Faster Exact MPE and Constrained Optimization with Deterministic Finite State Automata

Filippo Bistaffa
IIIa-CSIC
filippo.bistaffa@iiia.csic.es

Abstract

We propose a concise function representation based on deterministic finite state automata for exact most probable explanation and constrained optimization tasks in graphical models. We then exploit our concise representation within Bucket Elimination (BE). We denote our version of BE as FABE. FABE significantly improves the performance of BE in terms of runtime and memory requirements by minimizing redundancy. Indeed, results on most probable explanation and weighted constraint satisfaction benchmarks show that FABE often outperforms the state of the art, leading to significant runtime improvements (up to 5 orders of magnitude in our tests).

1 Introduction

Graphical models are a widely used theoretical framework that provides the basis for many reasoning tasks both with probabilistic and deterministic information [Dechter, 2013]. These models employ graphs to concisely represent the structure of the problem and the relations among variables.

One of the most important algorithms for exact reasoning on graphical models is Bucket Elimination (BE) proposed by Dechter [1996; 2013], a general approach based on the concept of variable elimination that accommodates many inference and optimization tasks, including most probable explanation (MPE) and constrained optimization. Solving these tasks is fundamental in real-world applications domains such as genetic linkage analysis [Fishelson and Geiger, 2002], protein side-chain interaction [Yanover et al., 2008], and Earth observation satellite management [Bensana et al., 1999]. BE is also a fundamental component—the approximate version of BE [Dechter, 1997] is used to compute the initial heuristic that guides the search—of all the AND/OR search algorithms by Marinescu et al. [2007; 2009; 2014; 2015] that represent the state of the art for exact MPE inference. On the other hand, BE is characterized by memory requirements that grow exponentially with respect to the induced width of the primal graph associated to the graphical model [Dechter, 2013], severely hindering its applicability to large exact reasoning tasks. As a consequence, several works have tried to mitigate this drawback [Dechter, 1997; Bistaffa et al., 2017; Bistaffa and Farinelli, 2018], but none of these approaches really managed to overcome such a limitation. One of the major reasons for such memory requirements is the fact that the functions employed during BE’s execution are usually represented as tables, whose size is the product of the domains of the variables in the scope, regardless of the actual values of such functions. This representation can lead to storing many repeated values in the same table, causing a potential waste of memory and other computational resources.

Against this background, in this paper we propose a function representation specifically devised for exact MPE inference and constrained optimization that, instead of the traditional mapping variable assignment → value, adopts a different approach that maps each value v to the minimal finite state automaton [Hopcroft and Ullman, 1979] representing all the variable assignments that are associated to v.

Compact function representations have already been investigated by different works [Srinivasan et al., 1990; Bahar et al., 1997; Chavira and Darwiche, 2007; Gogate and Domingos, 2013], which have analyzed the potential benefits of the use of different types of decision diagrams from a theoretical point of view. Nonetheless, as noted by Chavira and Darwiche [2007] the effectiveness of these approaches has been limited in practice, because the overhead they incur may very well outweigh any gains. Indeed, no approach based on decision diagrams has ever been shown to outperform the current state of the art, i.e., the AND/OR search algorithms by Marinescu et al. [2014; 2015].

To address the above-mentioned limitations, in this paper we propose the novel use of automata theory techniques to efficiently represent and manipulate functions in a compact way within our approach called FABE. By representing each value only once, and by exploiting the well-known capabilities of automata of compactly representing sets of strings (with a reduction that can be up to exponential with respect to a full table), we significantly improve the performance of BE in terms of runtime and memory requirements. In more detail, this paper advances the state of the art in the following ways:

- We propose a function representation for exact MPE inference and constrained optimization based on finite state automata, which we exploit within FABE.
- Results on standard benchmark datasets show that FABE often outperforms the current state of the art, with improvements of up to 5 orders of magnitude in our tests.
• Results also show that FABE outperforms the structured message passing (SMP) approach by Gogate and Dominigos [2013], i.e., the most recent approach in the literature based on decision diagrams.

• Our work paves the way for an improved version of BE as a key component of AND/OR search algorithms, in which the computation of the initial heuristic can represent a bottleneck [Kishimoto et al., 2015].

This paper is structured as follows. Section 2 provides the background on graphical models and deterministic finite state automata. Section 3 positions our approach wrt existing literature. Section 4 presents our function representation and how we exploit it within FABE. Section 5 presents our experimental evaluation on standard benchmark datasets, in which we compare FABE against the state of the art. Section 6 concludes the paper and outlines future research directions.

2 Background

2.1 Graphical Models

Graphical models (e.g., Bayesian Networks [Pearl, 1989], Markov Random Fields [Lauritzen, 1996], and Cost Networks [Dechter, 2013]) capture the factorization structure of a distribution over a set of variables. Formally, a graphical model is a tuple \( \mathcal{M} = (\mathcal{X}, \mathcal{D}, \mathcal{F}) \), where \( \mathcal{X} = \{X_i : i \in V\} \) is a set of variables indexed by set \( V \) and \( \mathcal{D} = \{D_i : i \in V\} \) is the set of their finite domains of values. \( \mathcal{F} = \{\psi_\alpha : \alpha \in F\} \) is a set of discrete local functions defined on subsets of variables, where \( F \subseteq 2^V \) is a set of variable subsets. We use \( \alpha \subseteq V \) and \( X_\alpha \subseteq X \) to indicate the scope of function \( \psi_\alpha \), i.e., \( X_\alpha = \text{var}(\psi_\alpha) = \{X_i : i \in \alpha\} \). The function scopes yield a primal graph \( G \) whose vertices are the variables and whose edges connect any two variables that appear in the scope of the same function. An important inference task that appears in many real-world applications is MPE. MPE finds a complete assignment to the variables that has the highest probability (i.e., a mode of the joint probability), namely: \( x^* = \arg \max_x \prod_{\alpha \in F} \psi_\alpha(X_\alpha) \). The task is NP-hard [Pearl, 1989]. Another important task over deterministic graphical models (e.g., Cost Networks) is the optimization task of finding an assignment or a configuration to all the variables that minimizes the sum of the local functions, namely: \( x^* = \arg \min_x \sum_{\alpha \in F} \psi_\alpha(X_\alpha) \). This is the task that has to be solved in Weighted Constraint Satisfaction Problems (WCSPs). The task is NP-hard to solve [Dechter, 2013].

To solve the above-mentioned tasks we consider the BE algorithm as discussed by Dechter [2013] (Algorithm 1). BE is a general algorithm that can accommodate several exact inference and optimization tasks over graphical models. In this paper we focus on the version that can optimally solve the above-mentioned MPE and optimization tasks. BE operates on the basis of a variable ordering \( d \), which is used to partition the set of functions into sets called buckets, each associated with one variable of the graphical model. Each function is placed in the bucket associated with the last bucket that is associated with a variable in its scope. Then, buckets are processed from last to first by means of two fundamental operations, i.e., combination (\( \otimes \in \{\prod, \sum\} \)) and projection (\( \lvert \in \{\max, \min\} \)). All the functions in bucket \( p \), i.e., the current bucket, are combined with the \( \otimes \) operation, and the result is the input of a \( \lvert \) operation. Such an operation removes \( X_p \) from the scope, producing a new function \( h_p \) that does not involve \( X_p \), which is then placed in the last bucket that is associated with a variable appearing in the scope of the new function. To solve the MPE (resp. optimization) task, \( \otimes = \prod \) (resp. \( \sum \)) and \( \lvert = \max \) (resp. \( \min \)) operators are used.

The computational complexity of the BE algorithm is directly determined by the ordering \( d \). Formally, BE’s time and space complexity are \( O(r \cdot k w^*(d) + 1) \) and \( O(n \cdot k w^*(d)) \) respectively, where \( k \) bounds the domain size, and \( w^*(d) \) is the induced width of its primal graph along \( d \) [Dechter, 2013]. Hence, it is of utmost importance to adopt a variable ordering \( d \) that minimizes the induced width \( w^*(d) \). The task of computing such ordering is NP-complete, and, for this reason, a greedy procedure is usually adopted to compute a variable ordering of acceptable quality. Several metric (\( \cdot \)) functions have been proposed in the literature. One of the most widely used is MIN-FILL, such that \( \text{metric}(X_i) \) is the number of fill edges for \( X_i \)’s parents in the primal graph [Dechter, 2013].

2.2 Deterministic Finite State Automata

Let \( \Sigma \) denote a finite alphabet of characters and \( \Sigma^* \) denote the set of all strings over \( \Sigma \). The size \( |\Sigma| \) of \( \Sigma \) is the number of characters in \( \Sigma \). A language over \( \Sigma \) is any subset of \( \Sigma^* \). A Deterministic Finite State Automaton (DFSA) [Hopcroft and Ullman, 1979] \( \delta \) is specified by a tuple \( (Q, \Sigma, \tau, s, F) \), where \( Q \) is a finite set of states, \( \Sigma \) is an input alphabet, \( \tau : Q \times \Sigma \to 2^Q \) is a transition function, \( s \in Q \) is the start state and \( F \subseteq Q \) is a set of final states. A string \( x \) over \( \Sigma \) is accepted (or recognized) by \( \delta \) if there is a labeled path from \( s \) to a final state in \( F \) such that this path spells out the string \( x \). Thus, the language \( L_\delta \) of a DFSA \( \delta \) is the set of all strings that are spelled out by paths from \( s \) to a final state in \( F \). It is well known that a general DFSA can accept an infinite language (i.e., a infinite set of strings) [Hopcroft and Ullman, 1979]. In this paper we focus on Deterministic Acyclic Finite State Automata (DAFSA), i.e., DFSA whose corresponding graph is a directed acyclic graph. In contrast with general DFSA, DAFSA only accept finite languages [Daciuk, 2002].
3 Related Work

In recent years, a strand of literature has investigated the use of different algorithms on AND/OR search spaces, progressively showing the effectiveness of these approaches for exact MPE inference and constrained optimization. Specifically, branch-and-bound (AOBB) [Marinescu and Dechter, 2009], best-first (AOBF) [Marinescu and Dechter, 2007], recursive best-first (RBFAOO) [Kishimoto and Marinescu, 2014] and parallel recursive best-first (SPRBFAOO) [Kishimoto et al., 2015] algorithms have been proposed. To this day, RBFAOO [Kishimoto and Marinescu, 2014] and SPRBFAOO [Kishimoto et al., 2015] still represent the state of the art for exact MPE inference, since, to the best of our knowledge, no algorithms have been shown to outperform them in terms of runtime. For this reason, we compare our approach against these two algorithms in Section 5. All these approaches use the standard tabular representation to store functions in memory. In the context of constrained optimization, the only notable approach that tries to reduce the size of tables in memory is the one by Bistaffa et al. [2017], which avoids representing unfeasible assignments during the solution of WCSPs by means of BE.

The task of concisely representing functions for inference has been studied by different works over the years. In the context of this work, the most closely related strand of literature is concerned with the use of decision diagrams to compactly represent knowledge. A decision diagram is a directed acyclic graph that maps several decision nodes to two or more terminal nodes. Different types of decision diagrams exist, depending on the domains of the variables and the types of employed operations. Along these lines, several works have proposed the use of Binary Decision Diagrams (BDDs) [Gogate and Domingos, 2013], Algebraic Decision Diagrams (ADDs) [Bahar et al., 1997], affine ADDs [Sanner and McAllester, 2005], Multi-valued Decision Diagrams (MDDs) [Srinivasan et al., 1990; Chavira and Darwiche, 2007] within variable elimination and/or message passing algorithms for inference.

While it is true that DAFSA and decision diagrams are very closely related, to the best of our knowledge the use of automata theory has never been investigated in the context of concise function representation and inference. In this sense, the use of DAFSA cannot be considered entirely novel (indeed, a DAFSA can be considered equivalent to an MDD). On the other hand, our proposed use of different automata theory techniques for automata compilation [Daciuk, 2002], minimization [Bubenzer, 2011], union [Han and Salomaa, 2008], and determination [Hopcroft and Ullman, 1979] is, as far as we know, entirely novel in the context of inference on graphical models.

Moreover, as noted by Chavira and Darwiche [2007] the effectiveness of approaches based on decision diagrams has been limited in practice, because the overhead they incur may very well outweigh any gains. Indeed, no approach based on any compact representations has ever been shown to outperform the current state of the art, i.e., the AND/OR search algorithms by Marinescu et al. [2014; 2015]. Against this background, we believe that our contribution, i.e., to show that a new approach based on well-known theoretical concepts can still be competitive with the current state of the art, can be very useful for the scientific community.

Given the limited amount of available space, in this paper we decided to compare FABE only with the SMP approach by Gogate and Domingos [2013], since it is, to the best of our knowledge, the most recent and relevant one among the above-mentioned works focusing on decision diagrams.\(^1\) We also remark that, despite the importance of considering this strand of literature, the current state of the art is represented by more recent approaches based on completely different techniques [Kishimoto and Marinescu, 2014; Kishimoto et al., 2015]. Hence, our experimental evaluation in Section 5 is mainly devoted to the comparison against these approaches.

The use of MDDs has also been investigated by Mateescu et al. [2008] within the above-mentioned AND/OR search algorithms, but this approach has been subsumed and outperformed by more recent and advanced approaches based on AND/OR search trees, such as (SP)RBFAOO. For this reason, in Section 5 we only compare with the most recent ones in such a strand of literature.\(^2\)

Lifted probabilistic inference (LPI) [Kersting, 2012] is also concerned with reducing redundancy within probabilistic inference. Specifically, LPI tackles redundancy between different factors, whereas we tackle redundancy inside the same factor. Assessing the effectiveness of the combined approach wrt to the separate ones is a non-trivial research question, which will be considered in future work. Finally, the approach by De Meulenaere et al. [2016] aims at representing compact tables by maintaining generalized arc consistency (i.e., by removing from domains all values that have no support on a constraint) and not by minimizing redundancy due to repeated values, as we explain in the next section.

4 A Novel DAFSA-based Version of BE

All the datasets commonly used as benchmarks for MPE [Kishimoto et al., 2015] and constrained optimization [Marinescu and Dechter, 2009] are characterized by a very high redundancy, i.e., many different variable assignments are associated to the same value in the local functions. Figure 1 shows that the value of redundancy for local functions (defined as \(1 - \frac{\text{number of unique values}}{\text{total number of values}}\)) for all MPE and WCSP instances is always above 80% (except for smaller grid instances).

Furthermore, in probabilistic graphical models, local functions represent probabilities with values in the interval \([0, 1]\), which, in theory, contains infinite real values. In practice, such values are represented by floating point numbers that can only represent a finite amount of values. Thus, while a table \(\psi\) has an arbitrarily large size that is the product of the domains of the variables in its scope, in practice the maximum number of unique values in \(\psi\) is bounded by a parameter that depends on the numerical representation. These remarks motivate the study of a novel concise representation that exploits such a redundancy to reduce the amount of computation. Notice that state of the art approaches for exact inference [Kishimoto et al., 2015] represent functions as full tables, whose size is the product of the domains of the variables in the scope.

\(^1\)Our decision is also motivated by the fact that, to the best of our knowledge, none of these works has an available implementation.

\(^2\)We cannot directly compare with the approach by Mateescu et al. [2008] also because its implementation is not publicly available.
In this paper we propose a way to represent functions by means of DAFSA, as shown in the example in Figure 2. In the traditional way of representing functions as tables, rows are indexed using variable assignments as keys (Figure 2, left).

In contrast, here we propose a novel approach that uses values as keys (Figure 2, right). Formally,

**Definition 1.** Given a function \( \psi \) that maps each possible assignment of the variables in its scope to a value \( v \in \mathbb{R} \cup \{\infty\} \), we denote as \( D(\psi) \) its corresponding representation in terms of DAFSA. Formally, \( D(\psi) = \{(\delta, \delta)\} \), where \( \psi \) is a value in \( \psi \) and \( \delta \) is the minimal DAFSA that accepts all the strings corresponding to the variable assignments that were mapped to \( v \) in \( \psi \). For the sake of simplicity, we do not represent the scope of the function in \( D(\psi) \), as we assume it is equal to \( \text{var}(\psi) \). We label a transition that accepts all the values of a variable’s domain as \( * \). Notice that each \( \delta \) is acyclic because it accepts a finite language [Daciuk, 2002].

**Remark 1.** Given that values are employed as keys in our function representation, it is crucial to ensure the absence of duplicates in such a set of keys, i.e., we must be able to correctly determine whether two values \( v_1 \) and \( v_2 \) are equal. While this is a trivial task in theory, in practice it can be very tricky when \( v_1 \) and \( v_2 \) are floating point numbers representing real values. Indeed, even if \( v_1 \) and \( v_2 \) are theoretically equal, their floating point representations can differ due to numerical errors implicit in floating point arithmetic, especially if \( v_1 \) and \( v_2 \) are the result of a series of operations whose numerical errors have accumulated. To mitigate this aspect, we use a well-known technique for comparing floating point numbers known as \( \epsilon \)-comparison, i.e., \( v_1 \) and \( v_2 \) are considered equal if they differ by a quantity smaller than a small \( \epsilon \). While there exist more advanced techniques of tackling numerical issues related to floating point numbers and their arithmetic [Higham, 2002], they are well beyond the scope of this paper. This should not be considered as an approximation, rather as a standard method to avoid the propagation of numerical errors.

A crucial property of DAFSA is that one path can accept multiple strings, or, in our case, represent multiple variable assignments. In the example in Figure 2, the DAFSA corresponding to \( v_3 \) contains only one path, but it represents both \( \langle 1, 0, 0 \rangle \) and \( \langle 1, 1, 0 \rangle \). By exploiting this property, our representation can reach a reduction in terms of memory that is, in the best case, up to exponential wrt the traditional table representation. We remark that memory is the main bottleneck that limits the scalability of BE, hence reducing its memory requirements is crucial, leading to significant improvements as shown by our results in the experimental section. Finally, our representation allows one to trivially avoid representing unfeasible assignments, similarly to [Bistaffa et al., 2017]. Predicting the size (e.g., the number of states) of a minimal DAFSA accepting a given set of strings remains, to the best of our knowledge, an open problem, since it depends on the common prefixes/suffixes of the input set.

A minimal DAFSA can be efficiently constructed from a set of assignments by using the algorithm described by Daciuk [2002]. Since all the strings accepted by each DAFSA are of the same length (equal to the cardinality of the scope of the function), so are all the paths in the DAFSA. Thus, there is a mapping between each edge at depth \( i \) in each path and the \( i \)th variable in the scope (see Figure 2). Without loss of generality, our representation always maintains the variables in the scope ordered wrt their natural ordering. We now discuss our DAFSA-based version of BE, i.e., its \( \otimes \) and \( \downarrow \) operations.

**4.1 A DAFSA-Based Version of \( \otimes \)**

In order to better discuss our DAFSA-based version the \( \otimes \) operation, let us first recall how this operation works for traditional tabular functions with an example (Figure 3).
The result of the $\otimes$ operation is a new function whose scope is the union of the scopes of the input functions, and in which the value of each variable assignment is the $\otimes \in \{\cdot, +\}$ of the values of the corresponding assignments (i.e., with the same assignments of the corresponding variables) in the input functions. For example, the assignment $\langle X_1 = 0, X_2 = 1, X_3 = 1, X_4 = 0 \rangle$ in the result table corresponds to $\langle X_1 = 0, X_2 = 1, X_4 = 0 \rangle$ and $\langle X_3 = 1, X_4 = 0 \rangle$ in the input tables, hence its value is $v_2 \otimes v_3$. The $\otimes$ operation is closely related to the inner join of relational algebra.

To efficiently implement $D(\psi_1) \otimes D(\psi_2)$ we will make use of the intersection operation on automata [Hopcroft and Ullman, 1979]. Intuitively, the intersection of two automata accepting respectively $L_1$ and $L_2$ is an automaton that accepts $L_1 \cap L_2$, i.e., all the strings appearing both in $L_1$ and $L_2$. In our case, we will exploit the intersection operation to identify all the corresponding variable assignments in $D(\psi_1)$ and $D(\psi_2)$. To make this possible, we first have to make sure that both functions have the same scope, so that corresponding levels in $D(\psi_1)$ and $D(\psi_2)$ correspond to the same variables. We achieve this by means of the ADDLEVELS operation.

**Definition 2.** Given two functions $D(\psi_1)$ and $D(\psi_2)$, the ADDLEVELS operation inserts (i) one or more levels labeled with * in each DAFSA and (ii) one or more variables in the respective scopes, in a way that the resulting scope is $\text{var}(\psi_1) \cup \text{var}(\psi_2)$. Each level and variable is added so as to maintain the scope ordered wrt the variable ordering.

Figure 4 shows an example of ADDLEVELS. The operation of adding one level to a DAFSA $\delta$ has a linear complexity wrt the number of paths in $\delta$. This has to be executed $|D(\psi_1)| \cdot |\text{var}(\psi_2) \setminus \text{var}(\psi_1)| + |D(\psi_2)| \cdot |\text{var}(\psi_1) \setminus \text{var}(\psi_2)|$ times.

Our DAFSA-based $\otimes$ operation is implemented by Algorithm 2. Intuitively, for each couple of values $(v_i, v_j)$, where $v_i$ and $v_j$ are values in $D(\psi_1)$ and $D(\psi_2)$ respectively, we compute the variable assignments associated to their $\otimes$ by computing the intersection $\delta_i \cap \delta_j$ between the corresponding DAFSA $\delta_i$ and $\delta_j$. The result is then associated to the value $v_i \otimes v_j$ in the output function. We maintain only one entry for each value $v_i \otimes v_j$ (see Remark 1 in this respect) by accumulating (i.e., taking the union of) all the DAFSA that are associated to the same value (Line 4).

Union and intersection over DAFSA have a time complexity of $O(nm)$ [Han and Salomaa, 2008], where $n$ and $m$ are the number of states of the input automata. Depending on their implementations, such operations may not directly produce a minimal DAFSA. Nonetheless, DAFSA can be minimized in linear time wrt the number of states with the algorithm by Bubenzer [2011].

**4.2 A DAFSA-Based Version of $\downarrow$**

The $\downarrow \in \{\max, \min\}$ operation effectively realizes variable elimination within the BE algorithm. Specifically, $\downarrow_X \psi$ removes $X_i$ from the scope of $\psi$, and, from all the rows that possibly have equal variable assignments as a result of the elimination of the column associated to $X_i$, it only maintains the one with the max (in the case of MPE, or min in the case of optimization) value. Like $\otimes$, $\downarrow$ is also related to a relational algebra operation, i.e., the project operation. In terms of SQL, $\downarrow_X \psi$ is equivalent to SELECT $\text{var}(\psi) \setminus X_i, \max(\psi(\cdot))$ FROM $\psi$ GROUP BY $\psi(\cdot) \setminus X_i$, in the case of max.

We realize the elimination of the column associated to $X_i$ with the REMOVELEVEL operation, which can be intuitively thought of as the inverse of ADDLEVELS. REMOVELEVEL($D(\psi), X_i$) removes $X_i$ from the scope of $D(\psi)$ and collapses all the edges at the level associated to $X_i$ from all the DAFSA in $D(\psi)$. REMOVELEVEL has similar computational properties wrt ADDLEVELS, so we do not repeat the discussion due to space limitations. Notice that the REMOVELEVEL operation could result in a non-deterministic automaton if the removal happens in correspondence of a branching. Our implementation takes this into account by employing a deterministic algorithm [Hopcroft and Ullman, 1979].

In general, determining an automaton could produce a growth (up to exponential, in the worst case) of the number of states. On the other hand, in all our experiments such a worst-case never happens and the growth factor is, on average, only around 10%. Our results confirm that such a small growth does not affect the overall performance of our approach, which is able to outperform the competitors as described in Section 5.
We then implement the maximization (resp. minimization) of the values as follows. Without loss of generality, we assume that the values \( v_1, \ldots, v_{|D(\psi)|} \) are in decreasing (resp. increasing) order. For each \((v_i, \delta_i) \in D(\psi)\), we subtract from \( \delta_i \) all \( \delta_j \) such that \( v_j \preceq v_i \) in the above-mentioned ordering (i.e., \( v_j \geq v_i \), resp. \( \leq \)). In this way, we remove all duplicate variable assignments and we ensure that each assignment is only associated to the maximum (resp. minimum) value, correctly implementing the \( \delta \) operation. Subtraction over DAFSA has a time complexity of \( O(nm) \) [Han and Salomaa, 2008], where \( n \) and \( m \) are the number of states of the input automata.

Algorithm 3 \( \parallel_X, D(\psi) \)

1. \( D(\psi)^\prime = \text{REMOVELEVEL}(D(\psi), X_i) \).
2. for all \((v_i, \delta_i) \in D(\psi)\) with decr. (resp. incr.) \( v_i \) do
3. \( \delta_i = \delta_i \setminus \delta_{\text{preceq}} \).
4. \( \delta_{\text{preceq}} = \delta_{\text{preceq}} \cup \delta_i \).
5. return \( D(\psi)^\prime \).

We then implement the maximization (resp. minimization) of the values as follows. Without loss of generality, we assume that the values \( v_1, \ldots, v_{|D(\psi)|} \) are in decreasing (resp. increasing) order. For each \((v_i, \delta_i) \in D(\psi)\), we subtract from \( \delta_i \) all \( \delta_j \) such that \( v_j \preceq v_i \) in the above-mentioned ordering (i.e., \( v_j \geq v_i \), resp. \( \leq \)). In this way, we remove all duplicate variable assignments and we ensure that each assignment is only associated to the maximum (resp. minimum) value, correctly implementing the \( \delta \) operation. Subtraction over DAFSA has a time complexity of \( O(nm) \) [Han and Salomaa, 2008], where \( n \) and \( m \) are the number of states of the input automata. Algorithm 3 details our \( \parallel \) implementation.

We remark that both our versions of \( \otimes \) and \( \parallel \) entirely operate on our concise representation, never expanding any function to a full table. We directly employ our \( \otimes \) and \( \parallel \) operations within Algorithm 1. We call our DAFSA-based version of BE “Finite state Automata Bucket Elimination” (FABE). Since the results of our \( \otimes \) and \( \parallel \) operations are equivalent to the original ones, it follows that, as BE, FABE is also an exact algorithm. Finally, we remark that our \( \otimes \) and \( \parallel \) operations can directly be used within the approximated version of BE, i.e., Mini-Bucket Elimination (MBE) [Dechter, 1997].

5 Experiments

We evaluate FABE on standard benchmark datasets\(^4\) for exact MPE inference (i.e., protein, pedigree, grid) [Kishimoto and Marinescu, 2014; Kishimoto et al., 2015] and WCSP (i.e., spot5, mastermind, iscas89) [Marinescu and Dechter, 2009], comparing it with several state of the art approaches. For MPE, we consider RBFAOO [Kishimoto and Marinescu, 2014] as our main competitor since it has been empirically shown to be superior to other sequential algorithms for exact MPE inference, namely AOBB [Marinescu and Dechter, 2009] and AOBF [Marinescu and Dechter, 2007] (see Section 3). We cannot directly compare against the parallel version of RBFAOO, i.e., SPRBFAOO [Kishimoto et al., 2015], because its implementation has not been made public. We discarded the option of re-implementing SPRBFAOO, as it would have probably led to an unfair comparison due to a sub-optimal implementation. Nonetheless, since RBFAOO is also used as a baseline for speed-up calculation in [Kishimoto et al., 2015], in Table 2 we compare our values of speed-up with the ones reported for SPRBFAOO by its authors. As a second competitor we consider toulbar [Hurley et al., 2016], a widely used open-source solver used for exact optimization. We also compare FABE against the SMP approach by Gogate and Domingos [2013] (see associated discussion in Section 3). Since SMP relies on ADDs (which cannot represent non-binary variables natively), we encode non-binary variables using one-hot encoding, following a standard practice. We do not show results comparing FABE against the standard version of BE with tabular functions [Dechter, 2013], since the latter runs out of memory on most of the instances. For WCSPs we also compare against CPLEX and GUROBI, two off-the-shelf solvers for constrained optimization.

FABE’s reported runtimes also include the compilation of automata. Since both FABE and RBFAOO require an ordering \( d \) (see Section 2), we consider this as a pre-processing phase and we do not include its runtime in the results (also because it is negligible wrt the solution phase). For each instance, we compute \( d \) using a weighted MIN-FILL heuristic [Dechter, 2013], and we use the same \( d \) for both algorithms.

We execute RBFAOO with the parameters detailed in authors’ previous work [Kishimoto and Marinescu, 2014; Kishimoto et al., 2015], including cache size and \( i \) parameter. Every other approach is executed with default parameter values. Following [Kishimoto et al., 2015], we set a time limit of 2 hours. We exclude from our analysis all instances that could not be solved by any algorithm in the considered time limit. FABE and SMP are implemented in C++.\(^5\) We employ the implementations of RBFAOO and toulbar provided by the authors. All experiments have been run on a cluster whose computing nodes have 2.50GHz CPUs and 384 GBytes of RAM. As for Remark 1, for FABE we consider \( \epsilon = 10^{-10} \).

Given the large number of instances in MPE datasets, in Table 1 we report the runtimes on the 12 largest instances wrt the number of variables. In Table 2 we report the aggregated results of the speed-ups achieved by FABE wrt other approaches, calculated considering the instances where both algorithms terminate within the time limit.

5.1 Summary of Results

Results confirm that FABE’s performance depends on the degree of redundancy. FABE obtains good performance on the protein and pedigree datasets, achieving speed-ups of \( \sim 1 \)–2 orders of magnitude, and solving a total of 34 instances that RBFAOO could not solve. We also observe that toulbar is superior on a handful of protein instances and on the grid dataset, which is characterized by low redundancy. Results also show that, despite not employing parallelism, FABE’s speed-up on the protein dataset is much higher than the one reported for SPRBFAOO, while it is comparable on the pedigree datasets.

As for WCSPs (detailed results in Table 3), FABE outperforms all competitors on the spot5 dataset, notably achieving a speed-up of 5 orders of magnitude wrt toulbar. On the mastermind dataset, FABE is comparable with toulbar (since both compute solutions in tenths of seconds) except for 3·8·5 and 10·8·3 instances, but it is, on average, better than every other competitor. We also observe that toulbar is clearly superior on the iscas89 dataset.

Finally, FABE consistently outperforms SMP using one-hot encoding, confirming that the use of additional encodings (required by the presence of non-binary variables that cannot


\(^5\)Code available at https://github.com/filippobistaffa/FABE.
We proposed FABE, an algorithm for exact MPE and constrained optimization that exploits our function representation based on DAFSA. Results obtained following an established methodology confirm the efficacy of our representation.

Future research directions include extending FABE to marginal inference tasks and integrating FABE to compute the initial heuristic for AND/OR search algorithms, which, at the moment, use the table-based implementation of BE. The computation of such an heuristic represents a bottleneck for high values of $i$, as acknowledged in [Kishimoto et al., 2015]. A faster version of MBE could represent an important contribution, allowing one to employ more precise heuristics and, thus, improving the performance of the AND/OR search.

### 6 Conclusions

Table 1: Runtime results (in seconds) on 12 largest MPE instances.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Protein</th>
<th>Pedigree</th>
<th>Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Redundancy</td>
<td>96%</td>
<td>85%</td>
<td>64%</td>
</tr>
<tr>
<td>FABE speed-up wrt RBFAOO</td>
<td>58.6 (1%, 11%)</td>
<td>5.5 (0%, 32%)</td>
<td>0.1 (43%, 0%)</td>
</tr>
<tr>
<td>FABE speed-up wrt SMP</td>
<td>1006.5 (1%, 38%)</td>
<td>6.8 (0%, 5%)</td>
<td>4.0 (43%, 70%)</td>
</tr>
<tr>
<td>SPRBFAOO speed-up wrt RBFAOO</td>
<td>~7</td>
<td>~7</td>
<td>~5</td>
</tr>
<tr>
<td>FABE speed-up wrt tourbar</td>
<td>1.0 (1%, 0%)</td>
<td>20.6 (0%, 23%)</td>
<td>0.0 (43%, 0%)</td>
</tr>
</tbody>
</table>

Table 2: Average speed-up results for MPE (top) and WCSP (bottom) instances. For SPRBFAOO we report the same speed-up values reported by the authors [Kishimoto et al., 2015]. Values in parentheses indicate the percentages of instances unsolved by first and second approach.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>protein</th>
<th>pedigree</th>
<th>grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Redundancy</td>
<td>85%</td>
<td>85%</td>
<td>87%</td>
</tr>
<tr>
<td>FABE speed-up wrt RBFAOO</td>
<td>36.9 (0%, 0%)</td>
<td>6.2 (0%, 0%)</td>
<td>0.4 (0%, 0%)</td>
</tr>
<tr>
<td>FABE speed-up wrt SMP</td>
<td>5.8 (0%, 0%)</td>
<td>2.5 (0%, 0%)</td>
<td>5.1 (0%, 0%)</td>
</tr>
<tr>
<td>FABE speed-up wrt tourbar</td>
<td>10615.5 (0%, 50%)</td>
<td>0.4 (0%, 0%)</td>
<td>0.1 (0%, 0%)</td>
</tr>
<tr>
<td>FABE speed-up wrt CPLEX</td>
<td>2.0 (0%, 0%)</td>
<td>7.2 (0%, 0%)</td>
<td>0.8 (0%, 0%)</td>
</tr>
<tr>
<td>FABE speed-up wrt GUROBI</td>
<td>1.2 (0%, 0%)</td>
<td>4.4 (0%, 0%)</td>
<td>0.8 (0%, 0%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>spot5</th>
<th>42b</th>
<th>505b</th>
<th>408b</th>
<th>29</th>
<th>503</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td>FABE</td>
<td>0.26</td>
<td>0.26</td>
<td>0.29</td>
<td>0.09</td>
<td>0.05</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>RBFAOO</td>
<td>13.37</td>
<td>10.27</td>
<td>9.97</td>
<td>5.61</td>
<td>1.37</td>
<td>1.36</td>
<td></td>
</tr>
<tr>
<td>SMP</td>
<td>1.62</td>
<td>1.80</td>
<td>1.60</td>
<td>0.72</td>
<td>0.27</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td>CPLEX</td>
<td>0.35</td>
<td>0.34</td>
<td>0.39</td>
<td>0.37</td>
<td>0.10</td>
<td>0.18</td>
<td></td>
</tr>
<tr>
<td>GUROBI</td>
<td>0.38</td>
<td>0.40</td>
<td>0.41</td>
<td>0.13</td>
<td>0.25</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td>tourbar</td>
<td>&gt; 2 h</td>
<td>&gt; 2 h</td>
<td>&gt; 2 h</td>
<td>0.10</td>
<td>1957.80</td>
<td>0.09</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>master</th>
<th>3·85</th>
<th>10·83</th>
<th>4·84</th>
<th>3·84</th>
<th>4·83</th>
<th>3·83</th>
</tr>
</thead>
<tbody>
<tr>
<td>FABE</td>
<td>247.27</td>
<td>69.30</td>
<td>0.36</td>
<td>0.22</td>
<td>0.10</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>RBFAOO</td>
<td>4.93</td>
<td>3.01</td>
<td>2.96</td>
<td>1.96</td>
<td>0.85</td>
<td>0.68</td>
<td></td>
</tr>
<tr>
<td>SMP</td>
<td>659.42</td>
<td>185.45</td>
<td>0.95</td>
<td>0.43</td>
<td>0.29</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>CPLEX</td>
<td>3.26</td>
<td>0.66</td>
<td>1.26</td>
<td>0.67</td>
<td>1.54</td>
<td>1.28</td>
<td></td>
</tr>
<tr>
<td>GUROBI</td>
<td>1.77</td>
<td>0.66</td>
<td>0.72</td>
<td>0.51</td>
<td>0.97</td>
<td>0.75</td>
<td></td>
</tr>
<tr>
<td>tourbar</td>
<td>&gt; 2 h</td>
<td>&gt; 2 h</td>
<td>&gt; 2 h</td>
<td>0.10</td>
<td>1957.80</td>
<td>0.09</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>iscas89</th>
<th>s1238</th>
<th>c880</th>
<th>s1196</th>
<th>s953</th>
<th>s1494</th>
<th>s1488</th>
</tr>
</thead>
<tbody>
<tr>
<td>FABE</td>
<td>38.50</td>
<td>25.36</td>
<td>73.54</td>
<td>286.43</td>
<td>1.42</td>
<td>1.12</td>
<td></td>
</tr>
<tr>
<td>RBFAOO</td>
<td>1.47</td>
<td>1.17</td>
<td>0.61</td>
<td>0.54</td>
<td>0.41</td>
<td>0.39</td>
<td></td>
</tr>
<tr>
<td>SMP</td>
<td>229.64</td>
<td>146.43</td>
<td>410.96</td>
<td>1464.98</td>
<td>9.78</td>
<td>6.02</td>
<td></td>
</tr>
<tr>
<td>CPLEX</td>
<td>0.19</td>
<td>0.20</td>
<td>0.09</td>
<td>0.19</td>
<td>0.28</td>
<td>0.23</td>
<td></td>
</tr>
<tr>
<td>GUROBI</td>
<td>0.17</td>
<td>0.14</td>
<td>0.09</td>
<td>0.15</td>
<td>0.26</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>tourbar</td>
<td>0.04</td>
<td>0.06</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.07</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Runtime results (in seconds) on WCSP instances.
Ethical Statement

There are no ethical issues.

Acknowledgments

The author was supported by the “YOMA Operational Research” project funded by the Botnar Foundation. The author gratefully acknowledges the computer resources at Artemisa, funded by the European Union ERDF and Comunitat Valenciana as well as the technical support provided by the Instituto de Fisica Corpuscular, IFIC (CSIC-UV).

References


