not being dual it returns a conflicting assignment (CA), i.e. an assignment that makes one of the given Boolean functions *True* and the other one *False*. The FK-B algorithm is a recursive algorithm that searches through the tree of assignments to find a CA. If it does not find any CA, it means that the given Boolean functions are dual.

In this paper, we propose six techniques applicable to the FK-B and hence to the dualization process. Although these techniques do not reduce the time complexity, they considerably reduce the running time in practice. We evaluate the proposed improvements by applying them to compute the MCSs from the EFMs in the 19 small- and medium-sized models from the BioModels database along with 4 models of biomass synthesis in *E. coli* that were used in an earlier computational survey **?**.

1 Introduction

Boolean functions, defined as functions whose input is a vector $x \in \{0,1\}^n$ and whose output is $f(x) \in \{0,1\}$, are a powerful modeling tool in a variety of settings. In many applications, such as those described in **?**, the relevant Boolean functions have a natural monotone structure, meaning that $x \leq y \implies f(x) \leq f(y)$, with the vector inequality interpreted component-wise. For this reason, they can be fully represented and analyzed in terms of either their minimal true settings or their maximal false settings. The **dualization** problem for monotone Boolean functions, which consists of translating between these two representations, is both a deep theoretical question as well as a practically important challenge. It is closely related to the **generation** problem, which consists of enumerating all the minimal true and maximal false settings of a monotone Boolean function specified as an oracle, i.e. providing the value of $f(x)$ given an input x **?**.

The dualization problem has numerous applications in subfields of mathematics such as graph theory (computing the transversal of a hypergraph), combinatorics (finding minimal hitting sets), and machine learning (model-based fault diagnosis), as well as more applied fields, including security, networking, distributed systems and computational biology. Our interest in the problem stems from computational biology, where we seek to perform a complete structural analysis of a metabolic network model by generating its elementary flux modes (EFMs) and minimal cut sets (MCSs), following **?**. This problem is of ongoing interest, see for example **?**.

The verification version of the dualization problem, also called the **duality** problem in the literature, consists of deciding whether a list of maximal false settings and a list of minimal true settings define the same monotone Boolean function. For instance $CNF = (x_1 \vee x_2) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_3)$, and

$DNF = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3) \vee (x_1 \wedge x_4)$ where $\vee$ represents "or" and $\wedge$ represents "and" are equivalent ways of describing a monotone Boolean function.

The computational complexity of the duality problem is not tightly characterized. Fredman and Khachiyan **?** found two novel algorithms for this decision problem that also extend to oracle-based generation of both the main function and its dual **?**. These algorithms, called FK-A and FK-B, either certify the duality or generate a new minimal true or maximal false setting in quasi-polynomial time in the joint size of the two lists. Their behaviour in practice is poorly understood. The only available open-source implementation for oracle-based generation is cl-jointgen **?** for FK-A, though some experiments on FK-A and FK-B are described in **?**, and an FK-A based dualization algorithm is also available **?**.

In this project, we address some computational challenges of using the FK-B algorithm for dualizing a monotone Boolean function. 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. While motivated by metabolic networks, our techniques are completely general.

A preliminary version of this work **?** includes results on three basic modifications to improve the performance of the FK-dualization procedure. These are producing multiple conflicting assignments in a single iteration, substantially reducing the number of redundancy tests during the execution of FK-B, and using a memoization technique to speed up dualization. We showed that each improvement alone produces a substantial speed-up, and in combination, they result in an order of magnitude speed gain in clock-time relative to a naive (unoptimized) implementation.

Here we extend that work by introducing three additional improvements that speed up the FK-B dualization algorithm. Briefly, these are choosing the splitting variable based on information learned in the previous stages, choosing whether to first set a variable to *true* or *false* when the the variable is almost equally frequent in both the CNF and DNF, and shrinking the CNF and DNF during dualization. The first two improvements are heuristics that use previous information to make a decision at the current state, while the third one is an exact test, but only applies in a special case. Our results show that these modifications further improve the performance of the FK-B dualization algorithm, with the splitting variable heuristic showing a significant impact on the most largest computations.

1.1 Definitions

Let $n \in \mathbb{N}$ 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 using any NOTs (negations). We denote the negation of any $x \in \mathcal{B}$ by $\bar{x}$.

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 $x = (x_1, x_2, \ldots, x_n) \in \mathcal{B}^n$, 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

$$f = \bigvee_{j=1}^{m} M_j, \text{ where } M_j = \bigwedge_{i \in T_j} x_i \tag{2}$$

for a collection of m sets $T_1, T_2, \ldots T_m$.

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 a *prime implicant* of f and m is called the *size* of f. In this case, the point $x \in \mathcal{B}^n$ defined by

$$x_i = \begin{cases} 1 & \text{if} \quad i \in T_j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

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 $y < x$ means that $y \leq x$ 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

$$f = \bigwedge_{j=1}^{m} C_j, \text{ where } C_j = \bigvee_{i \in S_j} x_i \tag{4}$$

for a collection of m sets $S_1, S_2, \ldots S_m$.

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

$$x_i = \begin{cases} 0 & \text{if} \quad i \in S_j \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

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 $\{i \in \{1, 2, \ldots, n\} \mid x_i = 1\}$.

We focus on two related problems for monotone Boolean functions, duality and dualization:

1. **Duality**: are two monotone Boolean functions defined by a DNF and a CNF equivalent?

2. **Dualization**: compute the CNF equivalent to a given monotone DNF.

These two problems can be easily transformed into one another, as we explain below. 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 **????** and references therein.

The algorithm with the best known worst-case performance guarantee for the dualization of a monotone Boolean function f, called the FK-B algorithm, has incremental quasi-polynomial running time **?**. More precisely, starting from a description of f in DNF, each iteration obtains an additional clause or verifies duality of the equivalent CNF, in $N^{o(logN)}$ time, where N is the total size of the DNF and the current, possibly incomplete, CNF.

1.2 Application of the Dualization Problem to Metabolic Network Structures

The dualization problem can be mapped to a variety of problems. The problem of interest in our application is that of analyzing the structure of a metabolic network model by finding its Elementary Flux Modes (EFMs) and Minimal Cut Sets (MCSs). The problem of finding the smallest size EFM or MCS in a metabolic network is NP-hard **?**, while the problem of finding all the EFMs or all the MCSs has an unknown complexity status, and can only be solved in reasonable time for small or medium-size metabolic networks **??????**.

In the context of a metabolic network model M, the monotone Boolean function f is defined on the characteristic vectors of 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 **??**. In the experimental results section we apply the FK-B algorithm with the proposed improvements to metabolic network models and study the impact these improvements have on its performance in generating the MCSs based on the pre-computed set of EFMs.

1.3 Some details of the FK-B dualization Algorithm

Algorithms 1 and 2 show the FK dualization and FK-B duality checking procedure respectively, following the presentations of **?** and **?**.

```
Algorithm 1: Fredman-Khachiyan Dualization
  Input: A monotone Boolean function $f$ on $\mathcal{B}^n$ expressed by its complete
         DNF $D$.
  Output: The complete CNF of $C = D^d$.

  Function: $FK$-Dualization($D$)
  $C = \emptyset$;
  Call $FK$-$B$ on the pair $(C, D)$;
  if the returned value is $\emptyset$ then
      return C;
  else
      let $X^* \in \mathcal{B}^n$ be the point returned by $FK$-B;
      compute a maximal false point of $C$, say $Y^*$, such that $X^* \leq Y^*$;
      $C = C \wedge \bigvee_{j \in supp(\overline{Y^*})} x_j$;
      go to Step 3;
  end
```

Algorithm 1 starts with an empty CNF, called C, and a complete DNF, called D. In a loop, it checks the equivalence of C and D using the FK-B algorithm. This algorithm either certifies duality, in which case the dualization is complete and the CNF is returned, or it returns a *conflicting assignment (CA)*, that is, an assignment X^* on which the CNF and the DNF take different values, i.e. either $CNF(X^*) = 1$ and $DNF(X^*) = 0$, or $CNF(X^*) = 0$ and $DNF(X^*) = 1$. It then identifies a maximal false point Y^* greater than X^*, and adds its complement to the current CNF as a new clause.

Algorithm 2, FK-B, takes two Boolean functions in the form of one CNF and one DNF, and checks if the inputs are equivalent for all possible Boolean assignments via a recursive approach. If they are not equivalent, it returns a conflicting assignment. The first step in this algorithm is to remove redundant clauses from both the CNF and the DNF, which is accomplished by comparing every pair of clauses c, c' separately in the CNF and the DNF, and eliminating c' whenever $c \leq c'$. This can be done in quadratic time in a direct way, and probably not much faster, see for example **?**.

Line 2 checks three necessary conditions for a CNF and DNF to be equivalent. These are:

1. Existence of a non-empty intersection between every clause in CNF and every monomial in DNF; if this condition is violated, a conflicting assignment is a monomial m from the DNF that does not intersect some clause of the CNF.

2. The presence of exactly the same variables in the CNF and the DNF; if this condition is violated, with x being a variable in the DNF that does not appear in the CNF, a conflicting assignment is obtained by $m - \{x\}$, where m is a monomial of the DNF that includes x. Alternatively, if x is a variable in the CNF that does not appear in the DNF and c is a clause

of the CNF that includes x, a possible conflicting assignment is obtained by $\{V - c\} \cup x$, where V is the set of all variables.

3. The maximum length of a monomial in the DNF is at most the number of clauses in the CNF, and the maximum length of a clause in the CNF is at most the number of monomials in the DNF. In a case that there is a monomial $m \in DNF$ that contains more variables than there are clauses in CNF, if $m' \subset m$ is a proper subset of m satisfying $m' \cap c \neq \emptyset$ for every clause c of CNF, a conflicting assignment is m'. In the other case that there is a clause $c \in CNF$ that contains more variables than number of monomials in DNF, if $c' \subset c$ is a proper subset of c satisfying $c' \cap m \neq \emptyset$ for every monomial m of DNF, the conflicting assignment would be $V - c'$, where V is the set of all variables.

We note that in each case, the conflicting assignment can be found in polynomial time in the length of the CNF and the DNF.

Line 4 addresses the case in which either the CNF or the DNF is very small, and the equivalence can be checked directly via exhaustive search through the tree of assignments in the CNF or the DNF, whichever is smaller. If they are inequivalent the procedure returns a conflicting assignment; otherwise, it returns $\emptyset$.

The recursive part of the algorithm starts from line 6 where a splitting variable is selected; based on its frequency in the CNF and the DNF, the splitting variable is set to either $False$ or $True$, and after that the current CNF and DNF are simplified by fixing this variable assignment and generating a recursive call to the FK-B algorithm on the new, smaller problem. Note that in line 7, 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

$\mu(n) \sim \log n / \log\log n$ is the largest integer k such that $k^k \leq n$. A similar definition applies to C.

Given that the FK-B algorithm, Algorithm 2, returns the first conflicting assignment (CA) that it finds between the given CNF and DNF, computing the dual of a given DNF using Algorithm 1 requires $N_{CNF} + 1$ iterations, where N_{CNF} is the size of the CNF that is dual to the given DNF.

1.3.1 Preprocessing of the DNF to simplify dualization

The DNF, D, is the input in the FK-dualization algorithm 1. In the context of metabolic networks, each monomial and variable in the DNF corresponds to an EFM and a reaction, respectively.

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:

1. Removing any reactions that are not part of any EFMs (also known as blocked reactions **??**), which correspond to unused variables;
2. Removing any reactions involved in all the EFMs (also referred to as essential reactions **??**), adding them as singleton MCSs during post-processing;
3. Collapsing any group of k reactions whose presence/absence patterns in the EFMs are identical (a special case of this is referred to as enzyme subsets **??**) into a single reaction, expanding each of the final MCSs involving this reaction into k copies during post-processing.

The pre-processing and post-processing steps are not necessary and can be omitted – in particular, the same dual function will be generated in the end. However, in some cases these steps, which can be performed quickly, allow us to perform the dualization computation on a reduced network. This gives a faster overall computation, so we routinely use this method.

2 Methods

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

In Algorithm 2, the first step (line 2) removes redundancy in both the CNF and the DNF. Redundancy removal involves an all-pairs comparison of the clauses (the monomials) in the CNF (the DNF) and removes any supersets found. In logic, removing redundancy is equivalent to applying the absorption rule to simplify the Boolean function. While there are algorithms that slightly improve the asymptotic running time, say by a log factor, not practical improvement in the quadratic running time is known, and it may be that none exists **?**.

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 called 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 clauses (monomials) in the CNF (DNF) being removed, which cannot generate any additional redundancy.

This is implemented using two binary flags, which are set if the redundancy in the CNF (the DNF) needs to be checked, and cleared otherwise. FKR, the algorithm with reduced redundancy checks, 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 FKR needs to check the redundancy in both the CNF and the DNF.

2.2 Finding Multiple Conflicting Assignments

Given that we can use any conflicting assignment between the current CNF and DNF to compute a new clause in the 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.

To this end, MCAs can be computed in the three situations below without a 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 the DNF and clause c in the 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 assignments are 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 any 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 $A_1, A_2 \neq \emptyset$ because the CNF is non-redundant. 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 $(x, y) \in A_1 \times A_2$, with A_1 and A_2 defined above. If no monomial in D is a subset of $\{x, y\}$ then $\{x, y\}$ is a conflicting assignment.

In all the aforementioned cases, whenever more than one conflicting assignment is found, we return all of them. An issue regarding the MCAs is that sometimes more than one CA can be mapped to a single clause in the CNF. In this case, we use the unique clauses resulting from the MCAs.

2.3 Dealing with Repeated Subproblems

Given that FK-B is a recursive algorithm which is called many times during dualization, certain subproblems are solved very frequently. In this case, memoizing (storing for future retrieval) these subproblems and their solutions via certificate CA's (with dual pairs being characterized by an empty set 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. To implement this idea, we compute the key for a given CNF-DNF pair, prior to calling the FK-B 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-B and store any computed CAs as a new record in the hash table.

As we experimented with different settings in the implementation of this memoization technique, we realized that solving small subproblems, with $|C| \leq 2$ or $|D| \leq 2$, from scratch was faster than storing them in the hash table and retrieving the CAs because the special case in line 4 of Algorithm 2 applies to them. Thus, we do not use hashing on these subproblems in our implementation.

On the other hand, it is challenging to store large subproblems because there are so many of them. This may be reduced to a degree by storing functions only up to symmetry. A Monotone Boolean Function (MBF) $f(x_1, x_2, \ldots, x_n)$ is *equivalent* to another MBF $g(x_1, x_2, \ldots, x_n)$ if there is a permutation $\sigma \in$

S_n such that $f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}) = g(x_1, x_2, \ldots, x_n)$. For example, the function $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$ is equivalent to $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3)$ via the permutation $\sigma = (12)(3)$. We can then try to put the MBF corresponding to the DNF into a *canonical form* for hashing.

Finding the canonical form can be done by simply generating all $n!$ permutations, multiplying the function's representation by a fixed matrix, and scanning through it once, choosing the lexicographically smallest function, as described in **?**. This procedure is doable when $n \leq 6$, otherwise it is difficult due to $n!$ possible permutations. After finding the specific permutation, the same permutation is applied to the CNF, leading to the overall representation used for hashing.

2.3.1 Choosing the Splitting Variable by a Weighting Approach

Selecting the splitting variable in the FK-B algorithm plays a key role in the speed of the algorithm. The splitting variable is the variable with the highest frequent variable available in the CNF and the DNF, with ties broken arbitrarily. Here, we present a another approach to choosing the splitting variable using a weighting scheme. The means the worst case analysis of Fredman and Khachiyan no longer applies, but in practice it may well reduce the number of FK calls required.

In this approach, each variable x_i is associated with a weight w_i and a depth d_i. The weight w_i is an estimate of the suitability of variable x_i to be chosen as the splitting variable, and the depth d_i is the smallest recursion depth at which variable x_i has been selected as the splitting variable during the FK-dualization procedure so far. The initial values for w_i and d_i are 100 and ∞ for each i, respectively.

The depth value d_i is updated during a dualization iteration, when the variable x_i is selected as the splitting variable. When this happens, if the current depth is d^*, d_i is updated to $\min(d_i, d^*)$. On the other hand, the weight w_i is updated between dualization iterations according to Algorithm 3. Briefly, the depths are normalized, the current weights are multiplied by the normalized depths, and then they are themselves normalized to form a discrete probability distribution.

In this approach, the splitting variable in each iteration of FK-dualization is chosen at random from the distribution defined by the weights; a higher weight corresponds to a higher probability of being selected as the splitting variable.

2.3.2 Choosing the Order of Settings for a Non-μ-frequent Splitting Variable

In Algorithm 2, if the chosen splitting variable is not μ-frequent [1] in either the CNF or the DNF, the splitting variable is first set to *False*, and if a conflicting assignment is not found, it is set to *True*. There is no theoretical reason behind this order for the settings, and we now present two heuristic approaches for deciding which of the two possible orders can lead to a conflicting assignment faster.

Both approaches make decisions based on the history of setting a variable to *True* or to *False* first in the previous iterations of FK-dualization. The first approach considers the variable's entire history, i.e. the values assigned to it in all the conflicting assignments found so far, and decides to first set the splitting variable to the value found in the *majority* of previously identified conflicting assignments. The second approach makes the decision in the exact same way, but only considers the conflicting assignments found in the $K = 5$ most recent iterations of FK-dualization.

[1] As mentioned earlier in page 5, 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.

Note that in the both approaches, if there is no history of assignments to a variable, the default order, first setting *False*, then setting *True*, is chosen.

2.3.3 Shrinking the CNF and the DNF in the Intermediate Steps of FK-B

In the intermediate steps of running the FK-B algorithm on a CNF and a DNF, it frequently happens that a variable appears as a singleton in the CNF, and also appears in all the monomials in the DNF. Alternatively, it can happen that a variable appears as a singleton in the DNF, and also appears in all the clauses in the CNF. In such a situation, the variable in question can be removed from both the CNF and the DNF without affecting equivalence (i.e. the original DNF and CNF are equal if and only if the reduced ones are). This operation reduces the problem size and lowers the number of recursive calls to FK-B.

This shrinkage step can be performed as a pre-processing step in the FK-B algorithm, before checking the three necessary conditions for equivalence.

3 Experimental Results

As mentioned in Section 1.2, the dualization problem can be used to solve the problem of finding the MCSs given the EFMs in a metabolic network model. Here, we use the metabolic models available in the BioModels database[2] to assess the proposed algorithms. To this end, we selected 19 small and medium-size models; their characteristics are shown in Table 1.

To prepare these models for the application of our dualization algorithm, we performed the following steps:

1. Parsing the biological models using the SBML parser in *MATLAB* to obtain a stoichiometric matrix containing the metabolites as rows and the

[2]`http://www.ebi.ac.uk/biomodels-main/publmodels`

reactions as columns;

2. Applying EFMTool **?** or FluxModeCalculator **?** to extract the EFMs into a matrix;

3. Converting the matrix into a binary one by setting all non-zero values to one, and preprocessing it using the steps outlined in Section 1.3.1; the resulting matrix is used as the input DNF for the dualization problem in order to find the MCSs, corresponding to the CNF.

Different experiments have been designed to elucidate the efficiency of the proposed modifications to the original FK algorithm, individually and in combination. For this purpose, we used various metrics to compare them to the original FK algorithm. These are:

1. Backtrack count: Quantifies how many recursive calls to the FK-B algorithm return no conflicting assignment. This count decreases with better splitting variable selection strategies.

2. Backtrack length: Presents the maximum depth of the recursive calls that return no conflicting assignment. This length decreases with better splitting variable selection strategies as well.

3. Iteration count: Quantifies the number of recursive calls to the FK-B algorithm. This count decreases with improved search strategies, and many of our heuristics may contribute to this.

4. Hash table hits: Quantifies the number of successful hash queries to bypass additional calls to the FK-B algorithm (used when the CNF and the DNF have size at least 3). This count increases when the same (or, with option C, equivalent) subproblem is solved multiple times.

5. Node count: Quantifies the total number of variable settings required to find all the conflicting assignments. This count decreases with a faster identification of the conflicting assignments.

We ran experiments using various combinations of the ideas in Section 2. We reference computational results using short codes to indicate the methods in question. These use the letters F, W, S, H, C, O and R, which denote the following:

F : The baseline **F**redman-Khachiyan algorithm as presented in Algorithm 2 with reduced redundancy tests, multiple conflicting assignments and using the most frequent variable as the splitting variable. Ties are broken randomly;

W : The splitting variable is chosen using the **w**eighting approach;

S : The CNF and the DNF are **s**hrunk during the intermediate steps of running FK;

H : A **h**ash table is used for small subproblems;

C : The **c**anonical form of the CNF and the DNF is stored in the hash table;

O : A heuristic is used to find the optimal **o**rder of settings for non-μ-frequent splitting variables;

R : The $K = 5$ most **r**ecent splits, not the full history, are used in the order heuristic (O) above.

We denote experiments that use several of these ideals by concatenating initials. The first symbol will be F when weighting is not used, and W when it is. From there, symbols are appended to indicate the additional ideas added, so for instance FHC does not use weighting, but does use hash tables and in particular with canonical forms. According to this scheme, the set combinations available for testing are denoted: F, FH, FHC, FO, FOR, FS, W, WH, WHC, WO, WOR and WS.

Now we present the experiments we carried out and discuss their results. We note that the FK-dualization procedure can include some randomization, for instance in tie-breaking. We use randomness only in the initialization, using a fixed *seed* to make the results reproducible. Changing this seed may alter the results, but our sensitivity analysis suggests that there is no significant impact on the relative contributions of each modification (data not shown).

We remark that in our previous study **?**, we gave clock-time comparisons for algorithms F, H and some others as compared to a naive implementation of FK-B. There were a clear improvement. A comparison of relative clock-time for the additional modifications studied here are included in Section 3.5.

3.1 Reducing the Number of Redundancy Tests

The redundancy test is a key bottleneck of FK-dualization, being called on each input to the FK-B algorithm. In this experiment we measure how much the flags proposed in Section 2.1 reduce the number of redundancy tests.

Figure 1 shows the total number of pairwise comparisons in the redundancy tests performed during FK-dualization. Out of 19 experiments, we saw an average decrease of 45% in the number of comparisons, with a minimum of 22% in BIOMD-162 and a maximum of 69% in BIOMD-107.

Given that this modification significantly reduces the running time of FK-dualization, we only discuss the FK-B algorithm with the modification reducing the number of redundancy tests from now on.

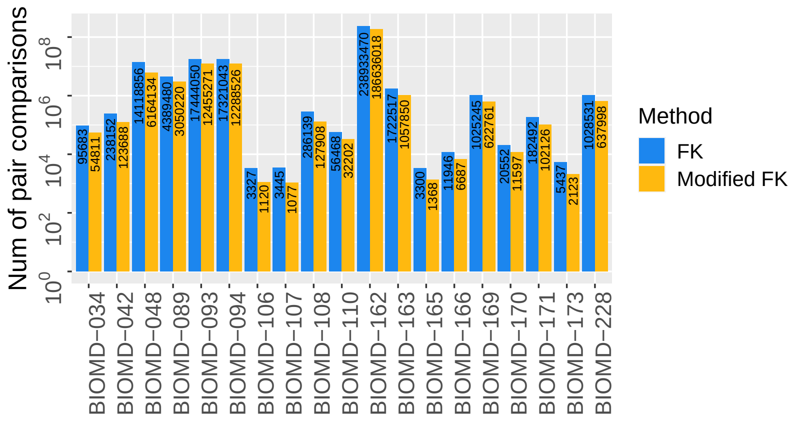


Figure 1: Total number of pairwise comparisons performed in the removing redundancy during FK-dulaization using the original FK algorithm and the modified version of FK which the number of redundancy tests are reduced. In the names of models we replace '0000000' by '-'.

3.2 Finding Multiple Conflicting Assignments

As discussed in Section 2.2, finding multiple CAs at the same time can reduce the iteration count for FK-dualization, each of which may require multiple FK iterations. Figure 2 shows the iteration counts for the original FK algorithm and the variant of FK that finds multiple CAs at the same time. Using this modification reduced the number of required iterations by 10% to 65%. Sizes of the CNF at each iteration are included as Figure 3 in on-line Appendix C.

Similar to the previous section, because this modification is clearly beneficial and independent of other changes, it will become part of the baseline for subsequent experiments.

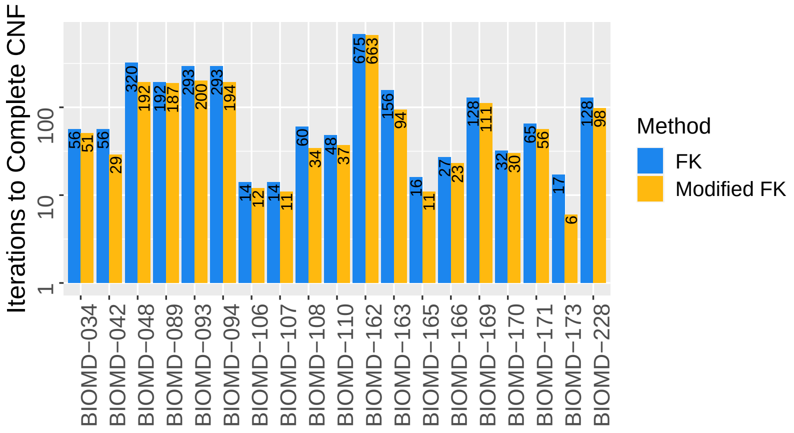


Figure 2: Number of FK-dulaization iterations to build the CNF from the DNF using the original FK algorithm and the modified version of FK which multiple CAs are found at once. In the names of models we replace '0000000' by '-'.

3.3 Comparing FK Variants Based on Splitting Variable Decisions

In this experiment, we compare variants of the FK-B algorithm. Our tables include the results for the F, FH, FHC, FO, FOR, FS and W variants of the algorithm. Additional variants are available in Supplementary File 1.

Figure 3 displays a sample result of this analysis for one model. The remaining models are available in the on-line appendix. The left panels show the size of the CNF at each iteration of the FK-dualization algorithm.

The right panels compare the methods based on the other metrics that we introduced. In these figures, the measurements whose values are zero are shown as bars under the baseline. Since hash tables are only used in the FH and FHC variants, only two bars appear above the baseline for the hash table hits. These variants also perform better on the backtrack count and backtrack length for 15/19 of the models. The iteration count is the lowest for the W variant

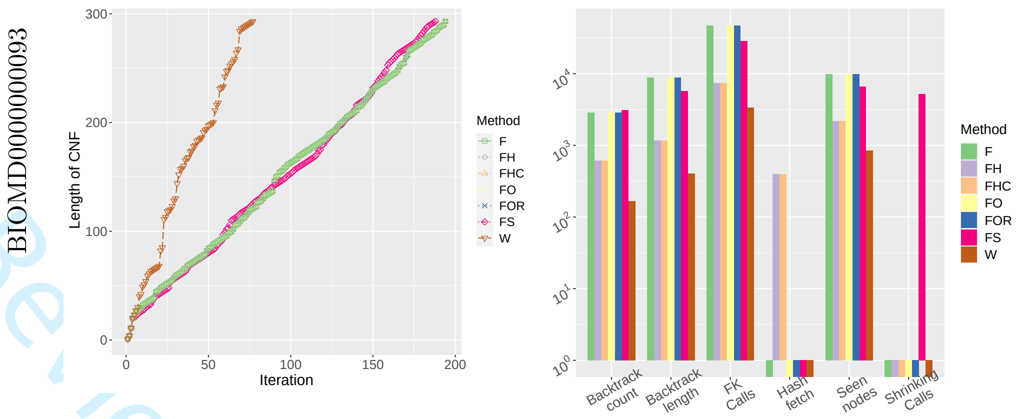


Figure 3: The left figure shows progression of constructing CNF versus iterations in the FK-dualization algorithm. The right figure illustrates the benefit of each improvement in FK-dualization based on five measures.

for 12/19 of the models, and the node counts are strongly correlated with the iteration counts. Interestingly, the FH and FHC variants have the exact same number of hash table hits, meaning that storing the subproblems in canonical form to take equivalence into account provides no additional benefit.

In conclusion, considering both the left and the right panels suggests that W is generally the best at reducing the number of FK calls as well as the number of iterations required to compute the CNF.

3.4 Analysis of the Pareto Frontier

In this experiment, we try to find the *minimal* subsets of modifications to the FK algorithm required to achieve the minimum value of three key metrics - iteration count, backtrack length, and node count - for all of the models (all possible combinations of the modifications that minimize each of these metrics are shown in Supplementary File 2). Tables 2-4 show the results. As it can be seen some modifications, e.g. W (weighting) or S (shrinkage), are seen in several rows in all the tables. Such patterns can give us an idea about the benefit of each modification.

We also created Table 5 to summarize the optimal modifications across all three tables for each model. The last column in this table shows the common modifications, i.e. those needed to achieve an optimal value of each of the metrics. For 14/19 of the models, there is at least one common modification. The two most frequently seen common modifications are S (shrinkage) and H (hash table). However, W (weighting) is also a frequently occurring modification in the rest of the table.

3.5 Comparison of Processing Times

Table 6 presents the processing time for each metabolic model using different FK variants that gives a better understanding on how each improvement can affect processing time. As shown, W variant is the fastest one in 17 models out of 19 models.

Additionally, we tested the F and W variants on four Ecoli metabolic networks including acetate, succinate, glycerol and glucose **??**. Table 7 presents their characteristics and the required time to find the MCSs. As shown, W variant works well in the two biggest networks and significantly reduces the processing times in comparison with F variant. The processing times we obtained here are competitive with the reported processing times for the FK-implementation in **?**, which also solved these problems more quickly via other methods.

4 Discussion

In 1996, Fredman and Khachiyan **?** proposed two novel algorithms, so-called FK-A and FK-B, to identify duality between two monotone Boolean functions whose time complexities are quasi-polynomial time of the input size. Considering the duality test algorithm, if one of the monotone Boolean function is known, its dual function can be produced using the conflicting assignment returning from FK-A or FK-B algorithms in an incremental manner. The computational complexity of such dualization algorithm would still be quasi-polynomial. However, the dualization algorithm is not fast enough in practice when the given monotone Boolean function is of medium or large size.

In this project, we proposed several improvements/techniques to reduce the FK-B running time in practice. These improvements do not affect the theoretical time complexity of the FK-B algorithm, however, they make it possible to use the FK-B to solve medium-to-large-scale dualization problem in practice.

The first improvement is using flags to reduce the number of redundancy tests, pair comparisons, in each recursive call of the FK-B algorithm. The second improvement is about returning multiple conflicting assignment instead of only one. In dualization, this helps to produce more than one monomial in the DNF in each iteration. The third one is about using a hash table and instead of solving repeated subproblems several times, fetch the conflicting assignments. In the fourth improvement, instead of choosing the most frequent variable in both the CNF and the DNF as splitting variable, we choose splitting variable randomly while the chance of a variable being selected is based on its history of acting as splitting variable in previous iterations. In this way, if a variable has been acted as splitting variable in previous iterations and led to a conflicting assignment fast it gets more chance to be selected as a splitting variable in the current point. In the fifth improvement, we focused on a situation where the splitting variable is not μ-frequent in either of CNF or DNF. In this case, instead of first setting the variable to *False* and search for a conflicting assignment(s) and in case of not finding any conflicting assignment(s) setting the variable to *True* and repeat the search again, we consider history of the variable when it has appeared in previous conflicting assignments. It is firstly set to *False* if in majority of previous identified conflicting assignments it is *False*, otherwise it is firstly set to *True* then if any conflicting assignments is found, the other option, i.e. *True* or *False*, is tested. The last improvement is about shrinking the CNF and the DNF in the middle of FK-B recursive calls where one variable appears as a singleton in the CNF/DNF and appears in all monomials/clauses in DNF/CNF. In this case, this variable can safely be removed from both CNF and DNF which results in having a smaller problem to solve.

The proposed modifications have been applied on FK-B algorithm and FK dualization and as an application we have used the modified algorithm in finding minimal cut sets based on given elementary flux modes in metabolic networks. The results show that the proposed improvements can reduce the running time by an order of magnitude on most of the examples.

It is noteworthy to highlight that the proposed modifications/techniques are general and applicable to any problems, e.g. transversal hypergraph generation problem **?**, that can be mapped to Monotone boolean function dualization problem.

5 Code and Data Availability

The data, code and on-line appendices are available on GitHub Repository: `https://github.com/NaSed/Modified_FK_B_Algorithm`.

6 Declarations of Interest

The authors have no financial or personal interests or beliefs that could affect the objectivity of this work.

7 Acknowledgements

Tamon Stephen acknowledges funding from NSERC Discovery Grants (RGPIN-2014-06371 and RGPIN-2019-07134).

Leonid Chindelevitch acknowledges funding from a Sloan Foundation Fellowship and the MRC Centre for Global Infectious Disease Analysis (reference MR/R015600/1), jointly funded by the UK Medical Research Council (MRC) and the UK Foreign, Commonwealth & Development Office (FCDO), under the MRC/FCDO Concordat agreement and is also part of the EDCTP2 programme supported by the European Union.

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

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

Function: $FK$-B$(C, D)$
remove redundant clauses from $D$ and $C$;
if a necessary condition is violated then
    Return conflicting assignment
end
if $\min\{|D|, |C|\} \leq 2$ then
    return $\emptyset$ or the conflicting assignment found by a direct check
else
    choose a splitting variable $x$;
    if $x$ is at most $\mu$-frequent in $D$ then
        $\mathcal{A} \leftarrow FK$-B$(D_1^x, C_0^x \wedge C_1^x)$ // recursive call for $x$ set to $False$ if
        $\mathcal{A} \neq \emptyset$ then
            Return $\mathcal{A}$
        for all clauses $c \in C_0^x$ do
            $\mathcal{A} \leftarrow FK$-B$(D_0^{c,x}, C_1^{c,x})$ // see $\langle 1 \rangle$ below
            if $\mathcal{A} \neq \emptyset$ then
                return $\mathcal{A} \cup \{x\}$
            end
        end
    else if $x$ is at most $\mu$-frequent in $C$ then
        $\mathcal{A} \leftarrow$ FK-B$(D_0^x \vee D_1^x, C_1^x)$ // recursive call for $x$ set to $True$
        if $\mathcal{A} \neq \emptyset$ then
            return $\mathcal{A} \cup \{x\}$
        end
        for all monomials $m \in D_0^x$ do
            $\mathcal{A} \leftarrow FK$-B$(D_1^{m,x}, C_0^{m,x})$ // see $\langle 2 \rangle$ below
            if $\mathcal{A} \neq \emptyset$ then
                return $\mathcal{A} \cup \{m\}$
            end
        end
    else
        $\mathcal{A} \leftarrow FK$-B$(D_1^x, C_0^x \wedge C_1^x)$ // recursive call for $x$ set to $False$
        if $\mathcal{A} = \emptyset$ then
            $\mathcal{A} \leftarrow FK$-B$(D_0^x \vee D_1^x, C_1^x)$ // recursive call for $x$ set to $True$
            if $\mathcal{A} \neq \emptyset$ then
                return $\mathcal{A} \cup \{x\}$
            end
        end
    end
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 by setting all the 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 by setting all the variables in m to $True$.

Algorithm 3: Updating variable weights. This algorithm is called between dualization iterations.

```
Input: A table $W$ with rows indexed by the variables, and two columns:
       1) Weight and 2) Depth.
Output: The updated weight table $W$.
Function: UpdatingWeights($W$)
$W[Weight] = W[Weight] \odot W[Depth]$; // element-wise multiplication;
$W[Weight] = \textbf{\textit{Normalize}}(W[Weight])$;
return $W$;
```

Table 1: Characteristics of models. Metabolites: number of metabolites; EFMs: number of elementary flux modes (monomials in the DNF); $n_r^{<pre}$: Number of reactions (variables) before the preprocessing steps; $n_r^{>pre}$: Number of reactions (variables) after the preprocessing steps; $n_{MCS}^{<post}$: Number of minimal cut sets (clauses in the CNF) before the postprocessing steps; $n_{MCS}^{>post}$: Number of minimal cut sets (clauses in the CNF) after the postprocessing steps.

Model	Metabolites	EFMs	$n_r^{<pre}$	$n_r^{>pre}$	$n_{MCS}^{<post}$	$n_{MCS}^{>post}$
BIOMD0000000034	9	13	22	22	56	56
BIOMD0000000042	15	35	25	20	56	188
BIOMD0000000048	23	63	25	14	320	12960
BIOMD0000000089	16	20	36	28	192	15552
BIOMD0000000093	34	24	46	24	293	2001
BIOMD0000000094	34	23	45	23	293	667
BIOMD0000000106	25	12	32	17	14	512
BIOMD0000000107	14	11	23	13	14	448
BIOMD0000000108	9	50	17	17	60	60
BIOMD0000000110	15	12	22	15	48	864
BIOMD0000000162	32	60	45	20	675	1928934
BIOMD0000000163	16	12	26	21	156	1296
BIOMD0000000165	37	20	30	9	16	576
BIOMD0000000166	3	18	9	9	27	27
BIOMD0000000169	11	17	27	23	128	1536
BIOMD0000000170	7	10	17	15	32	128
BIOMD0000000171	12	16	26	23	65	264
BIOMD0000000173	26	14	26	10	17	4617
BIOMD0000000228	9	13	22	20	128	512

Table 2: Achieving the minimum calls to FK using minimal FK improvements.

Model	FK calls	Successful HashFetch	Backtrack counts	Backtrack length	Seen nodes	Method
BIOMD0000000034	666	1	23	54	249	WH
BIOMD0000000042	253	46	8	12	126	FHS
BIOMD0000000048	2176	0	175	543	1207	W
BIOMD0000000089	1700	0	2	8	269	WS
BIOMD0000000093	2748	0	96	226	695	WS
BIOMD0000000094	2748	0	96	226	695	WS
BIOMD0000000106	63	6	0	0	15	FHS
BIOMD0000000107	37	0	0	0	16	WS
BIOMD0000000108	292	28	8	24	169	FHS
BIOMD0000000110	319	0	1	2	100	W
BIOMD0000000162	8244	0	4	12	853	WS
BIOMD0000000163	790	1	4	11	220	WH
BIOMD0000000165	62	0	3	6	32	WS
BIOMD0000000166	141	0	0	0	48	WS
BIOMD0000000169	868	1	0	0	225	WH
BIOMD0000000170	122	0	2	7	55	W
BIOMD0000000171	229	0	7	18	84	WS
BIOMD0000000173	29	2	1	1	14	FHS
BIOMD0000000228	331	0	4	8	107	WS

Table 3: Achieving the minimum backtracking length using minimal FK improvements.

Model	FK calls	Successful HashFetch	Backtrack counts	Backtrack length	Seen nodes	Method
BIOMD0000000034	678	45	0	0	86	FHS
BIOMD0000000042	253	46	8	12	126	FHS
BIOMD0000000048	2288	322	109	292	1436	FHS
BIOMD0000000089	6769	181	0	0	347	FHS
BIOMD0000000093	2748	0	96	226	695	WS
BIOMD0000000094	2748	0	96	226	695	WS
BIOMD0000000106	63	6	0	0	15	FHS
BIOMD0000000107	37	0	0	0	16	WS
BIOMD0000000108	292	28	8	24	169	FHS
BIOMD0000000110	505	35	1	1	82	FH
BIOMD0000000162	79013	650	0	0	2584	FHS
BIOMD0000000163	2337	86	0	0	176	FH
BIOMD0000000165	64	9	1	1	31	FH
BIOMD0000000166	141	0	0	0	48	WS
BIOMD0000000169	868	1	0	0	225	WH
BIOMD0000000170	232	18	2	4	60	FH
BIOMD0000000171	449	68	6	8	223	FHS
BIOMD0000000173	29	2	1	1	14	FHS
BIOMD0000000228	2530	89	2	2	216	FHS

Table 4: Achieving the minimum number of seen nodes using minimal FK improvements.

Model	FK calls	Successful HashFetch	Backtrack counts	Backtrack length	Seen nodes	Method
BIOMD0000000034	678	45	0	0	86	FHS
BIOMD0000000042	253	46	8	12	126	FHS
BIOMD0000000048	2220	4	187	563	1169	WHS
BIOMD0000000089	1700	0	2	8	269	WS
BIOMD0000000093	2748	0	96	226	695	WS
BIOMD0000000094	2748	0	96	226	695	WS
BIOMD0000000106	63	6	0	0	15	FHS
BIOMD0000000107	37	0	0	0	16	WS
BIOMD0000000108	292	28	8	24	169	FHS
BIOMD0000000110	453	33	3	7	79	FHS
BIOMD0000000162	8244	0	4	12	853	WS
BIOMD0000000163	2337	86	0	0	176	FH
BIOMD0000000165	64	9	1	1	31	FH
BIOMD0000000166	196	13	0	0	46	FHS
BIOMD0000000169	2256	87	0	0	205	FHS
BIOMD0000000170	122	0	2	7	55	W
BIOMD0000000171	229	0	7	18	84	WS
BIOMD0000000173	29	2	1	1	14	FHS
BIOMD0000000228	563	0	2	4	103	W

Table 5: Modifications helped to achieve minimum value for each measurement.

Model	Minimizing FK calls	Minimizing backtracking length	Minimizing seen nodes	Common modification
BIOMD0000000034	WH	FHS	FHS	H
BIOMD0000000042	FHS	FHS	FHS	FHS
BIOMD0000000048	W	FHS	WHS	
BIOMD0000000089	WS	FHS	WS	S
BIOMD0000000093	WS	WS	WS	WS
BIOMD0000000094	WS	WS	WS	WS
BIOMD0000000106	FHS	FHS	FHS	FHS
BIOMD0000000107	WS	WS	WS	WS
BIOMD0000000108	FHS	FHS	FHS	FHS
BIOMD0000000110	W	FH	FHS	
BIOMD0000000162	WS	FHS	WS	S
BIOMD0000000163	WH	FH	FH	H
BIOMD0000000165	WS	FH	FH	
BIOMD0000000166	WS	WS	FHS	S
BIOMD0000000169	WH	WH	FHS	H
BIOMD0000000170	W	FH	W	
BIOMD0000000171	WS	FHS	WS	S
BIOMD0000000173	FHS	FHS	FHS	FHS
BIOMD0000000228	WS	FHS	W	

Table 6: Comparing processing times. The numbers show the time in seconds and in each row the lowest number is bold.

Model	F	FH	FHC	FO	FOR	FS	W
BIOMD0000000034	0.81	1.69	1.7	0.75	0.78	0.63	**0.51**
BIOMD0000000042	2.04	1.85	1.94	2	1.96	**1.22**	1.25
BIOMD0000000048	25.43	12.02	12.15	25.58	25.76	22.18	**7.39**
BIOMD0000000089	14.66	7.82	7.73	14.64	14.71	10.81	**4.71**
BIOMD0000000093	46.87	20.46	19.79	47.46	47.01	38.1	**11.28**
BIOMD0000000094	40.68	20.03	20.16	40.44	40.64	36.57	**10.8**
BIOMD0000000106	0.08	1.45	1.44	0.08	0.08	**0.05**	**0.05**
BIOMD0000000107	0.07	1.45	1.45	0.07	0.07	0.05	**0.04**
BIOMD0000000108	1.71	1.97	2	5.34	2.34	1.08	**0.62**
BIOMD0000000110	0.47	1.64	1.64	0.48	0.48	0.41	**0.19**
BIOMD0000000162	649.62	310.64	310.99	650.85	656.5	683.58	**64.21**
BIOMD0000000163	4.28	3.97	3.86	4.3	4.29	4.49	**1.65**
BIOMD0000000165	0.06	1.48	1.47	0.06	0.07	0.04	**0.04**
BIOMD0000000166	0.14	1.53	1.53	0.14	0.14	0.15	**0.06**
BIOMD0000000169	4.15	3.38	3.38	4.16	4.16	3.04	**1.41**
BIOMD0000000170	0.2	1.51	1.51	0.2	0.2	0.15	**0.06**
BIOMD0000000171	1.26	2	1.99	1.18	1.2	0.9	**0.28**
BIOMD0000000173	**0.03**	1.49	1.49	**0.03**	0.04	0.04	0.05

Table 7: Comparing processing times for Ecoli metabolic networks. In each row the lowest processing time is bold.

Model	# of reactions	# of EFMs	# of MCSs	Processing Time (seconds)	
				F	W
Acetate	21	363	54	2.47	**1.94**
Glucose	34	21592	857	317905.62 (88.3 hrs)	**225271.3 (62.58 hrs)**
Glycerol	28	9479	376	19131.28 (5.31 hrs)	**9005.03 (2.5 hrs)**
Succinate	26	3421	159	**179.43**	220.95

ON-LINE APPENDICES FOR THE PAPER *SPEEDING UP THE STRUCTURAL ANALYSIS OF METABOLIC NETWORK MODELS USING THE FREDMAN-KHACHIYAN ALGORITHM B*

NAFISEH SEDAGHAT, TAMON STEPHEN, AND LEONID CHINDELEVITCH

Appendix A. Summary of Results Based on Splitting Variable Decisions

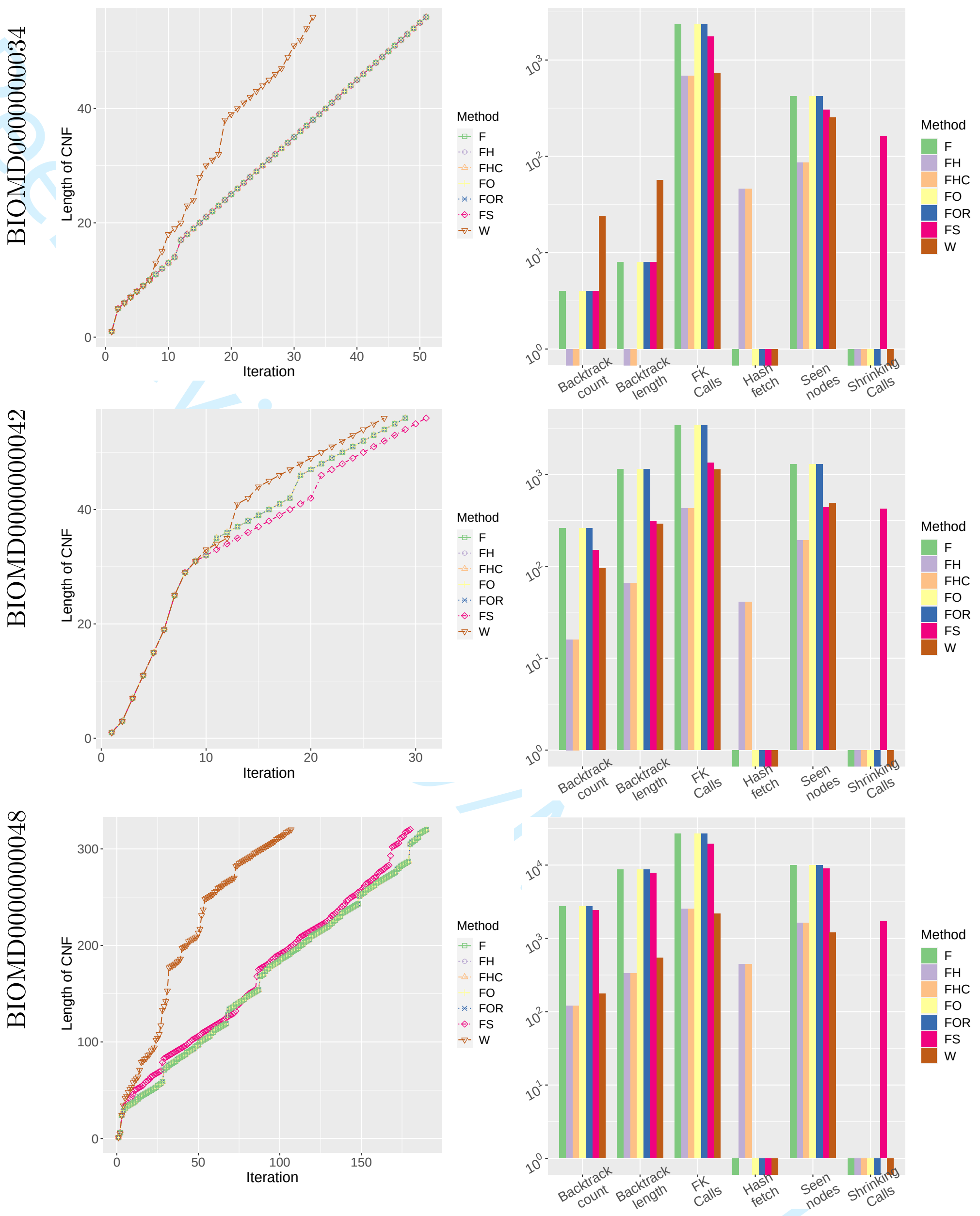


Figure 1. The figures in each row belongs to a model. The figures in the first column show progression of constructing CNF versus iterations in FK-dualization algorithm. The figures in the second column illustrate how beneficial each improvement is in FK-dualization based on five measures: 'Backtrack count' shows how many times wrong branches of tree of assignments have been chosen to go through that finally it had to return to the higher levels; 'Backtrack length' shows how deep it has gone through the wrong branches; 'Seen nodes' shows the number of variables that have been set to either *true* or *false* or both to reach to the conflicting assignment(s); 'FK Calls' indicates to the number of recursive calls to FK algorithm; and 'Hash fetch' shows the number of successful fetches to the hash table in case that keys are not stored in the canonical form and if $|C| < \tau$ and $|D| < \tau$ where $\tau = 3$, the hash table is not used.

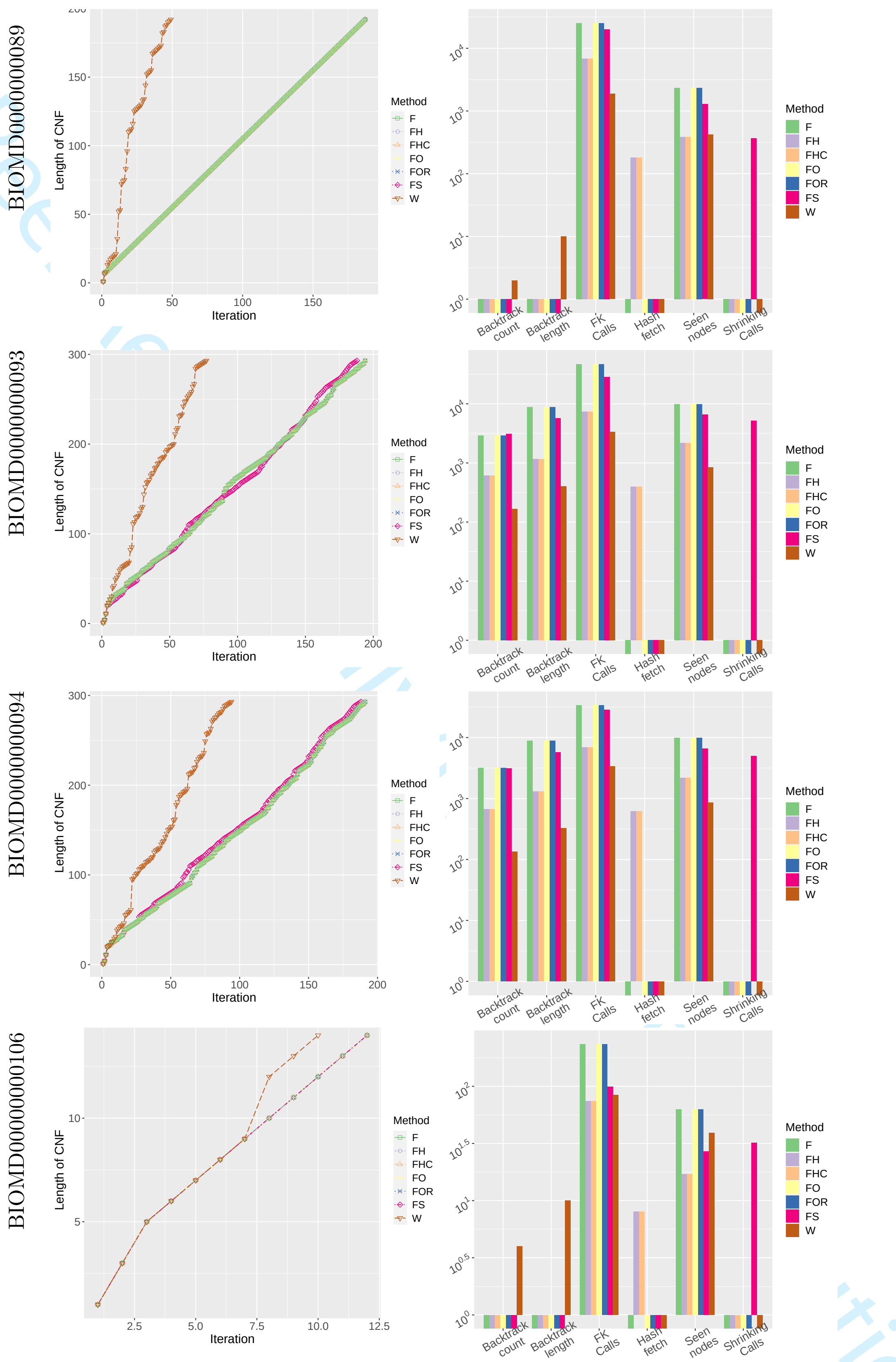


FIGURE 1. Continued.

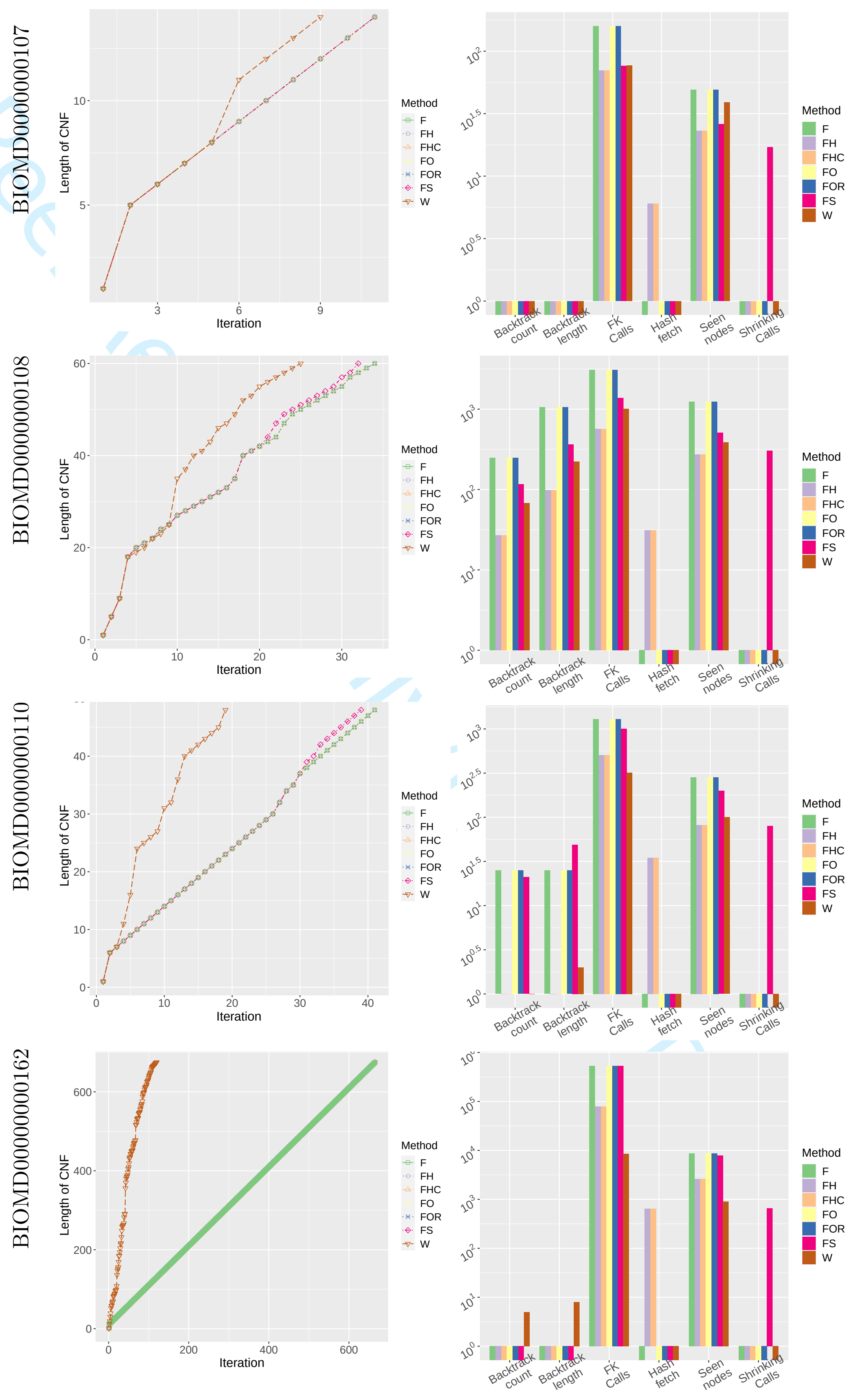


FIGURE 1. Continued.

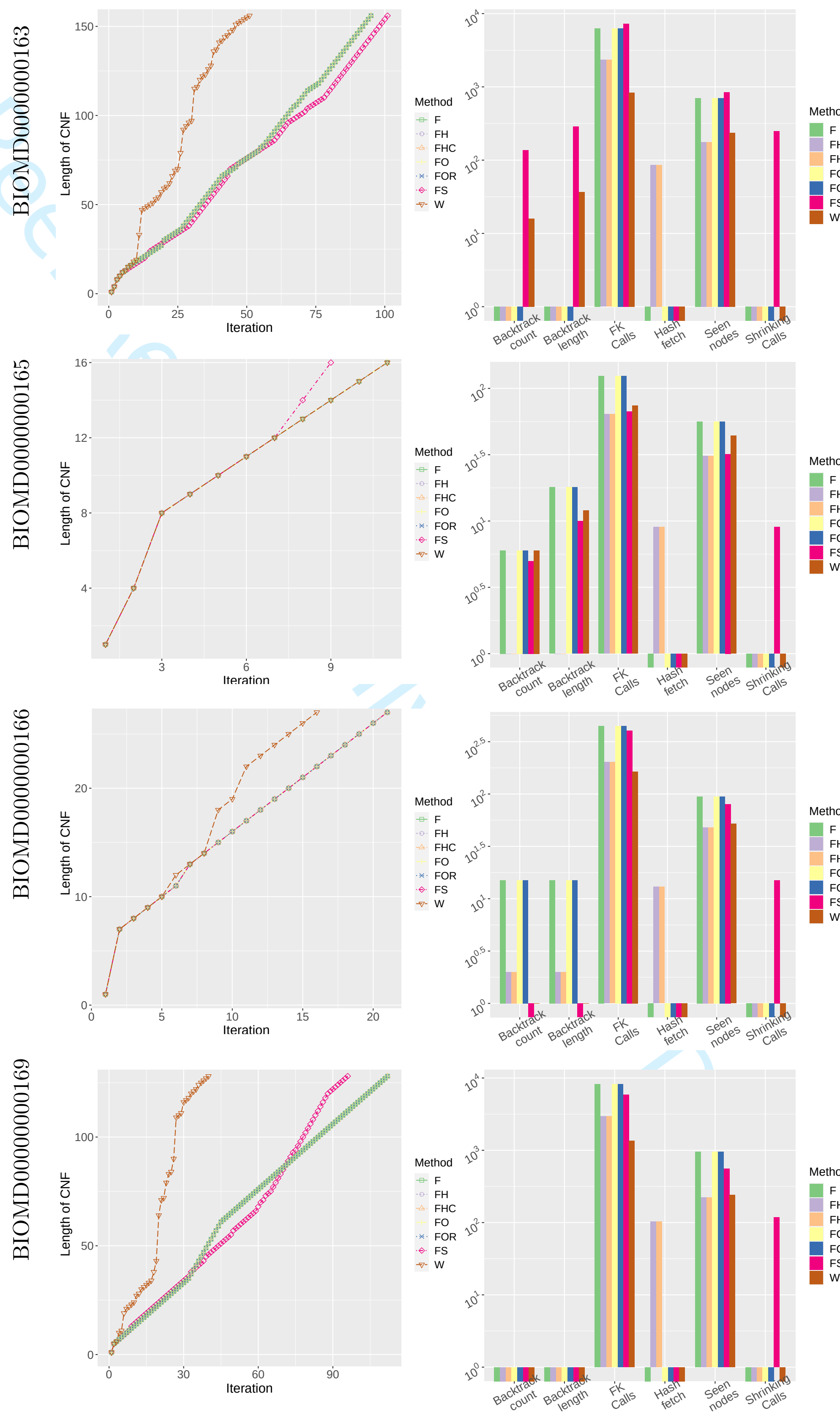


FIGURE 1. Continued.

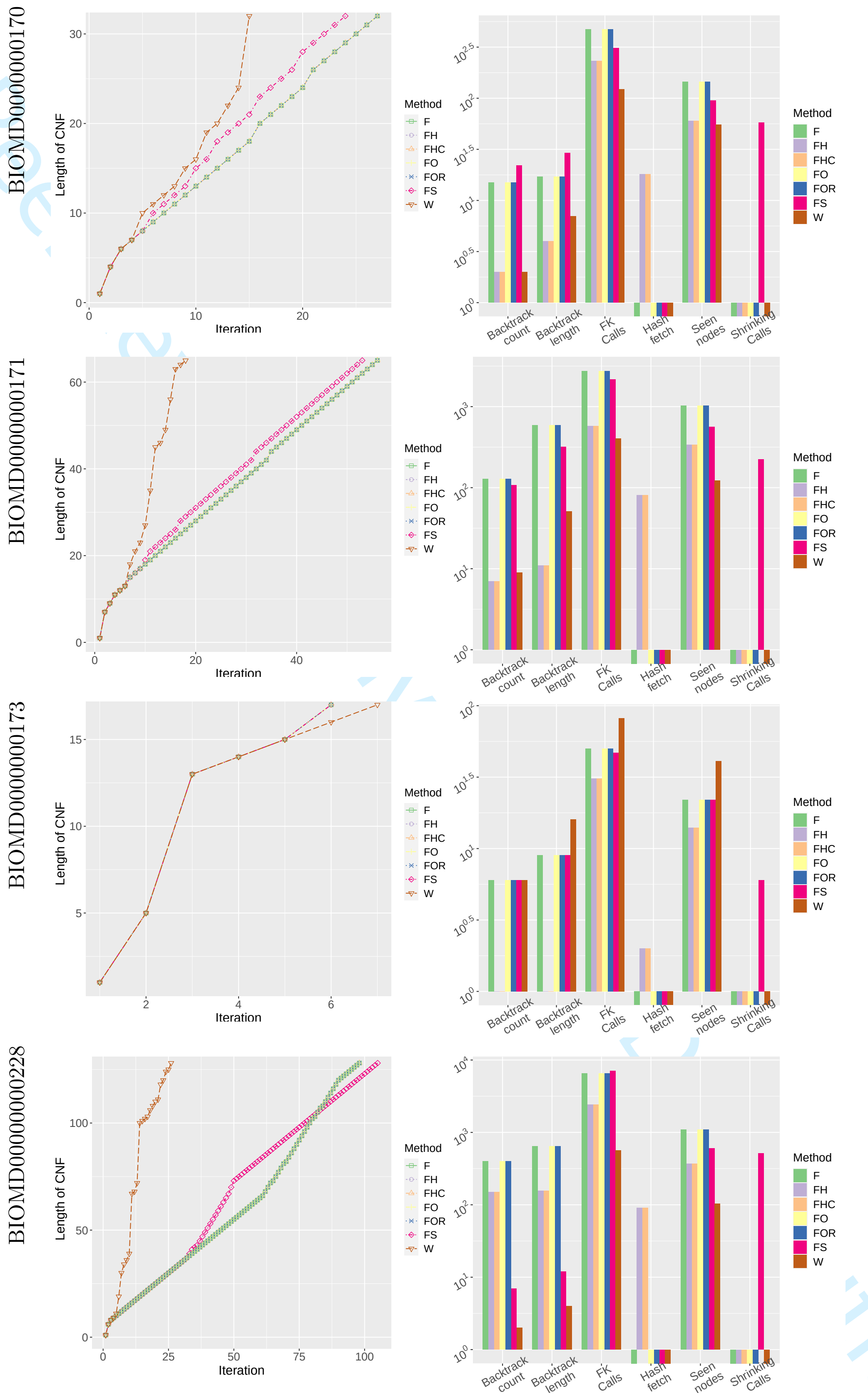
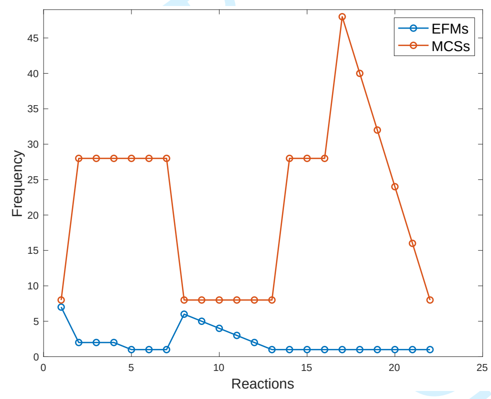


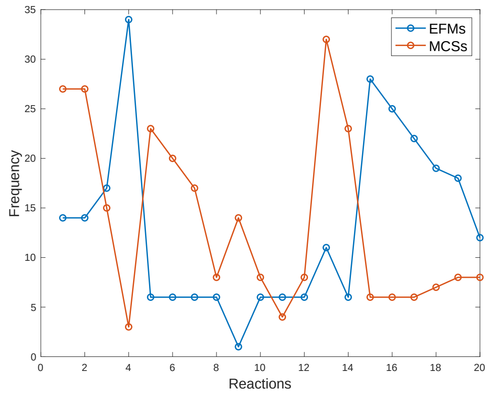
Figure 1. Continued.

Appendix B. Analyzing Reactions in the Biological Models

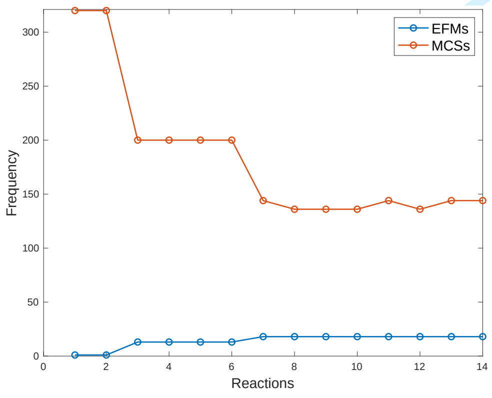
Figure 2 shows the occurrence frequency of reactions in EFMs and MCSs for each model. As shown, in most of the models, occurrence frequency of reactions in the EFMs are less than the MCSs, e.g. *BIOMD0000000034* and *BIOMD0000000048*.



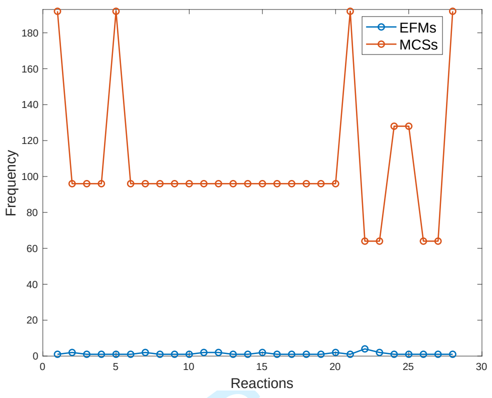
(1) BIOMD0000000034



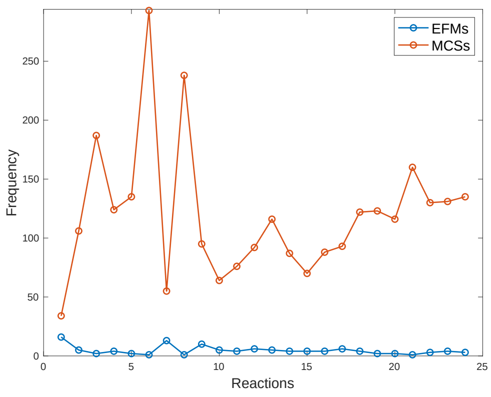
(2) BIOMD0000000042



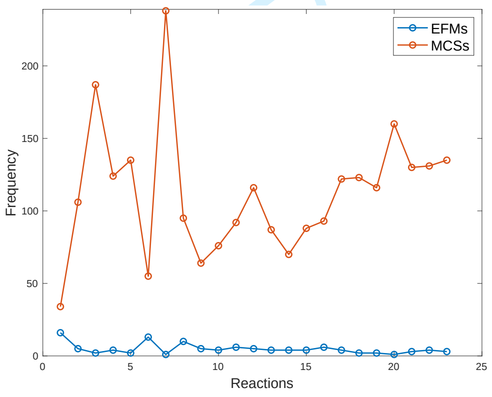
(3) BIOMD0000000048



(4) BIOMD0000000089

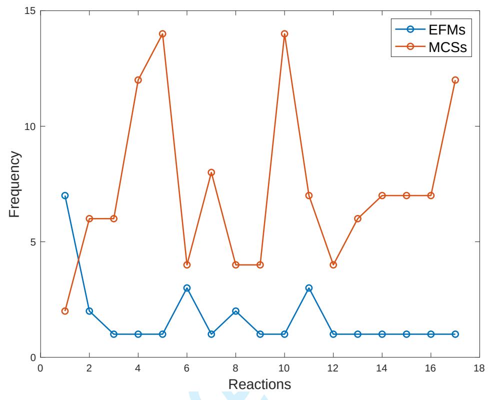


(5) BIOMD0000000093

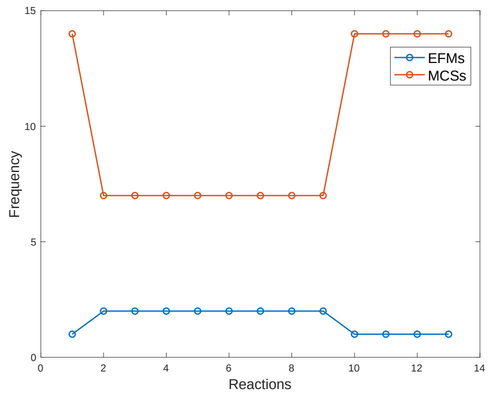


(6) BIOMD0000000094

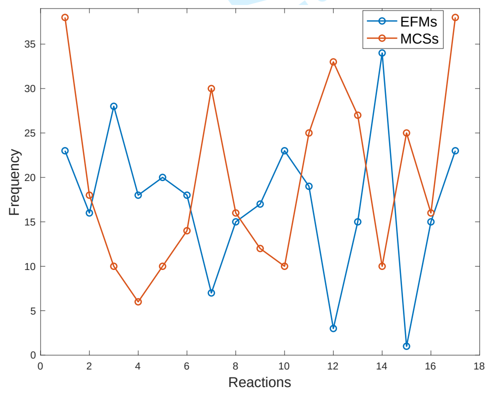
Figure 2. Frequency of occurrence of reactions in EFMs and MCSs.



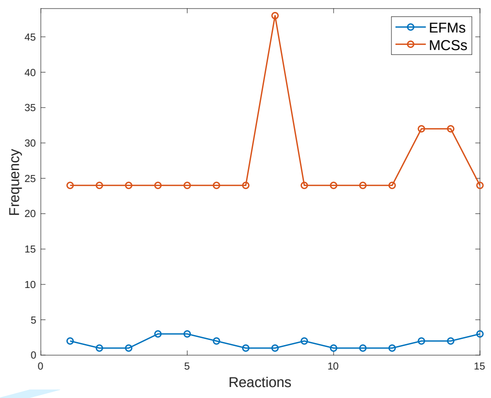
(7) BIOMD0000000106



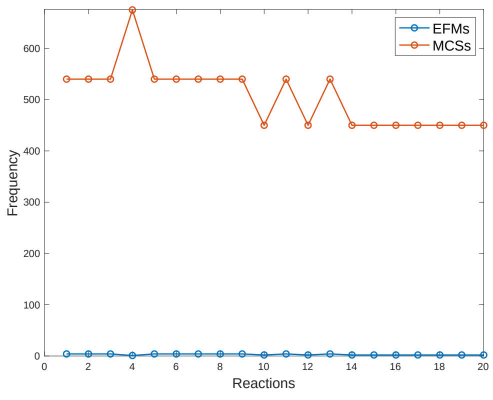
(8) BIOMD0000000107



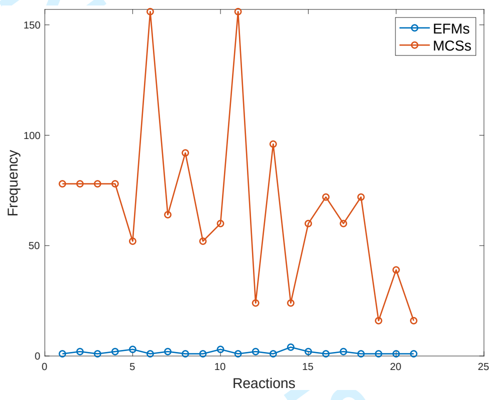
(9) BIOMD0000000108



(10) BIOMD0000000110

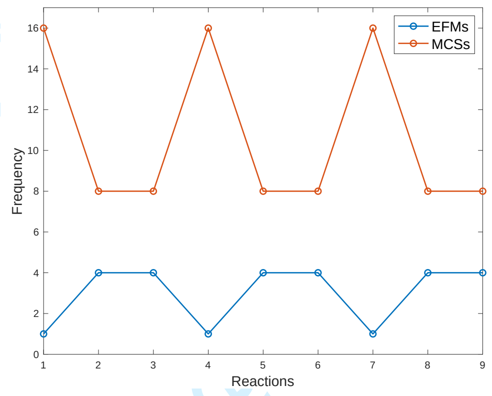


(11) BIOMD0000000162

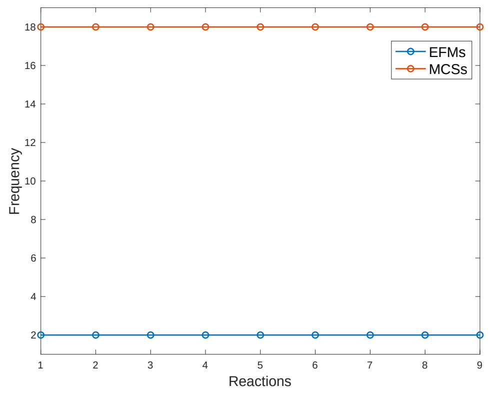


(12) BIOMD0000000163

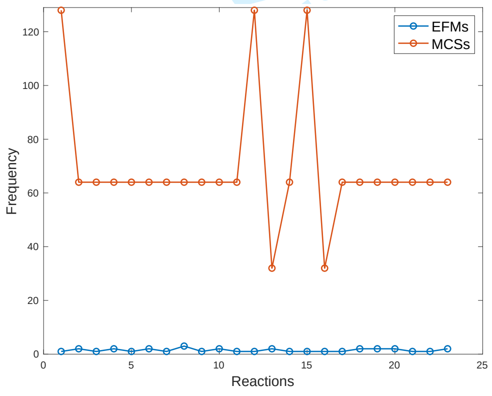
FIGURE 2. Continued.



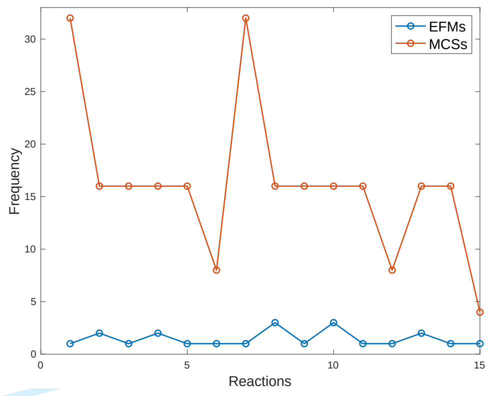
(13) BIOMD0000000165



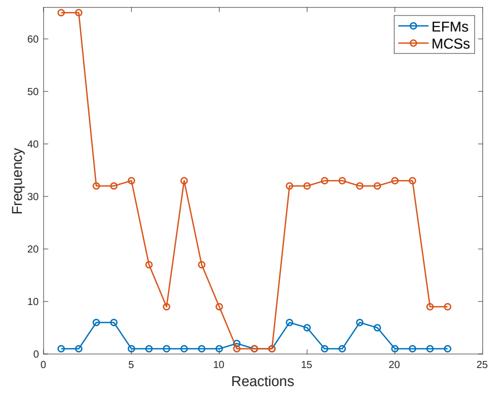
(14) BIOMD0000000166



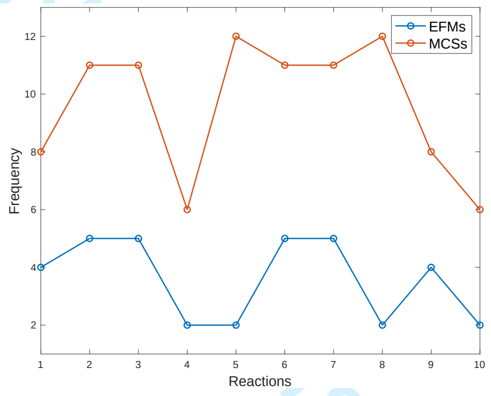
(15) BIOMD0000000169



(16) BIOMD0000000170

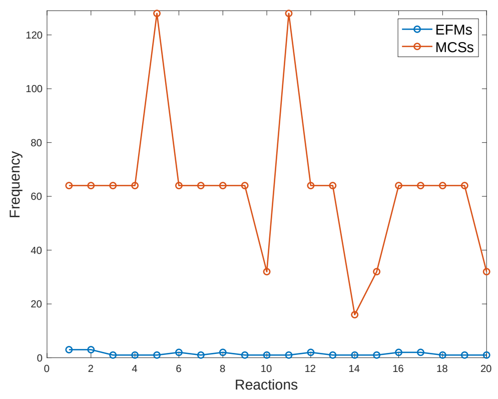


(17) BIOMD0000000171



(18) BIOMD0000000173

Figure 2. Continued.

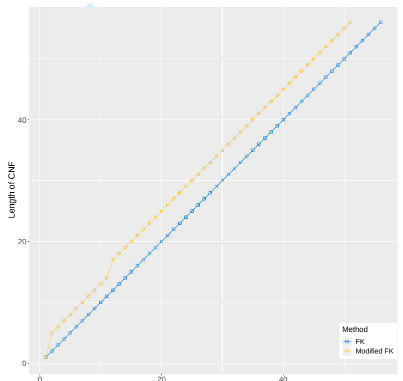


(19) BIOMD0000000228

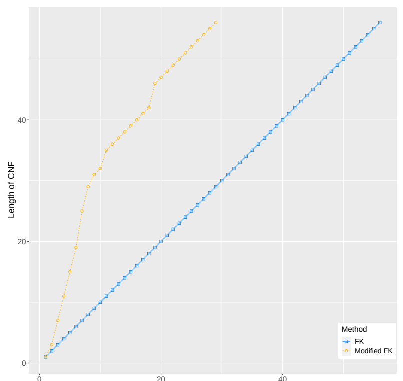
FIGURE 2. Continued.

Appendix C. CNF Completion Progress

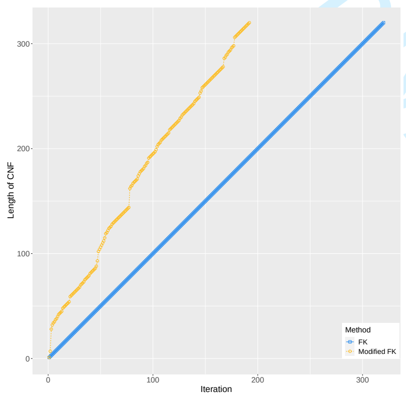
Figure 3 demonstrates the progression of constructing CNF when FK and modified FK in the dualization procedure.



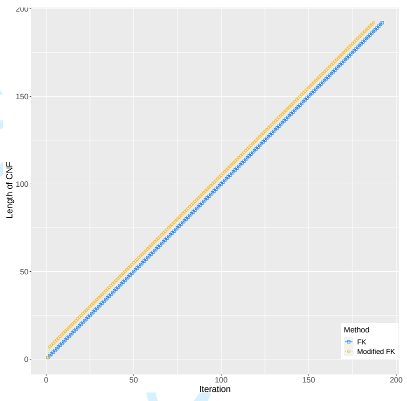
(1) BIOMD0000000034



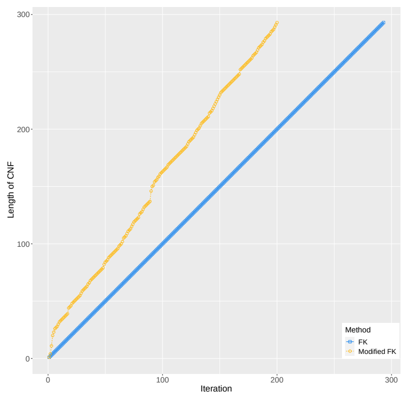
(2) BIOMD0000000042



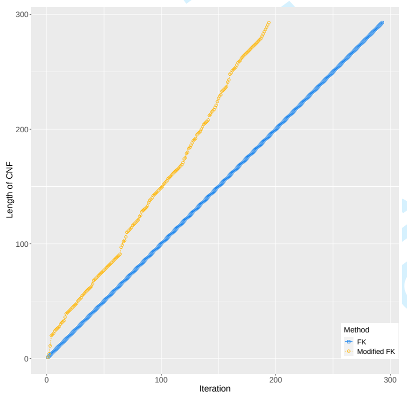
(3) BIOMD0000000048



(4) BIOMD0000000089

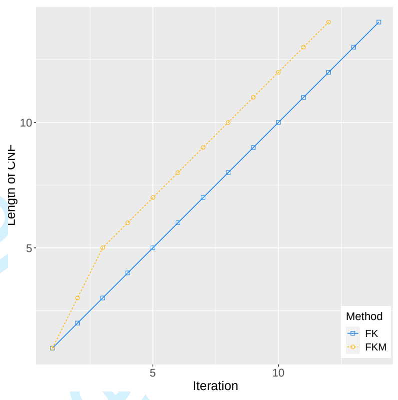


(5) BIOMD0000000093

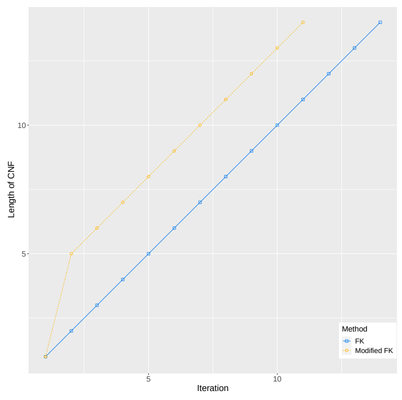


(6) BIOMD0000000094

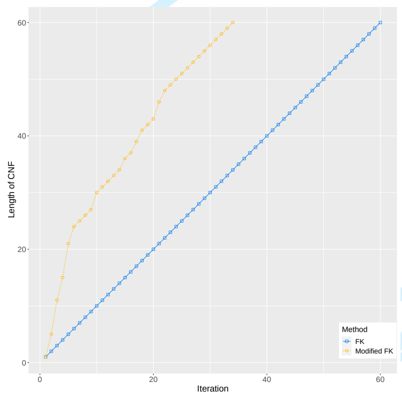
FIGURE 3. Progression of constructing CNF across FK-dualization iterations when FK and FKM have been used for equivalency check between the CNF and the DNF.



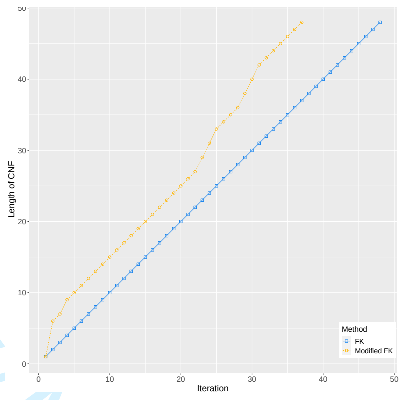
(7) BIOMD0000000106



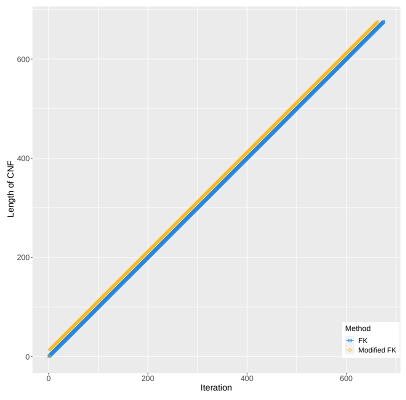
(8) BIOMD0000000107



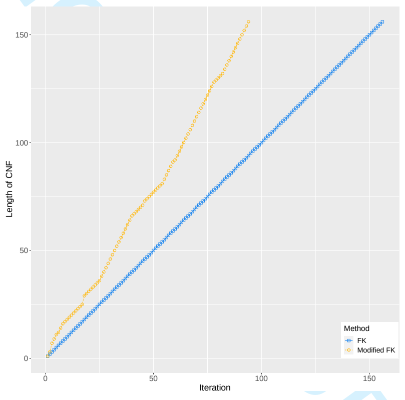
(9) BIOMD0000000108



(10) BIOMD0000000110

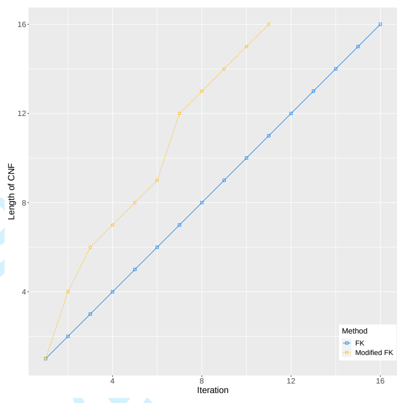


(11) BIOMD0000000162

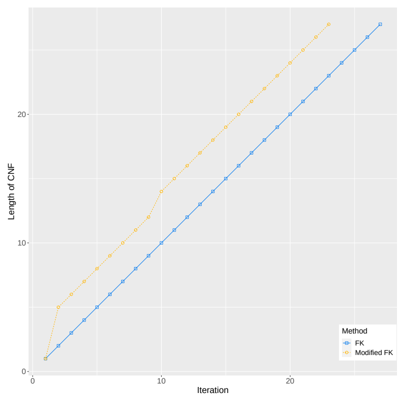


(12) BIOMD0000000163

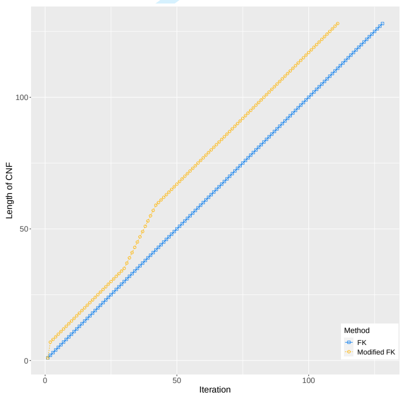
FIGURE 3. Continued.



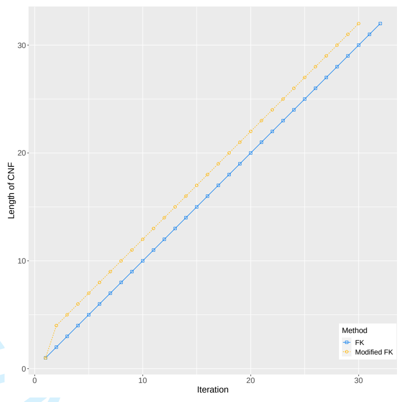
(13)
BIOMD0000000165



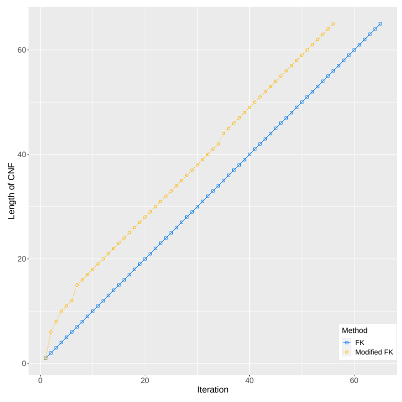
(14)
BIOMD0000000166



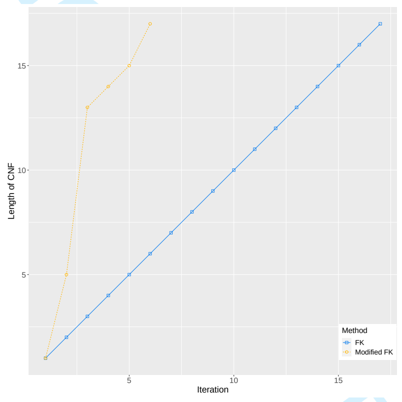
(15)
BIOMD0000000169



(16)
BIOMD0000000170

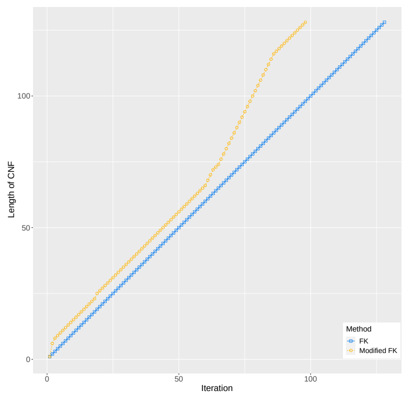


(17)
BIOMD0000000171



(18)
BIOMD0000000173

FIGURE 3. Continued.



(19)
BIOMD0000000228

FIGURE 3. Continued.

School of Computing Science, Simon Fraser University
Email address: nf_sedaghat@sfu.ca

Department of Mathematics, Simon Fraser University
Email address: tamon@sfu.ca

MRC Centre for Global Infectious Disease Analysis, School of Public Health, Imperial College London.
Email address: l.chindelevitch@imperial.ac.uk