On the Detection of Commutative Factors in Factor Graphs: Necessary and Sufficient Conditions

Malte Luttermann MALTE.LUTTERMANN@UNI-HAMBURG.DE
Institute for Humanities-Centered Artificial Intelligence, University of Hamburg, Germany

Ralf Möller RALF.MOELLER@UNI-HAMBURG.DE
Institute for Humanities-Centered Artificial Intelligence, University of Hamburg, Germany

Marcel Gehrke MARCEL.GEHRKE@UNI-HAMBURG.DE
Institute for Humanities-Centered Artificial Intelligence, University of Hamburg, Germany

Editors: Gustau Camps-Valls, Manuele Leonelli and Gherardo Varando

Abstract

Exploiting the indistinguishability of objects in a probabilistic graphical model such as a factor graph is key to lifted probabilistic inference algorithms and allows for tractable probabilistic inference problems with respect to domain sizes. A central building block for the exploitation of indistinguishable objects in factor graphs is the identification of commutative factors, i.e., factors whose output values are invariant under permutations of input values assigned to a subset of their arguments. In this paper, we revisit the theoretical foundations underlying the state-of-the-art algorithm to detect commutative factors. Specifically, we show that in its current form, the state-of-the-art algorithm relies on a central theorem that is mistakenly regarded as a sufficient condition to identify commutative factors, while it actually only implies necessary condition. Consequently, the state of the art might, as we show in this paper, deliver incorrect results. To fix the flaws currently present in the state of the art, we prove a slightly modified version of the aforementioned theorem, which serves as a necessary condition to identify commutative factors. Moreover, we present a corrected version of the state-of-the-art algorithm, which keeps its efficiency while ensuring correctness and introduce a complementary algorithm with tighter worst-case bounds.

Keywords: colour passing; factor graphs; lifted inference.

1. Introduction

Reasoning under uncertainty is a core challenge in artificial intelligence, and probabilistic graphical models offer a principled framework for this task by factorising a joint probability distribution into a product of local functions (which we call factors). Computing marginal and conditional distributions—the central inference problem—is, however, intractable in general as the computational cost grows exponentially with the number of random variables (randvars) in propositional models such as Bayesian networks, Markov networks, or factor graphs (FGs) (Cooper, 1990). This intractability even persists for approximate inference (Dagum and Luby, 1993). Lifted inference algorithms exploit the indistinguishability of objects to allow for tractable probabilistic inference problems (i.e., inference problems that are solvable in polynomial time) with respect to domain sizes while maintaining exact answers (Niepert and Van den Broeck, 2014). To perform lifted inference, however, a lifted representation encoding equivalent semantics as the initially given propositional probabilistic model must be constructed. Constructing such a lifted representation requires the

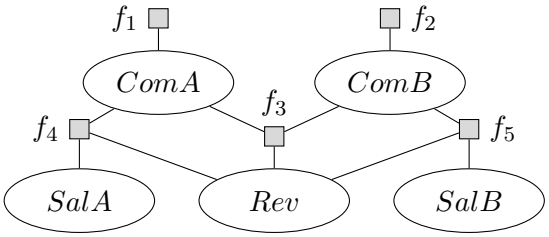
© 2026 M. Luttermann, R. Möller & M. Gehrke.

identification of indistinguishable objects, which also includes the identification of commutative factors. A commutative factor is a factor (i.e., a function) that outputs the same value regardless of the order of the input values of a subset of its arguments. In this paper, we solve the problem of efficiently detecting commutative factors by revisiting the theoretical foundations underlying the efficient detection of commutative factors and providing efficient algorithms with correctness guarantees that are missing in the current state of the art.

Previous work. Poole (2003) has introduced the notion of a parametric factor graph (PFG) as a lifted representation combining concepts from first-order logic with probabilistic modelling. To perform lifted inference, Poole (2003) has proposed lifted variable elimination (LVE) as a lifted inference algorithm that operates on a PFG, and since its introduction, LVE has continuously been refined by many researchers to reach its current form (Taghipour et al., 2013). The current state of the art to construct a lifted representation such as a PFG from a given propositional model is the advanced colour passing (ACP) algorithm (Luttermann et al., 2024a), which has also been generalised to enable approximate lifted model construction (Luttermann et al., 2025; Speller et al., 2025). An essential subroutine of lifted model construction (and hence of ACP) is to compute subsets of commutative arguments of a factor to be able to exploit the indistinguishability of objects. In its original form, ACP applies a brute-force approach to compute subsets of commutative arguments of a factor. Such a brute-force approach iterates over all possible subsets of arguments and checks whether a subset is actually commutative, thus requiring $O(2^n)$ sets to check for a factor with n arguments in the worst case. To avoid a computationally expensive brute-force approach, Luttermann et al. (2024b) have presented the detection of commutative factors (DECOR) algorithm, which applies a pruning strategy to heavily restrict the search space of subsets of arguments to consider. However, we found that a central theorem to the DECOR algorithm does not hold in its current form and hence, the correctness of DECOR is not guaranteed.

Our contributions. We first show that the central theorem underlying the DECOR algorithm provides only a necessary, not a sufficient, condition for the identification of commutative arguments, and we prove a corrected version of the theorem. Second, based on the corrected theorem, we present the DECOR+ algorithm, which is guaranteed to return a correct solution. We formally prove the correctness of DECOR+ and show that it maintains the efficiency of DECOR. Third, we introduce a complementary bottom-up algorithm, called Apriori-style detection of commutative factors (A-DECOR), which is inspired by the Apriori algorithm (Agrawal and Srikant, 1994) and exploits the downward closure and transitivity of pairwise commutativity, thereby yielding an algorithm with tighter worst-case bounds.

Structure of this paper. The remaining part of this paper is organised as follows. In Sec. 2, we introduce the necessary background on FGs and commutative factors. Then, in Sec. 3, we investigate the DECOR algorithm and provide a counterexample showing that DECOR does not guarantee correctness in its current form. We further present corrected theoretical results that clarify the necessary conditions for the identification of commutative factors. Thereafter, in Sec. 4, we show how our new theoretical results can be incorporated into the DECOR algorithm to ensure correctness while maintaining efficiency. In Sec. 5, we introduce the A-DECOR algorithm, which is a complementary bottom-up algorithm with tighter worst-case bounds before we empirically evaluate DECOR+ and A-DECOR Sec. 6.



(a)

$ComA$	$ComB$	Rev	$\phi_3(ComA, ComB, Rev)$
high	high	high	φ_1
high	high	low	φ_2
high	low	high	φ_3
high	low	low	φ_4
low	high	high	φ_3
low	high	low	φ_4
low	low	high	φ_5
low	low	low	φ_6

(b)

Figure 1: (*a*) An example for an FG modelling the interplay between the competences of two employees *Alice* and *Bob*, the revenue of the company they work for, and their salaries. (*b*) An exemplary function definition for ϕ_3, where $\varphi_1, \ldots, \varphi_6 \in \mathbb{R}_{\geq 0}$.

2. Background

An FG represents a joint probability distribution over a set of randvars by decomposing it into a product of local functions called factors (Frey et al., 1997; Kschischang et al., 2001). In the following, we write range(R) to denote the values a randvar R can take (called range).

Definition 1 (Factor Graph) *An* FG *is a tuple* $M = (\boldsymbol{R}, \boldsymbol{F}, \boldsymbol{E}, \boldsymbol{\Phi})$ *where* $\boldsymbol{R} = \{R_1, \ldots, R_n\}$ *is a set of randvars,* $\boldsymbol{F} = \{f_1, \ldots, f_m\}$ *is a set of factor names, and* $\boldsymbol{\Phi} = \{\phi_1, \ldots, \phi_m\}$ *is a set of function definitions (called factors). Each* $\phi_j(\mathcal{R}_j)$ *defines a function* $\phi_j \colon \times_{R \in \mathcal{R}_j} \text{range}(R) \to \mathbb{R}_{\geq 0}$ *defined over an argument sequence* $\mathcal{R}_j$ *of randvars from* $\boldsymbol{R}$ *that outputs positive real numbers, called potentials, of which at least one is non-zero. For each pair of variable node* $R_i \in \boldsymbol{R}$ *and factor node* $f_j \in \boldsymbol{F}$*, there is an edge* $\{R_i, f_j\} \in \boldsymbol{E} \subseteq \{\{R, f\} \mid R \in \boldsymbol{R} \wedge f \in \boldsymbol{F}\}$ *if* $R_i \in \mathcal{R}_j$*. The full joint distribution encoded by* M *for an assignment* $(R_1 = r_1, \ldots, R_n = r_n)$ *to the randvars in* $\boldsymbol{R}$*, abbreviated as* $\boldsymbol{R} = \boldsymbol{r}$*, is defined as*

$$P_M(\boldsymbol{R} = \boldsymbol{r}) = \frac{1}{Z} \prod_{j=1}^{m} \phi_j(\mathcal{R}_j = \boldsymbol{r}_j),$$

where $\boldsymbol{r}_j$ *is a projection of* $\boldsymbol{r}$ *to the argument list of* ϕ_j *and* Z *is the normalisation constant, defined as* $Z = \sum_{\boldsymbol{r} \in \text{range}(R_1) \times \ldots \times \text{range}(R_n)} \prod_{j=1}^{m} \phi_j(\mathcal{R}_j = \boldsymbol{r}_j)$.

Example 1 *Figure 1(a) displays an FG* $M = (\boldsymbol{R}, \boldsymbol{F}, \boldsymbol{E}, \boldsymbol{\Phi})$ *that models the interplay between the competences* $ComA$ *and* $ComB$ *of two employees Alice and Bob, the revenue* Rev *of the company they work for, and their corresponding salaries* $SalA$ *and* $SalB$*. The range of each randvar is* $\{\text{high}, \text{low}\}$*. An exemplary function definition for* ϕ_3 *is given in Fig. 1(b), where* $\varphi_1, \ldots, \varphi_6 \in \mathbb{R}_{\geq 0}$*. We omit the remaining function definitions for brevity.*

In this particular example, the revenue of the company equally depends on the competences of *Alice* and *Bob*. Consider the factor $\phi_3(ComA, ComB, Rev)$. Each assignment of range values to $ComA$, $ComB$, and Rev is mapped to a potential, which indicates the compatibility of the assigned values (i.e., a higher potential indicates a higher probability

for the assigned values to occur together). Specifically, we can verify that the revenue of the company equally depends on the competences of *Alice* and *Bob* because it holds that

$$\phi_3(\text{high}, \text{low}, \text{high}) = \phi_3(\text{low}, \text{high}, \text{high}) = \varphi_5, \text{ and}$$
$$\phi_3(\text{high}, \text{low}, \text{low}) \ = \phi_3(\text{low}, \text{high}, \text{low}) \ = \varphi_6.$$

In other words, the potential (and hence also the probability) for a high (low, respectively) revenue of the company is the same if one of the employees is highly competent and the other is not, regardless of which employee is the highly competent one. This property is referred to as the *commutativity* of a factor—here, ϕ_3 is commutative with respect to $ComA$ and $ComB$, i.e., $\phi_3(ComA, ComB, Rev) = \phi_3(ComB, ComA, Rev)$.

Definition 2 (Commutative Factor, Luttermann et al., 2024a) *Let $\phi(R_1, \dots, R_n)$ denote a factor and let $\boldsymbol{C} \subseteq \{R_1, \dots, R_n\}$ be a subset of ϕ's arguments with $|\boldsymbol{C}| > 1$. Then, ϕ is* commutative *with respect to $\boldsymbol{C}$ if for all assignments $(r_1, \dots, r_n) \in \times_{i=1}^{n} \text{range}(R_i)$ it holds that $\phi(r_1, \dots, r_n) = \phi(r_{\pi(1)}, \dots, r_{\pi(n)})$ for all permutations π of $\{1, \dots, n\}$ with $\pi(i) = i$ for all $R_i \notin \boldsymbol{C}$. The arguments in $\boldsymbol{C}$ are called commutative arguments.*

Since in our example, the revenue only depends on the number of highly competent employees and not on knowing which specific employee is highly competent, it is possible to compress ϕ_3 by grouping $ComA$ and $ComB$ together and counting how many high (and low, respectively) values there are, which is an essential part of the current state-of-the-art algorithm ACP to construct a lifted representation (Luttermann et al., 2024a). Specifically, commutative arguments might be grouped and represented by a so-called counting randvar (Milch et al., 2008), which can then efficiently be handled by lifted inference algorithms. A counting randvar counts the occurrences of specific range values in an assignment for a subset of a factor's arguments, and hence, it allows us to compactly encode a factor where only the number of randvars having a particular range value is relevant for the output but not which specific randvars are assigned that range value. We provide additional details about the concept of a counting randvar in App. A. To obtain the most compression possible, our goal is to find a subset $\boldsymbol{C}$ of commutative arguments of *maximum size*.

3. Efficient Detection of Commutative Factors with Buckets

To prune the search space of argument subsets when searching for a maximum sized subset of commutative arguments, the DECOR algorithm partitions the potentials a factor maps its arguments to into so-called *buckets*. A bucket counts the occurrences of specific range values in an assignment to a subset of arguments.

Definition 3 (Bucket, Luttermann et al., 2024b) *Let $\phi(R_1, \dots, R_n)$ denote a factor and let $\boldsymbol{C} \subseteq \{R_1, \dots, R_n\}$ denote a subset of ϕ's arguments such that all arguments in $\boldsymbol{C}$ have the same range $\boldsymbol{V}$. A* bucket *b entailed by $\boldsymbol{C}$ is a set of tuples $\{(v_i, n_i)\}_{i=1}^{|\boldsymbol{V}|}$ such that $n_i \in \mathbb{N}$ specifies the number of occurrences of range value $v_i \in \boldsymbol{V}$ in an assignment for all randvars in $\boldsymbol{C}$ (i.e., $\sum_{i=1}^{|\boldsymbol{V}|} n_i = |\boldsymbol{C}|$). A shorthand notation for $\{(v_i, n_i)\}_{i=1}^{|\boldsymbol{V}|}$ is $[n_1, \dots, n_{|\boldsymbol{V}|}]$. In abuse of notation, we denote by $\phi^{\succ}(b)$ the ordered multiset (with respect to their order in the potential table of ϕ) of potentials the assignments represented by b are mapped to by ϕ. The set of all buckets entailed by the arguments of ϕ is denoted as $\mathcal{B}(\phi)$.*

$ComA$	$ComB$	Rev	$\phi_3(ComA, ComB, Rev)$	b
high	high	high	φ_1	$[3,0]$
high	high	low	φ_2	$[2,1]$
high	low	high	φ_3	$[2,1]$
high	low	low	φ_4	$[1,2]$
low	high	high	φ_3	$[2,1]$
low	high	low	φ_4	$[1,2]$
low	low	high	φ_5	$[1,2]$
low	low	low	φ_6	$[0,3]$

b	$\phi_3^{\succ}(b)$
$[3,0]$	$\langle\varphi_1\rangle$
$[2,1]$	$\langle\varphi_2, \varphi_3, \varphi_3\rangle$
$[1,2]$	$\langle\varphi_4, \varphi_4, \varphi_5\rangle$
$[0,3]$	$\langle\varphi_6\rangle$

Table 1: A factor $\phi_3(ComA, ComB, Rev)$ and the partition of its potentials into buckets.

Example 2 *Consider the factor $\phi_3(ComA, ComB, Rev)$ depicted in Table 1 and let $\boldsymbol{C} = \{ComA, ComB, Rev\}$ with $\text{range}(ComA) = \text{range}(ComB) = \text{range}(Rev) = \{\text{high}, \text{low}\}$. Then, $\boldsymbol{C}$ entails the four buckets $\mathcal{B}(\phi_3) = \{(\text{high}, 3), (\text{low}, 0)\}, \{(\text{high}, 2), (\text{low}, 1)\}, \{(\text{high}, 1), (\text{low}, 2)\}, \{(\text{high}, 0), (\text{low}, 3)\}$ (or $\mathcal{B}(\phi_3) = \{[3,0], [2,1], [1,2], [0,3]\}$ in shorthand notation). According to the potential table of ϕ_3, it holds that $\phi_3^{\succ}([3,0]) = \langle\varphi_1\rangle$, $\phi_3^{\succ}([2,1]) = \langle\varphi_2, \varphi_3, \varphi_3\rangle$, $\phi_3^{\succ}([1,2]) = \langle\varphi_4, \varphi_4, \varphi_5\rangle$, and $\phi_3^{\succ}([0,3]) = \langle\varphi_6\rangle$, where the order of the potentials in each ordered multiset is given by their order of appearance in ϕ_3's potential table.*

Buckets induce helpful properties that allow for pruning the search space of possible subsets of commutative arguments, as we investigate next.

3.1. Theoretical Foundations of Commutativity Detection with Buckets

Luttermann et al. (2024b) show that the number of duplicate potentials in the buckets induces an upper bound for the size of a subset of commutative arguments of a factor.

Proposition 4 (Luttermann et al., 2024b) *Let $\phi(R_1, \ldots, R_n)$ be a factor that is commutative with respect to $\boldsymbol{C} \subseteq \{R_1, \ldots, R_n\}$. Then, for every bucket $b \in \mathcal{B}(\phi)$ with $|\phi^{\succ}(b)| > 1$, there exists a potential φ that occurs at least $|\boldsymbol{C}|$ times in $\phi^{\succ}(b)$.*

Hence, we can conclude that any commutative subset $\boldsymbol{C}$ contains at most as many arguments as the minimum number of duplicate potentials over all buckets containing more than one potential. In addition, buckets allow us to directly restrict candidates of possibly commutative arguments. Originally, the following claim has been established as the central theorem of DECOR to identify subsets of commutative arguments by looking at the element-wise intersection of the assignments corresponding to identical potentials in a bucket.

Claim 5 (Luttermann et al., 2024b) *Let $\phi(R_1, \ldots, R_n)$ denote a factor and let $b \in \mathcal{B}(\phi)$ with $|\phi^{\succ}(b)| > 1$ denote a bucket entailed by ϕ. Further, let $\boldsymbol{\Psi} = \{\varphi_1, \ldots, \varphi_\ell\}$ with $|\boldsymbol{\Psi}| > 2$ denote an arbitrary maximal set of identical potentials in $\phi^{\succ}(b)$ such that $\phi(r_1^1, \ldots, r_n^1) = \varphi_1, \ldots, \phi(r_1^\ell, \ldots, r_n^\ell) = \varphi_\ell$. Then, computing the element-wise intersection $(r_1^1, \ldots, r_n^1) \cap \ldots \cap (r_1^\ell, \ldots, r_n^\ell) = (\{r_1^1\} \cap \ldots \cap \{r_1^\ell\}, \ldots, \{r_n^1\} \cap \ldots \cap \{r_n^\ell\})$ and adding all arguments R_i for which $\{r_i^1\} \cap \ldots \cap \{r_i^\ell\} = \emptyset$ to a set $\boldsymbol{C}_b$ yields a set of candidate commutative arguments for bucket b such that $\boldsymbol{C} = \bigcap_b \boldsymbol{C}_b$ is a subset of commutative arguments of ϕ.*

We now show that this claim (reprinted from (Luttermann et al., 2024b)), which is central to DECOR, is flawed in its current form, as it only establishes a necessary condition for the identification of commutative arguments, but not a sufficient one as originally claimed.

Theorem 6 *The application of Claim 5 to compute a subset of arguments $\boldsymbol{C}$ does not guarantee that the computed set $\boldsymbol{C}$ is actually a subset of commutative arguments.*

Proof Consider $\phi(R_1, R_2, R_3, R_4)$ with $\text{range}(R_i) = \{\text{high}, \text{low}\}$ for all $i \in \{1, \ldots, 4\}$ and let $\phi(\text{low}, \text{low}, \text{high}, \text{high}) = \varphi_1$, $\phi(\text{high}, \text{high}, \text{low}, \text{low}) = \varphi_1$, and $\phi(\boldsymbol{r}) = \varphi_2$ with $\varphi_1 \neq \varphi_2$ for all remaining assignments $\boldsymbol{r}$. Applying Claim 5 yields $\boldsymbol{C} = \{R_1, R_2, R_3, R_4\}$ although ϕ is not commutative with respect to $\boldsymbol{C}$ (a detailed computation of $\boldsymbol{C}$ is given in App. B). ■

Theorem 6 shows that applying Claim 5 is not guaranteed to yield a valid subset of commutative arguments. To solve this problem, we next rephrase Claim 5 as a necessary condition—because the subset obtained from applying Claim 5 does not miss any truly commutative arguments, but it may happen that too many arguments are included.

3.2. Necessary Condition for the Detection of Commutative Arguments

The upcoming theorem presents a corrected version of Claim 5 that captures the necessary condition for the identification of commutative arguments by looking at the element-wise intersection of the assignments corresponding to groups of identical potentials in a bucket.

Theorem 7 *Let $\phi(R_1, \ldots, R_n)$ denote a factor, let $\boldsymbol{C} \subseteq \{R_1, \ldots, R_n\}$ with $|\boldsymbol{C}| > 1$ denote a subset of ϕ's arguments of maximum size such that ϕ is commutative with respect to $\boldsymbol{C}$, and let $b \in \mathcal{B}(\phi)$ denote any bucket entailed by the arguments of ϕ such that $|\phi^{\succ}(b)| > 1$ holds. Then, there exists a maximal set of identical potentials $\boldsymbol{\Psi} = \{\varphi_1, \ldots, \varphi_\ell\} \subseteq \phi^{\succ}(b)$ in $\phi^{\succ}(b)$ with $|\boldsymbol{\Psi}| > 1$ and $\phi(r_1^1, \ldots, r_n^1) = \varphi_1, \ldots, \phi(r_1^\ell, \ldots, r_n^\ell) = \varphi_\ell$ such that for all $R_i \in \boldsymbol{C}$ the element-wise intersection $\left(r_1^1, \ldots, r_n^1\right) \cap \ldots \cap \left(r_1^\ell, \ldots, r_n^\ell\right) = \left(\{r_1^1\} \cap \ldots \cap \{r_1^\ell\}, \ldots, \{r_n^1\} \cap \ldots \cap \{r_n^\ell\}\right)$ contains an empty set at position $i \in \{1, \ldots, n\}$.*

Proof Since we consider only buckets b with at least two elements in $\phi^{\succ}(b)$, all corresponding assignments contain at least two distinct assigned values because all assignments that contain only a single value are those that belong to a bucket that is mapped to a single potential. According to Def. 2, for all assignments $(r_1, \ldots, r_n) \in \times_{i=1}^{n} \text{range}(R_i)$ it holds that $\phi(r_1, \ldots, r_n) = \phi(r_{\pi(1)}, \ldots, r_{\pi(n)})$ for all permutations π of $\{1, \ldots, n\}$ with $\pi(i) = i$ for all $R_i \notin \boldsymbol{C}$. Let $(r_1^1, \ldots, r_n^1)$ denote any assignment that belongs to bucket b, that is, $\phi(r_1^1, \ldots, r_n^1) = \varphi$ with $\varphi \in \phi^{\succ}(b)$, such that $(r_1^1, \ldots, r_n^1)$ contains at least two distinct assigned values, say r_j^1 and r_k^1 with $r_j^1 \neq r_k^1$, at any two positions $j, k \in \{1, \ldots, n\}$ for which it holds that $R_j \in \boldsymbol{C}$ and $R_k \in \boldsymbol{C}$ are commutative arguments of ϕ. Further, let $\boldsymbol{\Psi} \subseteq \phi^{\succ}(b)$ denote the set of all potentials $\varphi = \phi(r_{\pi(1)}, \ldots, r_{\pi(n)})$ resulting from applying all permutations π of $\{1, \ldots, n\}$ with $\pi(i) = i$ for all $R_i \notin \boldsymbol{C}$ to the assignment $(r_1^1, \ldots, r_n^1)$. By construction, all potentials in $\boldsymbol{\Psi}$ are equal to φ and hence $\boldsymbol{\Psi}$ is a set of identical potentials. As it holds that $|\boldsymbol{C}| > 1$, there are at least two positions that are not fixed in each permutation π and that contain distinct assigned values in the assignment $(r_1^1, \ldots, r_n^1)$, which implies that there exist at least two distinct assignments, say $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ with $\boldsymbol{r}_1 \neq \boldsymbol{r}_2$, such that $\phi(\boldsymbol{r}_1) = \varphi \in \boldsymbol{\Psi}$ and

$\phi(\boldsymbol{r}_2) = \varphi \in \boldsymbol{\Psi}$ and thus, it holds that $|\boldsymbol{\Psi}| > 1$. Now, let $(r_1^1, \ldots, r_n^1), \ldots, (r_1^\ell, \ldots, r_n^\ell)$ denote the assignments corresponding to the potentials in $\boldsymbol{\Psi}$, which are obtained by applying all permutations π of $\{1, \ldots, n\}$ with $\pi(i) = i$ for all $R_i \notin \boldsymbol{C}$ to the assignment $(r_1^1, \ldots, r_n^1)$. As each position $i \in \{1, \ldots, n\}$ with $R_i \in \boldsymbol{C}$ is not fixed in the permutations and there exist at least two distinct assigned values that are permuted over all positions belonging to arguments in $\boldsymbol{C}$, there are at least two assignments among $(r_1^1, \ldots, r_n^1), \ldots, (r_1^\ell, \ldots, r_n^\ell)$ such that their assigned values at each such position i differ. Thus, computing the intersection of the assigned values at any such position i yields an empty set. In case there are further potentials in $\phi^{\succ}(b)$ that are equal to φ but not contained in $\boldsymbol{\Psi}$ so far (that is, if $\boldsymbol{\Psi}$ is not maximal), we extend $\boldsymbol{\Psi}$ by adding these potentials to $\boldsymbol{\Psi}$ such that $\boldsymbol{\Psi}$ becomes a maximal set of identical potentials. The element-wise intersection of all assignments corresponding to the potentials in $\boldsymbol{\Psi}$ then still yields an empty set at each position $i \in \{1, \ldots, n\}$ for which it holds that $R_i \in \boldsymbol{C}$ because the intersection has already been empty at these positions before extending $\boldsymbol{\Psi}$ and the extension of $\boldsymbol{\Psi}$ only adds additional sets to be intersected. ■

We next apply Thm. 7 to the factor $\phi_3(ComA, ComB, Rev)$ depicted in Table 1.

Example 3 *Consider $\phi_3(ComA, ComB, Rev)$ from Table 1 and let $b = [2, 1]$. There are two maximal groups of identical potentials in $\phi_3^{\succ}(b)$: $\boldsymbol{\Psi}_1 = \{\varphi_2\}$ and $\boldsymbol{\Psi}_2 = \{\varphi_3, \varphi_3\}$. Since $|\boldsymbol{\Psi}_1| = 1$, no candidates of commutative arguments are induced by $\boldsymbol{\Psi}_1$. However, $\boldsymbol{\Psi}_2$ contains the potential φ_3 twice and the corresponding assignments are* (high, low, high) *and* (low, high, high). *The element-wise intersection is then given by* ({high}, {low}, {high}) ∩ ({low}, {high}, {high}) = (∅, ∅, {high}). *As the element-wise intersection is empty at positions one and two, the set $\{ComA, ComB\}$ is a candidate subset of commutative arguments.*

By applying Thm. 7, we are able to significantly prune the search space of possible subsets of commutative arguments. Nevertheless, as Thm. 7 provides only a necessary condition, we need an additional verification step in the end to check whether the computed candidate subset is indeed a subset of commutative arguments. We next show how Thm. 7 is applied in practice to efficiently identify maximum sized subsets of commutative arguments.

4. The DECOR+ Algorithm

To enable the efficient computation of maximum sized subsets of commutative arguments in practice, we introduce a revised version of the DECOR algorithm (called DECOR+) based on the theoretical results presented in the previous section. Algorithm 1 presents the entire DECOR+ algorithm, which we now describe in detail. DECOR+ starts with the candidate subset of all arguments and then iterates over all buckets containing at least two potentials to compute partitions of maximal groups of identical potentials in each bucket, which are then used to prune the search space according to Thm. 7. Specifically, if there are no duplicate potentials in a bucket at all, there cannot be any commutative arguments (Prop. 4) and DECOR+ immediately returns (Line 5). Otherwise, DECOR+ computes a subset $\boldsymbol{C}_i$ of candidate arguments permitted by each maximal group $\boldsymbol{\Psi}_i$ of identical potentials in the currently considered bucket. The subset $\boldsymbol{C}_i$ contains all arguments R_i for which the element-wise intersection of the assignments corresponding to the potentials in $\boldsymbol{\Psi}_i$ is empty at position i. The computed subsets $\boldsymbol{C}_i$ of candidate arguments for each group of

Algorithm 1 Detection of Commutative Factors Revised (DECOR+)

Input: A factor $\phi(R_1, \ldots, R_n)$.
Output: A set containing all maximum sized subsets of commutative arguments of ϕ.

$\boldsymbol{C}_{cand} \leftarrow \{\{R_1, \ldots, R_n\}\}$
for each bucket $b \in \mathcal{B}(\phi)$ **do**
  if $|\phi^{\succ}(b)| < 2$ **then continue**
  $\boldsymbol{\Psi}_1, \ldots, \boldsymbol{\Psi}_\ell \leftarrow$ Partition of $\phi^{\succ}(b)$ into maximal groups of identical potentials such that $|\boldsymbol{\Psi}_i| \geq 2$ holds for all $i \in \{1, \ldots, \ell\}$
  if $\ell < 1$ **then return** $\emptyset$
  $\boldsymbol{C}' \leftarrow \emptyset$
  for each group $\boldsymbol{\Psi}_i \in \{\boldsymbol{\Psi}_1, \ldots, \boldsymbol{\Psi}_\ell\}$ **do**
    $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n \leftarrow$ Intersection of positions corresponding to all potentials in $\boldsymbol{\Psi}_i$
    $\boldsymbol{C}_i \leftarrow \{R_i \mid i \in \{1, \ldots, n\} \wedge \boldsymbol{P}_i = \emptyset\}$
    if $\nexists \boldsymbol{C}_j \in \boldsymbol{C}' : \boldsymbol{C}_i \subseteq \boldsymbol{C}_j$ **then** $\boldsymbol{C}' \leftarrow \boldsymbol{C}' \cup \{\boldsymbol{C}_i\}$
  $\boldsymbol{C}_\cap \leftarrow \emptyset$
  for each candidate set $\boldsymbol{C}_i \in \boldsymbol{C}_{cand}$ **do**
    for each candidate set $\boldsymbol{C}_j \in \boldsymbol{C}'$ **do**
      if $|\boldsymbol{C}_i \cap \boldsymbol{C}_j| \geq 2 \wedge \nexists \boldsymbol{C}'' \in \boldsymbol{C}_\cap : (\boldsymbol{C}_i \cap \boldsymbol{C}_j) \subseteq \boldsymbol{C}''$ **then** $\boldsymbol{C}_\cap \leftarrow \boldsymbol{C}_\cap \cup \{\boldsymbol{C}_i \cap \boldsymbol{C}_j\}$
  $\boldsymbol{C}_{cand} \leftarrow \boldsymbol{C}_\cap$
  if $\boldsymbol{C}_{cand} = \emptyset$ **then return** $\emptyset$
return $verify(\boldsymbol{C}_{cand})$

identical potentials in the current bucket b are then collected into a set $\boldsymbol{C}'$ of candidate subsets for the current bucket b (such that no candidate subset is subsumed by another candidate subset). The collected candidate subsets permitted by a specific bucket are then intersected with the candidate subsets in $\boldsymbol{C}_{cand}$ collected from the previous buckets to obtain candidate subsets that are permitted by all buckets considered so far (again, subsets that are subsumed by another candidate subset after the intersection are removed). If there are no candidate subsets left at some point, DECOR+ immediately returns. Finally, if there are candidate subsets left after iterating over all buckets, DECOR+ runs a verification step to check whether the candidate subsets are indeed subsets of commutative arguments. The verification step considers every candidate subset $\boldsymbol{C} \in \boldsymbol{C}_{cand}$ and checks whether the factor ϕ is indeed commutative with respect to $\boldsymbol{C}$ (by directly applying Def. 2). If the check succeeds, $\boldsymbol{C}$ is kept as a confirmed subset of commutative arguments. Otherwise, $\boldsymbol{C}$ might still contain a strict subset of commutative arguments and DECOR+ continues by checking all $(|\boldsymbol{C}| - 1)$-element subsets of $\boldsymbol{C}$, then all $(|\boldsymbol{C}| - 2)$-element subsets of $\boldsymbol{C}$, and so on, until either a commutative subset is found or no subsets of size at least two remain. Since we are only interested in a single maximum sized subset of commutative arguments, DECOR+ might stop the verification as soon as the first commutative subset is found (we nevertheless formulate DECOR+ as a general algorithm in Alg. 1 that is able to return all maximum sized subsets of commutative arguments if desired). An example run of DECOR+ is given in App. C. We also emphasise that DECOR+ can handle factors with arguments having arbitrary ranges. In this paper, we consider identical ranges of all arguments for brevity,

however, it is possible to apply DECOR+ to factors with arguments having arbitrary ranges $\boldsymbol{V}_1, \ldots, \boldsymbol{V}_k$ by considering the buckets for all arguments with range $\boldsymbol{V}_i$ separately for all $i \in \{1, \ldots, k\}$ as only arguments with the same range can be in the same set of commutative arguments. Next, we prove the correctness of DECOR+, i.e., we show that the pruning step in DECOR+ does not remove any truly commutative arguments such that the verification step is guaranteed to find a maximum sized subset of commutative arguments.

Theorem 8 *Let $\phi(R_1, \ldots, R_n)$ denote a factor that is given as an input to DECOR+ (Alg. 1). Then, the set returned by DECOR+ contains exactly the maximum sized subsets of commutative arguments of ϕ's arguments, or the empty set if no such subset exists.*

Proof Let $\boldsymbol{C}^* \subseteq \{R_1, \ldots, R_n\}$ with $|\boldsymbol{C}^*| \geq 2$ denote any maximum sized subset of commutative arguments of ϕ. By Thm. 7, for every bucket $b \in \mathcal{B}(\phi)$ with $|\phi^{\succ}(b)| > 1$, there exists a maximal set of identical potentials $\boldsymbol{\Psi}_i \subseteq \phi^{\succ}(b)$ with $|\boldsymbol{\Psi}_i| > 1$ such that the element-wise intersection of the assignments corresponding to the potentials in $\boldsymbol{\Psi}_i$ contains an empty set at every position $j \in \{1, \ldots, n\}$ for which $R_j \in \boldsymbol{C}^*$. Consequently, the set $\boldsymbol{C}_i$ computed for $\boldsymbol{\Psi}_i$ in the inner loop of Alg. 1 (Line 9) satisfies $\boldsymbol{C}^* \subseteq \boldsymbol{C}_i$, and hence either $\boldsymbol{C}_i$ itself is added to $\boldsymbol{C}'$ or it is subsumed by another candidate subset already in $\boldsymbol{C}'$ that contains $\boldsymbol{C}^*$. The cross-intersection step preserves $\boldsymbol{C}^*$ as a subset of some entry in $\boldsymbol{C}_{cand}$ at every iteration, because the intersection of two supersets of $\boldsymbol{C}^*$ is itself a superset of $\boldsymbol{C}^*$ and the subsumption check only removes entries that are themselves subsumed by a superset. By induction over the buckets, every maximum sized subset $\boldsymbol{C}^*$ of commutative arguments of ϕ is contained in some entry of $\boldsymbol{C}_{cand}$ after the bucket loop terminates. The correctness then immediately follows as the verification step is exact (it directly applies Def. 2) and considers the subsets in order of decreasing size (if verification fails, an empty set is returned). ■

Even though DECOR+ might have to check all subsets of a candidate subset in the worst case during the verification step, a major advantage of DECOR+ is that the number of candidate subsets computed by DECOR+ depends on the number of maximal groups of identical potentials in the buckets. In most practical settings, potentials are not identical unless they stem from a subset of commutative arguments. Therefore, DECOR+ heavily restricts the search space in most practical settings, which we also confirm later in our experiments. Specifically, the bucket structure of a factor ϕ provides an even tighter bound than the bound implied by Prop. 4 on the size of any commutative subset of ϕ.

Proposition 9 *Let $\phi(R_1, \ldots, R_n)$ denote a factor that is commutative with respect to $\boldsymbol{C} \subseteq \{R_1, \ldots, R_n\}$ and let $\boldsymbol{C}'_b$ denote the set of candidate subsets computed by DECOR+ for a bucket $b \in \mathcal{B}(\phi)$ (Lines 6 to 10 of Alg. 1). Then, $|\boldsymbol{C}| \leq \min_{b \in \mathcal{B}(\phi), |\phi^{\succ}(b)|>1} \max_{\boldsymbol{C}_i \in \boldsymbol{C}'_b} |\boldsymbol{C}_i|$.*

Proof Let $b \in \mathcal{B}(\phi)$ with $|\phi^{\succ}(b)| > 1$. By Thm. 7, there exists a maximal set of identical potentials $\boldsymbol{\Psi} \subseteq \phi^{\succ}(b)$ with $|\boldsymbol{\Psi}| > 1$ such that the element-wise intersection over the assignments corresponding to the potentials in $\boldsymbol{\Psi}$ contains an empty set at every position $i \in \{1, \ldots, n\}$ with $R_i \in \boldsymbol{C}$. Hence, the candidate subset $\boldsymbol{C}_{\boldsymbol{\Psi}}$ computed by DECOR+ for $\boldsymbol{\Psi}$ in Line 9 satisfies $\boldsymbol{C} \subseteq \boldsymbol{C}_{\boldsymbol{\Psi}}$. Due to the subsumption check in Line 10, either $\boldsymbol{C}_{\boldsymbol{\Psi}} \in \boldsymbol{C}'_b$ or $\boldsymbol{C}_{\boldsymbol{\Psi}}$ is subsumed by some $\boldsymbol{C}_j \in \boldsymbol{C}'_b$ with $\boldsymbol{C}_{\boldsymbol{\Psi}} \subseteq \boldsymbol{C}_j$. In both cases, there exists $\boldsymbol{C}_i \in \boldsymbol{C}'_b$ with

$\boldsymbol{C} \subseteq \boldsymbol{C}_i$, and thus $|\boldsymbol{C}| \leq |\boldsymbol{C}_i| \leq \max_{\boldsymbol{C}_i \in \boldsymbol{C}'_b} |\boldsymbol{C}_i|$. As this bound holds for each $b \in \mathcal{B}(\phi)$ with $|\phi^{\succ}(b)| > 1$, the claim follows by taking the minimum over all such buckets. ■

Motivated by the example given in the proof of Thm. 6 showing that the candidate subsets passed to the verification step of DECOR+ may strictly over-approximate the actual subsets of commutative arguments, we next present a complementary approach that avoids the verification step entirely and hence requires less commutativity checks in the worst case.

5. An Apriori-Style Algorithm for the Detection of Commutative Factors

The DECOR+ algorithm approaches the search for a maximum sized subset of commutative arguments *top-down*: It starts with a candidate set of all arguments and aggregates evidence from the buckets to gradually narrow down the candidate subsets. We now explore a *bottom-up* approach inspired by the well-known *Apriori* algorithm for association rule mining (Agrawal and Srikant, 1994). The idea is to enumerate commutative subsets level-by-level, starting from pairs of arguments and extending them step by step. The underlying key observation for this approach is that commutativity is a *downward-closed* property.

Proposition 10 *Let $\phi(R_1, \ldots, R_n)$ denote a factor and let $\boldsymbol{C} \subseteq \{R_1, \ldots, R_n\}$ with $|\boldsymbol{C}| \geq 2$ denote a subset of arguments such that ϕ is commutative with respect to $\boldsymbol{C}$. Then, for every subset $\boldsymbol{C}' \subseteq \boldsymbol{C}$ with $|\boldsymbol{C}'| \geq 2$, ϕ is also commutative with respect to $\boldsymbol{C}'$.*

Proof Let $\boldsymbol{C}' \subseteq \boldsymbol{C}$ with $|\boldsymbol{C}'| \geq 2$ and let π' denote any permutation of $\{1, \ldots, n\}$ with $\pi'(i) = i$ for all $R_i \notin \boldsymbol{C}'$. Since $\boldsymbol{C}' \subseteq \boldsymbol{C}$, every $R_i \notin \boldsymbol{C}$ also satisfies $R_i \notin \boldsymbol{C}'$, and hence the condition $\pi'(i) = i$ for all $R_i \notin \boldsymbol{C}'$ implies $\pi'(i) = i$ for all $R_i \notin \boldsymbol{C}$. Consequently, π' is also a valid permutation for the commutativity of ϕ with respect to $\boldsymbol{C}$ such that $\phi(r_1, \ldots, r_n) = \phi(r_{\pi'(1)}, \ldots, r_{\pi'(n)})$ holds for every assignment $(r_1, \ldots, r_n)$. ■

The contrapositive of Prop. 10 states that the set of non-commutative subsets is upward-closed: Once a subset $\boldsymbol{C}$ has been found to be non-commutative, every superset of $\boldsymbol{C}$ is also non-commutative. A second structural property that further accelerates the bottom-up search is that the commutativity of a subset of arguments is fully determined by the commutativity of its constituent pairs, that is, pairwise commutativity is transitive.

Theorem 11 *Let $\phi(R_1, \ldots, R_n)$ denote a factor and let $\boldsymbol{C} \subseteq \{R_1, \ldots, R_n\}$ with $|\boldsymbol{C}| \geq 2$ denote a subset of arguments. Then, ϕ is commutative with respect to $\boldsymbol{C}$ if and only if ϕ is commutative with respect to every pair $\{R_i, R_j\} \subseteq \boldsymbol{C}$ with $i \neq j$.*

Proof The forward direction follows directly from Prop. 10. For the converse direction, assume that ϕ is commutative with respect to every pair $\{R_i, R_j\} \subseteq \boldsymbol{C}$ with $i \neq j$, and let π denote any permutation of $\{1, \ldots, n\}$ with $\pi(i) = i$ for all $R_i \notin \boldsymbol{C}$. Then, π permutes only indices of arguments in $\boldsymbol{C}$, and hence can be written as a composition $\pi = \tau_1 \circ \tau_2 \circ \cdots \circ \tau_m$ of transpositions, where each transposition τ_ℓ swaps two indices i_ℓ and j_ℓ with $R_{i_\ell}, R_{j_\ell} \in \boldsymbol{C}$. By assumption, ϕ is commutative with respect to $\{R_{i_\ell}, R_{j_\ell}\}$ for every $\ell \in \{1, \ldots, m\}$, that is, ϕ is invariant under each transposition τ_ℓ. Consequently, ϕ is invariant under the composition $\pi = \tau_1 \circ \tau_2 \circ \cdots \circ \tau_m$, that is, $\phi(r_1, \ldots, r_n) = \phi(r_{\pi(1)}, \ldots, r_{\pi(n)})$ holds for every assignment $(r_1, \ldots, r_n)$ and hence, by Def. 2, ϕ is commutative with respect to $\boldsymbol{C}$. ■

Algorithm 2 Apriori-Style Detection of Commutative Factors (A-DECOR)

Input: A factor $\phi(R_1, \ldots, R_n)$.
Output: A set containing all maximum sized subsets of commutative arguments of ϕ.

1: $\boldsymbol{L}_2 \leftarrow \{\{R_i, R_j\} \mid 1 \leq i < j \leq n \wedge \phi \text{ is commutative with respect to } \{R_i, R_j\}\}$
2: $\boldsymbol{L} \leftarrow \boldsymbol{L}_2, \quad k \leftarrow 2$
3: **while** $\boldsymbol{L}_k \neq \emptyset$ **do**
4: $\quad \boldsymbol{L}_{k+1} \leftarrow \emptyset$
5: $\quad$ **for each** $\boldsymbol{C} \in \boldsymbol{L}_k$ **do**
6: $\quad\quad$ **for each** $R_i \notin \boldsymbol{C}$ **do**
7: $\quad\quad\quad$ **if** $\forall R_j \in \boldsymbol{C}\colon \{R_i, R_j\} \in \boldsymbol{L}_2$ **then** $\boldsymbol{L}_{k+1} \leftarrow \boldsymbol{L}_{k+1} \cup \{\boldsymbol{C} \cup \{R_i\}\}$
8: $\quad$ **if** $\boldsymbol{L}_{k+1} \neq \emptyset$ **then** $\boldsymbol{L} \leftarrow \boldsymbol{L}_{k+1}$
9: $\quad k \leftarrow k + 1$
10: **return** $\boldsymbol{L}$

Together, Prop. 10 and Thm. 11 enable an efficient pruning strategy, which builds the basis of the A-DECOR algorithm, depicted in Alg. 2. A-DECOR starts by computing the set $\boldsymbol{L}_2$ of all pairs of commutative arguments in Line 1 by checking commutativity for every pair $\{R_i, R_j\}$ of arguments individually. Subsequently, A-DECOR enters a loop in which it extends each subset $\boldsymbol{C} \in \boldsymbol{L}_k$ by every argument $R_i \notin \boldsymbol{C}$ to form a candidate $\boldsymbol{C}' = \boldsymbol{C} \cup \{R_i\}$ of size $k+1$. By Thm. 11, $\boldsymbol{C}'$ is commutative if and only if every pair of arguments in $\boldsymbol{C}'$ is commutative. The pairs internal to $\boldsymbol{C}$ are commutative since $\boldsymbol{C} \in \boldsymbol{L}_k$ and $\boldsymbol{L}_k$ contains only commutative subsets, so it suffices to verify the $|\boldsymbol{C}|$ pairs $\{R_i, R_j\}$ with $R_j \in \boldsymbol{C}$ are commutative (i.e., are in $\boldsymbol{L}_2$). This is exactly the test performed in Line 7, which obviates the need for any additional commutativity check on $\boldsymbol{C}'$ itself. Throughout the bottom-up search, A-DECOR keeps track of the most recent non-empty layer $\boldsymbol{L}_k$ in $\boldsymbol{L}$ (Line 8). A-DECOR terminates as soon as no commutative subset of the current size k is found, and returns the largest commutative subsets found thus far (stored in $\boldsymbol{L}$).

The correctness of A-DECOR follows directly from Prop. 10 and Thm. 11 and we give a formal correctness proof in App. D, where we also show how A-DECOR can be implemented efficiently. It is noteworthy that A-DECOR is guaranteed to only perform $O(n^2)$ commutativity checks in the worst case (compared to $O(2^n)$ checks for DECOR and DECOR+).

6. Experiments

In addition to our theoretical results, we empirically evaluate DECOR+ and A-DECOR to assess their performance in practice. We compare the run times of the original DECOR algorithm (which is not guaranteed to return a correct solution), DECOR+, A-DECOR, and the brute-force algorithm (which checks every possible subset of arguments in order of decreasing size for commutativity). To keep the comparison to DECOR fair, we use the same input instances as in the original DECOR paper (Luttermann et al., 2024b) and also add new instances with varying distributions of subsets of commutative arguments. The input factors used in this evaluation contain $n \in \{2, 4, 6, 8, 10, 12, 14, 16\}$ Boolean arguments and we vary the number k of commutative arguments between 0 and n. We run each algorithm

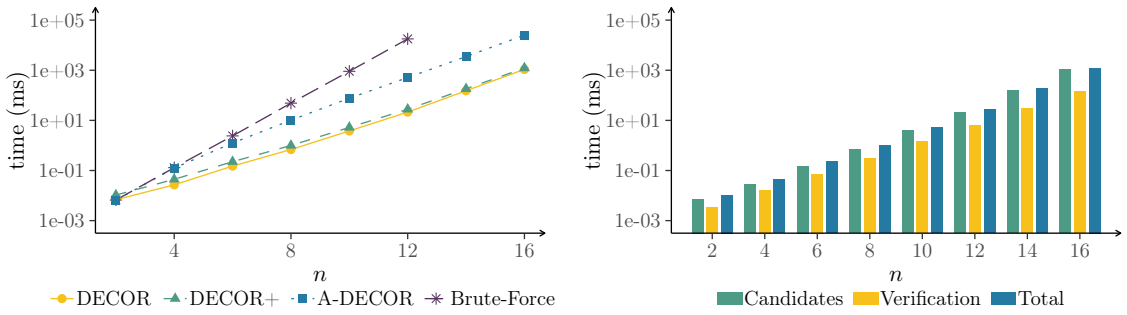


Figure 2: (*a*) Average run times of DECOR, DECOR+, A-DECOR, and the brute-force algorithm for input factors with n arguments. (*b*) Proportion of the run time of DECOR+ spent in the candidate generation loop and in the verification step.

with a timeout of five minutes per instance and report the average run time over all instances for each choice of n. In App. E, we also report results for individual scenarios separately.

Figure 2(*a*) displays for each choice of n the average run times of the algorithms over all choices of numbers of commutative arguments k, where k is varied between 0 and n. While the brute-force algorithm only scales to small instances and runs into the timeout as n grows, DECOR+ solves all instances comfortably within the timeout and matches the run time of the (unsound) original DECOR algorithm—the correctness guarantee provided by DECOR+ thus comes essentially for free. It also becomes evident that A-DECOR is not able to match the performance of DECOR+, even though it comes with a tighter worst case bound, which can be explained by the fact that A-DECOR always performs $\Omega(n^2)$ commutativity checks, each iterating over all entries in the factor's potential table (which has exponential size in the number of arguments) while DECOR+ requires mostly just a constant number of iterations over the potential table in practice. Figure 2(*b*) decomposes the total run time of DECOR+ into the candidate generation loop (Line 2) and the verification step (Line 17) at the end. We note that in our experiments, there is at most one candidate subset to be verified by DECOR+ for each instance, which highlights the effectiveness of the pruning strategy in the candidate generation loop of DECOR+ and also explains the low run times for the verification step (note that the y-axis is log-scaled and hence, the "Candidates" and "Verification" bars do not visually add up to the "Total" bar, whereas their numbers do).

7. Conclusion

We have revisited the theoretical foundations of detecting commutative factors in an FG and identified a flaw in the central theorem of the original DECOR algorithm. Based on a corrected version of the theorem, we have presented DECOR+, a revised algorithm that augments the search space pruning with an explicit verification step and is guaranteed to return a correct solution. We also introduced a complementary bottom-up algorithm A-DECOR with tighter worst-case bounds than DECOR+. Our experimental evaluation confirms the practical efficiency of DECOR+ and the effectiveness of its pruning strategy.

Declaration on Generative AI

During the preparation of this work, the authors used Claude Code (Anthropic) to discuss ideas and to polish text passages. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the Twentieth International Conference on Very Large Data Bases (VLDB-1994)*, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

Gregory F. Cooper. The Computational Complexity of Probabilistic Inference using Bayesian Belief Networks. *Artificial Intelligence*, 42:393–405, 1990.

Paul Dagum and Michael Luby. Approximating Probabilistic Inference in Bayesian Belief Networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

Brendan J. Frey, Frank R. Kschischang, Hans-Andrea Loeliger, and Niclas Wiberg. Factor Graphs and Algorithms. In *Proceedings of the Thirty-Fifth Annual Allerton Conference on Communication, Control, and Computing*, pages 666–680. Allerton House, 1997.

Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

Malte Luttermann, Tanya Braun, Ralf Möller, and Marcel Gehrke. Colour Passing Revisited: Lifted Model Construction with Commutative Factors. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-2024)*, pages 20500–20507. AAAI Press, 2024a.

Malte Luttermann, Johann Machemer, and Marcel Gehrke. Efficient Detection of Commutative Factors in Factor Graphs. In *Proceedings of the Twelfth International Conference on Probabilistic Graphical Models (PGM-2024)*, pages 38–56. PMLR, 2024b.

Malte Luttermann, Jan Speller, Marcel Gehrke, Tanya Braun, Ralf Möller, and Mattis Hartwig. Approximate Lifted Model Construction. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-2025)*, pages 9077–9085. IJCAI Organization, 2025.

Brian Milch, Luke S. Zettlemoyer, Kristian Kersting, Michael Haimes, and Leslie Pack Kaelbling. Lifted Probabilistic Inference with Counting Formulas. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, pages 1062–1068. AAAI Press, 2008.

Mathias Niepert and Guy Van den Broeck. Tractability through Exchangeability: A New Perspective on Efficient Probabilistic Inference. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-2014)*, pages 2467–2475. AAAI Press, 2014.

David Poole. First-Order Probabilistic Inference. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-2003)*, pages 985–991. Morgan Kaufmann Publishers Inc., 2003.

Jan Speller, Malte Luttermann, Marcel Gehrke, and Tanya Braun. Compression versus Accuracy: A Hierarchy of Lifted Models. In *Proceedings of the Twenty-Eighth European Conference on Artificial Intelligence (ECAI-2025)*, pages 5051–5058. IOS Press, 2025.

Nima Taghipour. *Lifted Probabilistic Inference by Variable Elimination*. PhD thesis, KU Leuven, 2013.

Nima Taghipour, Daan Fierens, Jesse Davis, and Hendrik Blockeel. Lifted Variable Elimination: Decoupling the Operators from the Constraint Language. *Journal of Artificial Intelligence Research*, 47:393–439, 2013.

Robert Endre Tarjan. A Class of Algorithms Which Require Nonlinear Time to Maintain Disjoint Sets. *Journal of Computer and System Sciences*, 18:110–127, 1979.

Appendix A. Counting Random Variables

In this section, we briefly discuss the concept of a counting randvar (CRV) (Milch et al., 2008), which allows us to compactly encode a factor where it does not matter which specific individual randvars have a certain range value but instead only the number of randvars having that range value is of interest. The range of a CRV is the space of histograms, that is, each range value is a histogram indicating how many of the randvars represented the CRV have a certain range value. Since a formal definition of a CRV requires concepts not directly related to the main topic of this paper, we refrain from giving a formal definition and instead illustrate the concept of a CRV by an example.

Example 4 *Consider the factor* $\phi_3(ComA, ComB, Rev)$ *from Fig. 1(*b*). Using* h *and* l *as shorthand notations for* high *and* low*, respectively,* ϕ_3 *encodes the following potential table:*

$$\begin{aligned} \phi_3(h,h,h) &= \varphi_1, \quad \phi_3(h,h,l) = \varphi_2, \quad \phi_3(h,l,h) = \varphi_3, \quad \phi_3(h,l,l) = \varphi_4, \\ \phi_3(l,h,h) &= \varphi_3, \quad \phi_3(l,h,l) = \varphi_4, \quad \phi_3(l,l,h) = \varphi_5, \quad \phi_3(l,l,l) = \varphi_6. \end{aligned}$$

It becomes evident that for a specific value of Rev*, it does not matter which specific employees have a certain competence but only how many employees have a certain competence: For* $Rev =$ high*, two* high *values for the competences are mapped to* φ_1*, one* high *and one* low *value are mapped to* φ_3*, and two* low *values are mapped to* φ_5*. Analogously, for* $Rev =$ low*, two* high *values for the competences are mapped to* φ_2*, one* high *and one* low *value are mapped to* φ_4*, and two* low *values are mapped to* φ_6*. Now, consider a CRV that counts over the competences of Alice and Bob, denoted as* $\#_E[Com(E)]$ *with* $\text{range}(Com(E)) = \{\text{low}, \text{high}\}$ *and* $\text{dom}(E) = \{Alice, Bob\}$ *being the domain of the logical variable* E *representing the employees. Then, there are* $m = |\text{range}(Com(E))| = 2$ *possible range values and* $n = 2$ *represented instances (*$Com(Alice)$*, or* $ComA$ *for short, and* $Com(Bob)$*, or* $ComB$ *for short). Hence, the histograms are* $[0, 2]$*,* $[1, 1]$*, and* $[2, 0]$ *(corresponding to* $\{(\text{high}, 0), (\text{low}, 2)\}$*,* $\{(\text{high}, 1), (\text{low}, 1)\}$*, and* $\{(\text{high}, 2), (\text{low}, 0)\}$ *in set notation, respectively). By replacing* $ComA$ *and* $ComB$ *with* $\#_E[Com(E)]$*, the factor* ϕ_3 *can be rewritten as* $\phi_3(\#_E[Com(E)], Rev)$*, where*

$$\begin{aligned} \phi_3([2,0],h) &= \varphi_1, \quad \phi_3([2,0],l) = \varphi_2, \quad \phi_3([1,1],h) = \varphi_3, \\ \phi_3([1,1],l) &= \varphi_4, \quad \phi_3([0,2],h) = \varphi_5, \quad \phi_3([0,2],l) = \varphi_6. \end{aligned}$$

A tabular illustration of $\phi_3(\#_E[Com(E)], Rev)$ *is given in Table 2. Both* $\phi_3(ComA, ComB, Rev)$ *and* $\phi_3(\#_E[Com(E)], Rev)$ *encode the same information. It is important to note that a representation using a CRV such as* $\phi_3(\#_E[Com(E)], Rev)$ *can efficiently be handled by common lifted inference algorithms (Milch et al., 2008; Taghipour, 2013).*

Appendix B. Detailed Proofs

In this section, we provide extended versions of some of the proofs given in the main paper.

Theorem 6 *The application of Claim 5 to compute a subset of arguments* $\boldsymbol{C}$ *does not guarantee that the computed set* $\boldsymbol{C}$ *is actually a subset of commutative arguments.*

$\#_E[Com(E)]$	Rev	$\phi_3(\#_E[Com(E)], Rev)$
$[2,0]$	high	φ_1
$[2,0]$	low	φ_2
$[1,1]$	high	φ_3
$[1,1]$	low	φ_4
$[0,2]$	high	φ_5
$[0,2]$	low	φ_6

Table 2: A compressed representation of the factor $\phi_3(ComA, ComB, Rev)$ from Ex. 4. The randvars $ComA$ and $ComB$ have now been replaced by a CRV $\#_E[Com(E)]$, which uses histograms to represent $ComA$ and $ComB$ simultaneously. Note that, when replacing randvars by a CRV, the underlying semantics of ϕ_3 (and hence of the full joint probability distribution encoded by the FG containing ϕ_3) remains unchanged while at the same time the size of ϕ_3's potential table no longer exponentially depends on the number of randvars in the argument list of ϕ_3.

Proof We prove the claim by counterexample. Consider a factor $\phi(R_1, R_2, R_3, R_4)$ with $\text{range}(R_1) = \text{range}(R_2) = \text{range}(R_3) = \text{range}(R_4) = \{\text{high}, \text{low}\}$. Further, let $\phi(\text{low}, \text{low}, \text{high}, \text{high}) = \varphi_1$, $\phi(\text{high}, \text{high}, \text{low}, \text{low}) = \varphi_1$, and $\phi(\boldsymbol{r}) = \varphi_2$ with $\varphi_1 \neq \varphi_2$ for all remaining assignments $\boldsymbol{r}$. A tabular illustration of the factor ϕ together with its buckets is given in Table 3. Clearly, ϕ is not commutative with respect to $\{R_1, R_2, R_3, R_4\}$ because, e.g., $\phi(\text{low}, \text{high}, \text{high}, \text{low}) = \varphi_2 \neq \varphi_1 = \phi(\text{high}, \text{high}, \text{low}, \text{low})$. However, applying Claim 5 to ϕ yields the subset $\boldsymbol{C} = \{R_1, R_2, R_3, R_4\}$ as a result: The buckets $[4,0]$ and $[0,4]$ each contain a single potential and are thus skipped. In the bucket $[3,1]$, all four assignments are mapped to φ_2, forming a single group $\boldsymbol{\Psi} = \{\varphi_2, \varphi_2, \varphi_2, \varphi_2\}$. The element-wise intersection of the assignments corresponding to the potentials in $\boldsymbol{\Psi}$ is then given by

$$\begin{aligned} &(\{\text{high}\}, \{\text{high}\}, \{\text{high}\}, \{\text{low}\}) \cap (\{\text{high}\}, \{\text{high}\}, \{\text{low}\}, \{\text{high}\}) \\ \cap\ &(\{\text{high}\}, \{\text{low}\}, \{\text{high}\}, \{\text{high}\}) \cap (\{\text{low}\}, \{\text{high}\}, \{\text{high}\}, \{\text{high}\}) \\ =\ &(\emptyset, \emptyset, \emptyset, \emptyset) \end{aligned}$$

and hence empty at every position, yielding $\boldsymbol{C}_{[3,1]} = \{R_1, R_2, R_3, R_4\}$. Analogously, the bucket $[1,3]$ yields $\boldsymbol{C}_{[1,3]} = \{R_1, R_2, R_3, R_4\}$. In the bucket $[2,2]$, the group $\boldsymbol{\Psi} = \{\varphi_1, \varphi_1\}$ (with corresponding assignments $(\text{low}, \text{low}, \text{high}, \text{high})$ and $(\text{high}, \text{high}, \text{low}, \text{low})$) yields the element-wise intersection $(\emptyset, \emptyset, \emptyset, \emptyset)$ and hence $\boldsymbol{C}_{[2,2]} = \{R_1, R_2, R_3, R_4\}$ (the same holds for the group $\boldsymbol{\Psi} = \{\varphi_2, \varphi_2, \varphi_2, \varphi_2\}$ in bucket $[2,2]$). Thus, $\boldsymbol{C} = \boldsymbol{C}_{[3,1]} \cap \boldsymbol{C}_{[2,2]} \cap \boldsymbol{C}_{[1,3]} = \{R_1, R_2, R_3, R_4\}$, yet ϕ is not commutative with respect to $\boldsymbol{C} = \{R_1, R_2, R_3, R_4\}$. ■

Theorem 8 *Let $\phi(R_1, \ldots, R_n)$ denote a factor that is given as an input to DECOR+ (Alg. 1). Then, the set returned by DECOR+ contains exactly the maximum sized subsets of commutative arguments of ϕ's arguments, or the empty set if no such subset exists.*

R_1	R_2	R_3	R_4	$\phi(R_1, R_2, R_3, R_4)$	b
high	high	high	high	φ_2	$[4, 0]$
high	high	high	low	φ_2	$[3, 1]$
high	high	low	high	φ_2	$[3, 1]$
high	high	low	low	φ_1	$[2, 2]$
high	low	high	high	φ_2	$[3, 1]$
high	low	high	low	φ_2	$[2, 2]$
high	low	low	high	φ_2	$[2, 2]$
high	low	low	low	φ_2	$[1, 3]$
low	high	high	high	φ_2	$[3, 1]$
low	high	high	low	φ_2	$[2, 2]$
low	high	low	high	φ_2	$[2, 2]$
low	high	low	low	φ_2	$[1, 3]$
low	low	high	high	φ_1	$[2, 2]$
low	low	high	low	φ_2	$[1, 3]$
low	low	low	high	φ_2	$[1, 3]$
low	low	low	low	φ_2	$[0, 4]$

b	$\phi^{\succ}(b)$
$[4, 0]$	$\langle \varphi_2 \rangle$
$[3, 1]$	$\langle \varphi_2, \varphi_2, \varphi_2, \varphi_2 \rangle$
$[2, 2]$	$\langle \varphi_1, \varphi_2, \varphi_2, \varphi_2, \varphi_2, \varphi_1 \rangle$
$[1, 3]$	$\langle \varphi_2, \varphi_2, \varphi_2, \varphi_2 \rangle$
$[0, 4]$	$\langle \varphi_2 \rangle$

Table 3: The potential table of the factor $\phi(R_1, R_2, R_3, R_4)$ used in the proof of Thm. 6 (left) and the multiset of potentials $\phi^{\succ}(b)$ associated with each bucket $b \in \mathcal{B}(\phi)$ (right). Only the assignments $(\text{low}, \text{low}, \text{high}, \text{high})$ and $(\text{high}, \text{high}, \text{low}, \text{low})$ are mapped to φ_1, while all remaining assignments are mapped to $\varphi_2 \neq \varphi_1$.

Proof We split the proof into two parts. First, we show that the bucket loop in Alg. 1 computes a set $\boldsymbol{C}_{cand}$ of candidate subsets that is guaranteed to contain every maximum sized subset of commutative arguments of ϕ. Second, we show that the subsequent verification step identifies a maximum sized subset of commutative arguments among the candidates.

Correctness of the bucket loop. Let $\boldsymbol{C}^* \subseteq \{R_1, \ldots, R_n\}$ with $|\boldsymbol{C}^*| \geq 2$ denote any maximum sized subset of commutative arguments of ϕ. By Thm. 7, for every bucket $b \in \mathcal{B}(\phi)$ with $|\phi^{\succ}(b)| > 1$, there exists a maximal set of identical potentials $\boldsymbol{\Psi}_i \subseteq \phi^{\succ}(b)$ with $|\boldsymbol{\Psi}_i| > 1$ such that the element-wise intersection of the assignments corresponding to the potentials in $\boldsymbol{\Psi}_i$ contains an empty set at every position $j \in \{1, \ldots, n\}$ for which $R_j \in \boldsymbol{C}^*$. Consequently, the set $\boldsymbol{C}_i$ computed for $\boldsymbol{\Psi}_i$ in the inner loop of Alg. 1 (Line 9) satisfies $\boldsymbol{C}^* \subseteq \boldsymbol{C}_i$, and hence either $\boldsymbol{C}_i$ itself is added to $\boldsymbol{C}'$ or it is subsumed by another candidate subset already in $\boldsymbol{C}'$ that contains $\boldsymbol{C}^*$. The cross-intersection step preserves $\boldsymbol{C}^*$ as a subset of some entry in $\boldsymbol{C}_{cand}$ at every iteration, because the intersection of two supersets of $\boldsymbol{C}^*$ is itself a superset of $\boldsymbol{C}^*$ and the subsumption check only removes entries that are themselves subsumed by a superset. By induction over the buckets, every maximum sized subset $\boldsymbol{C}^*$ of commutative arguments of ϕ is contained in some entry of $\boldsymbol{C}_{cand}$ after the bucket loop terminates.

Correctness of the verification step. After the bucket loop, the verification step iterates over the candidate subsets in $\boldsymbol{C}_{cand}$ in order of decreasing size and, for each candidate

$\boldsymbol{C} \in \boldsymbol{C}_{cand}$, checks whether ϕ is commutative with respect to $\boldsymbol{C}$ and, if not, recursively checks all $(|\boldsymbol{C}| - 1)$-element subsets of $\boldsymbol{C}$, then all $(|\boldsymbol{C}| - 2)$-element subsets of $\boldsymbol{C}$, and so on, until either a commutative subset is found or no subset of size at least two remains. By the first part of the proof, every maximum sized subset of commutative arguments of ϕ is contained in some entry of $\boldsymbol{C}_{cand}$, and the verification step thus eventually checks every such maximum sized subset. The check itself is exact (it directly applies Def. 2), so the first commutative subset returned by the verification step is necessarily a subset of commutative arguments of ϕ. Moreover, as the verification step considers candidates in order of decreasing size and stops as soon as a commutative subset is found, the returned subset is maximum sized—any larger commutative subset would have been considered (and accepted) earlier. If no candidate passes the commutativity check at any size, then by the first part of the proof (each maximum sized subset of commutative arguments is contained in $\boldsymbol{C}_{cand}$), ϕ admits no commutative subset of size at least two, and Alg. 1 correctly returns the empty set. ■

Appendix C. Example Run of the DECOR+ Algorithm

The following example showcases an example run of the DECOR+ algorithm (Alg. 1) on the factor $\phi_3(ComA, ComB, Rev)$ from Table 1.

Example 5 *Let us take a look at how DECOR+ computes a maximum sized subset of commutative arguments for the factor* $\phi_3(ComA, ComB, Rev)$ *from Table 1. Initially, DECOR+ starts with the set* $\boldsymbol{C}_{cand} = \{\{ComA, ComB, Rev\}\}$ *containing the single candidate subset of all arguments. The buckets* $[3, 0]$ *and* $[0, 3]$ *each contain only a single potential and are thus skipped by DECOR+. For the bucket* $[2, 1]$ *with* $\phi_3^{\succ}([2, 1]) = \langle\varphi_2, \varphi_3, \varphi_3\rangle$*, DECOR+ finds the maximal group of identical potentials* $\boldsymbol{\Psi}_1 = \{\varphi_3, \varphi_3\}$ *corresponding to the assignments* $(\text{high}, \text{low}, \text{high})$ *and* $(\text{low}, \text{high}, \text{high})$*. The remaining group* $\{\varphi_2\}$ *contains less than two elements and is thus ignored. DECOR+ then computes the element-wise intersection of the assignments corresponding to the potentials in* $\boldsymbol{\Psi}_1$*, which is given by* $(\{\text{high}\}, \{\text{low}\}, \{\text{high}\}) \cap (\{\text{low}\}, \{\text{high}\}, \{\text{high}\}) = (\emptyset, \emptyset, \{\text{high}\})$*. As the element-wise intersection is empty at positions one and two, DECOR+ adds the set* $\boldsymbol{C}_1 = \{ComA, ComB\}$ *to the set* $\boldsymbol{C}'$ *of candidate subsets for the bucket* $[2, 1]$*. Afterwards, DECOR+ computes the intersection of* $\{ComA, ComB, Rev\}$ *(as* $\boldsymbol{C}_{cand} = \{\{ComA, ComB, Rev\}\}$*) and* $\{ComA, ComB\}$ *(as* $\boldsymbol{C}' = \{\{ComA, ComB\}\}$*), which yields* $\boldsymbol{C}_{\cap} = \{\{ComA, ComB\}\}$ *and thus also* $\boldsymbol{C}_{cand} = \{\{ComA, ComB\}\}$ *for the next iteration. For the bucket* $[1, 2]$ *with* $\phi_3^{\succ}([1, 2]) = \langle\varphi_4, \varphi_4, \varphi_5\rangle$*, DECOR+ finds the maximal group of identical potentials* $\boldsymbol{\Psi}_1 = \{\varphi_4, \varphi_4\}$ *corresponding to the assignments* $(\text{high}, \text{low}, \text{low})$ *and* $(\text{low}, \text{high}, \text{low})$*, and computes the element-wise intersection* $(\{\text{high}\}, \{\text{low}\}, \{\text{low}\}) \cap (\{\text{low}\}, \{\text{high}\}, \{\text{low}\}) = (\emptyset, \emptyset, \{\text{low}\})$*. Again, positions one and two are empty and thus DECOR+ adds the set* $\boldsymbol{C}_1 = \{ComA, ComB\}$ *to the set* $\boldsymbol{C}'$ *of candidate subsets for the bucket* $[1, 2]$*. Thereafter, DECOR+ computes the intersection of* $\{ComA, ComB\}$ *(as* $\boldsymbol{C}_{cand} = \{\{ComA, ComB\}\}$*) and* $\{ComA, ComB\}$ *(as* $\boldsymbol{C}' = \{\{ComA, ComB\}\}$*), which yields* $\boldsymbol{C}_{cand} = \boldsymbol{C}_{\cap} = \{\{ComA, ComB\}\}$*. As there are no further buckets left, DECOR+ proceeds to the verification step and checks whether* ϕ_3 *is commutative with respect to* $\{ComA, ComB\}$*. The check succeeds and DECOR+ returns the set* $\{\{ComA, ComB\}\}$ *as a set containing all maximum sized subsets of commutative arguments of* ϕ_3*.*

Appendix D. Correctness and Efficiency Improvements of A-DECOR

In this section, we first formally state and prove the correctness of the A-DECOR algorithm (Alg. 2), and afterwards also show how A-DECOR can be implemented efficiently by computing connected components in a commutativity graph.

D.1. Correctness of the A-DECOR Algorithm

We now show that for a given factor ϕ, A-DECOR (Alg. 2) returns exactly the set of all commutative subsets of ϕ's arguments of maximum size.

Theorem 12 *Let $\phi(R_1, \ldots, R_n)$ denote a factor that is given as an input to A-DECOR (Alg. 2). Then, the set returned by A-DECOR contains exactly the maximum sized subsets of commutative arguments of ϕ, or the empty set if no such subset exists.*

Proof We first show by induction on k that $\boldsymbol{L}_k$ contains exactly the commutative subsets of $\{R_1, \ldots, R_n\}$ of size k.

Base case ($k = 2$): $\boldsymbol{L}_2$ is constructed in Line 1 by directly checking commutativity for every pair of arguments and contains exactly the commutative pairs.

Inductive step ($k \to k+1$): Assume that $\boldsymbol{L}_k$ contains exactly the commutative subsets of size k. Let $\boldsymbol{C}'$ denote any commutative subset of size $k+1$. We first show that any truly commutative subset $\boldsymbol{C}'$ of size $k+1$ is added to $\boldsymbol{L}_{k+1}$ in Line 7. By Prop. 10, every subset of $\boldsymbol{C}'$ of size at least two is also commutative; in particular, $\boldsymbol{C}' \setminus \{R_i\} \in \boldsymbol{L}_k$ for any $R_i \in \boldsymbol{C}'$ (by the induction hypothesis), and every pair $\{R_i, R_j\} \subseteq \boldsymbol{C}'$ with $i \neq j$ is contained in $\boldsymbol{L}_2$. Picking any $R_i \in \boldsymbol{C}'$ and setting $\boldsymbol{C} = \boldsymbol{C}' \setminus \{R_i\}$, the iteration of Lines 5 and 6 considers the candidate $\boldsymbol{C}' = \boldsymbol{C} \cup \{R_i\}$ (which is indeed commutative), and the pruning condition in Line 7 is satisfied because every pair $\{R_i, R_j\}$ with $R_j \in \boldsymbol{C}$ is in $\boldsymbol{L}_2$. Hence $\boldsymbol{C}'$ is added to $\boldsymbol{L}_{k+1}$. Conversely, suppose a subset $\boldsymbol{C}' = \boldsymbol{C} \cup \{R_i\}$ is added to $\boldsymbol{L}_{k+1}$ in Line 7. We next show that $\boldsymbol{C}'$ is indeed commutative. In particular, every pair $\{R_i, R_j\}$ with $R_j \in \boldsymbol{C}$ is in $\boldsymbol{L}_2$ by the pruning condition in Line 7, and every pair $\{R_j, R_\ell\} \subseteq \boldsymbol{C}$ with $j \neq \ell$ is in $\boldsymbol{L}_2$ by Prop. 10 (since $\boldsymbol{C} \in \boldsymbol{L}_k$ is commutative by the induction hypothesis). Thus, every pair of arguments in $\boldsymbol{C}'$ is in $\boldsymbol{L}_2$, and Thm. 11 implies that $\boldsymbol{C}'$ is commutative. Consequently, $\boldsymbol{L}_{k+1}$ contains exactly the commutative subsets of size $k+1$.

Hence, for every $k \geq 2$, the layer $\boldsymbol{L}_k$ computed inside the loop coincides with the set of all commutative subsets of $\{R_1, \ldots, R_n\}$ of size k. By Line 8, the variable $\boldsymbol{L}$ is updated to $\boldsymbol{L}_{k+1}$ whenever $\boldsymbol{L}_{k+1}$ is non-empty. Consequently, when the loop in Line 3 terminates with $\boldsymbol{L}_k = \emptyset$, the variable $\boldsymbol{L}$ holds the largest non-empty layer $\boldsymbol{L}_{k-1}$, which contains exactly the commutative subsets of maximum size. If no pair of arguments is commutative, then $\boldsymbol{L}_2 = \emptyset$, the loop in Line 3 does not execute, $\boldsymbol{L}$ remains the empty set as initialised in Line 2, and Alg. 2 returns the empty set. ■

D.2. A Connected-Component Algorithm for Detecting Commutative Factors

A-DECOR terminates only after iterating up to $n-1$ levels in the worst case, even though every level $k \geq 3$ relies exclusively on pair information already contained in $\boldsymbol{L}_2$ (cf. Thm. 11).

Algorithm 3 Connected-Component Detection of Commutative Factors (CC-DECOR)

Input: A factor $\phi(R_1, \ldots, R_n)$.
Output: A set containing all maximum sized subsets of commutative arguments of ϕ.

1: $\boldsymbol{L}_2 \leftarrow \{\{R_i, R_j\} \mid 1 \leq i < j \leq n \wedge \phi \text{ is commutative with respect to } \{R_i, R_j\}\}$
2: $\boldsymbol{M} \leftarrow \boldsymbol{L}_2$
3: **while** $\exists \boldsymbol{C}_1, \boldsymbol{C}_2 \in \boldsymbol{M} : \boldsymbol{C}_1 \neq \boldsymbol{C}_2 \wedge \boldsymbol{C}_1 \cap \boldsymbol{C}_2 \neq \emptyset$ **do**
4: $\quad \boldsymbol{M} \leftarrow (\boldsymbol{M} \setminus \{\boldsymbol{C}_1, \boldsymbol{C}_2\}) \cup \{\boldsymbol{C}_1 \cup \boldsymbol{C}_2\}$
5: **return** $\{\boldsymbol{C} \in \boldsymbol{M} \mid \forall \boldsymbol{C}' \in \boldsymbol{M} : |\boldsymbol{C}| \geq |\boldsymbol{C}'|\}$

A stronger consequence of transitivity allows us to skip the level-by-level enumeration entirely: Any two commutative subsets that share at least one argument can be merged into a single (larger) commutative subset.

Theorem 13 *Let $\phi(R_1, \ldots, R_n)$ denote a factor and let $\boldsymbol{C}_1, \boldsymbol{C}_2 \subseteq \{R_1, \ldots, R_n\}$ with $|\boldsymbol{C}_1|, |\boldsymbol{C}_2| \geq 2$ denote two subsets of arguments such that ϕ is commutative with respect to both $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$, and $\boldsymbol{C}_1 \cap \boldsymbol{C}_2 \neq \emptyset$. Then, ϕ is commutative with respect to $\boldsymbol{C}_1 \cup \boldsymbol{C}_2$.*

Proof Let $R_k \in \boldsymbol{C}_1 \cap \boldsymbol{C}_2$ denote a shared argument and let $\{R_i, R_j\} \subseteq \boldsymbol{C}_1 \cup \boldsymbol{C}_2$ denote any pair of arguments with $i \neq j$. If $\{R_i, R_j\} \subseteq \boldsymbol{C}_1$ or $\{R_i, R_j\} \subseteq \boldsymbol{C}_2$, then ϕ is commutative with respect to $\{R_i, R_j\}$ by Prop. 10. Otherwise, R_i and R_j lie in different subsets, and we may assume without loss of generality that $R_i \in \boldsymbol{C}_1 \setminus \boldsymbol{C}_2$ and $R_j \in \boldsymbol{C}_2 \setminus \boldsymbol{C}_1$. Then, $\{R_i, R_k\} \subseteq \boldsymbol{C}_1$ and $\{R_k, R_j\} \subseteq \boldsymbol{C}_2$, so ϕ is commutative with respect to both $\{R_i, R_k\}$ and $\{R_k, R_j\}$ by Prop. 10. Hence, ϕ is invariant under both transpositions τ_{ik} and τ_{kj} (i.e., the transpositions swapping indices i and k and indices k and j, respectively) and consequently, ϕ is invariant under $\tau_{ij} = \tau_{ik} \circ \tau_{kj} \circ \tau_{ik}$ (where the composition sends $i \mapsto k \mapsto j \mapsto j$, $j \mapsto j \mapsto k \mapsto i$, and $k \mapsto i \mapsto i \mapsto k$, leaving i and j swapped while k returns to its original position) because ϕ is invariant under each individual transposition on the right-hand side. Thus, ϕ is commutative with respect to $\{R_i, R_j\}$ and as $\{R_i, R_j\}$ is an arbitrary pair of arguments in $\boldsymbol{C}_1 \cup \boldsymbol{C}_2$, ϕ is commutative with respect to every pair of arguments in $\boldsymbol{C}_1 \cup \boldsymbol{C}_2$. Theorem 11 then implies that ϕ is commutative with respect to $\boldsymbol{C}_1 \cup \boldsymbol{C}_2$. ■

Theorem 13 suggests an alternative algorithm, which we call connected-component detection of commutative factors (CC-DECOR), that operates in only two phases: It first computes the set $\boldsymbol{L}_2$ of all commutative pairs of arguments (analogously to A-DECOR) and then iteratively merges overlapping subsets of arguments until a fixed point is reached.

D.3. The CC-DECOR Algorithm

The CC-DECOR algorithm is presented in Alg. 3 and starts by computing the set $\boldsymbol{L}_2$ of all commutative pairs of arguments in Line 1 (analogously to A-DECOR). The set $\boldsymbol{M}$ is then initialised with the commutative pairs in Line 2 and iteratively updated by replacing any two distinct overlapping subsets $\boldsymbol{C}_1, \boldsymbol{C}_2 \in \boldsymbol{M}$ by their union $\boldsymbol{C}_1 \cup \boldsymbol{C}_2$ (Lines 3 and 4). By Thm. 13, every set added to $\boldsymbol{M}$ is a commutative subset of arguments, and the merging procedure terminates as soon as the subsets in $\boldsymbol{M}$ are pairwise disjoint. Finally, CC-DECOR returns the subsets of maximum cardinality among the disjoint subsets in $\boldsymbol{M}$ in Line 5.

The merging procedure in Lines 3 and 4 can equivalently be viewed as computing the connected components of the *commutativity graph* of ϕ, that is, the graph whose vertices are the arguments of ϕ and whose edges are the pairs in $\boldsymbol{L}_2$. A standard union-find data structure on $\{R_1, \ldots, R_n\}$ that calls $union(R_i, R_j)$ for every pair $\{R_i, R_j\} \in \boldsymbol{L}_2$ thus yields an implementation in which the merging phase runs in $O(n^2 \cdot \alpha(n))$ time, where α denotes the inverse Ackermann function (Tarjan, 1979). The dominant cost of CC-DECOR is therefore the computation of $\boldsymbol{L}_2$ in Line 1, which requires $\binom{n}{2}$ commutativity checks on pairs of arguments—the same cost as the initialisation step of A-DECOR.

D.4. Correctness of the CC-DECOR Algorithm

We now show that for a given factor ϕ, CC-DECOR (Alg. 3) returns exactly the set of all commutative subsets of ϕ's arguments of maximum size.

Theorem 14 *Let $\phi(R_1, \ldots, R_n)$ denote a factor that is given as an input to CC-DECOR (Alg. 3). Then, the set returned by CC-DECOR contains exactly the maximum sized subsets of commutative arguments of ϕ, or the empty set if no such subset exists.*

Proof We first show that the following invariant holds throughout the execution of the loop in Line 3: Every subset in $\boldsymbol{M}$ is a commutative subset of arguments of ϕ of size at least two. Initially, $\boldsymbol{M} = \boldsymbol{L}_2$ contains exactly the commutative pairs of arguments by construction, so the invariant holds. At each iteration, two distinct overlapping subsets $\boldsymbol{C}_1, \boldsymbol{C}_2 \in \boldsymbol{M}$ are replaced by their union $\boldsymbol{C}_1 \cup \boldsymbol{C}_2$ in Line 4, which is a commutative subset of size at least two by Thm. 13. Hence, the invariant is preserved. The loop terminates because each iteration strictly decreases $|\boldsymbol{M}|$ (two subsets are removed and one is added) and $\boldsymbol{M}$ is finite, and upon termination, the subsets in $\boldsymbol{M}$ are pairwise disjoint.

Let $K = \max_{\boldsymbol{C} \in \boldsymbol{M}} |\boldsymbol{C}|$ denote the size of the largest subset in $\boldsymbol{M}$ at termination (with $K = 0$ if $\boldsymbol{M} = \emptyset$). By the invariant, every $\boldsymbol{C} \in \boldsymbol{M}$ is a commutative subset of size at least two whenever $\boldsymbol{M} \neq \emptyset$, so the maximum size of any commutative subset of ϕ is at least K. Conversely, let $\boldsymbol{C}^* \subseteq \{R_1, \ldots, R_n\}$ with $|\boldsymbol{C}^*| \geq 2$ denote any commutative subset of ϕ. By Prop. 10, every pair $\{R_i, R_j\} \subseteq \boldsymbol{C}^*$ with $i \neq j$ is contained in $\boldsymbol{L}_2$, that is, in $\boldsymbol{M}$ at the start of the loop. Since any two such pairs that share an argument are eventually merged, induction over the merging steps yields that there exists $\boldsymbol{C} \in \boldsymbol{M}$ at termination with $\boldsymbol{C}^* \subseteq \boldsymbol{C}$, and hence $|\boldsymbol{C}^*| \leq |\boldsymbol{C}| \leq K$. Consequently, the maximum size of any commutative subset of ϕ equals K, and the maximum sized subsets in $\boldsymbol{M}$ are exactly the maximum sized commutative subsets of ϕ. The filtering step in Line 5 returns exactly those subsets of maximum cardinality in $\boldsymbol{M}$ (or the empty set if $\boldsymbol{M} = \emptyset$, that is, if no commutative subset of size at least two exists), which completes the proof. ■

Appendix E. Additional Experimental Results

In addition to the experimental results from Sec. 6, we provide further experimental results in this section. While the plots given in Fig. 2 show averages of multiple runs for different choices of the number of commutative arguments k, we now present separate plots for each of the choices of the number of commutative arguments k to highlight the influence of k on

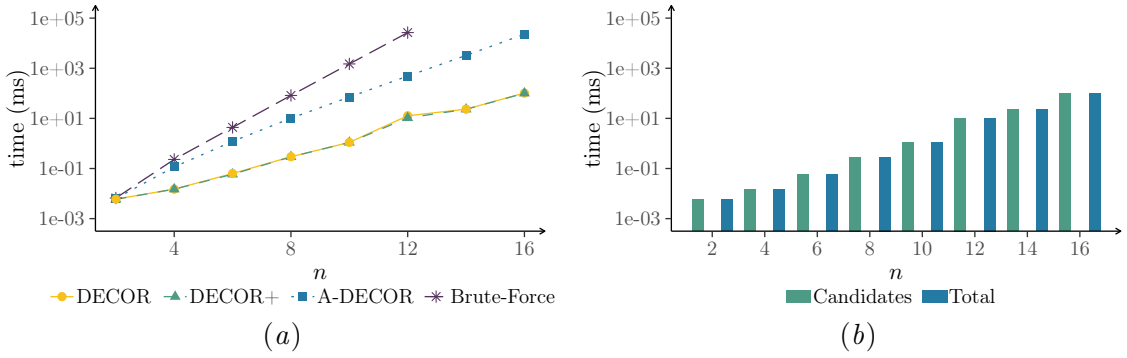


Figure 3: (*a*) Run times of DECOR, DECOR+, A-DECOR, and the brute-force algorithm for input factors with n arguments, of which $k = 0$ arguments are commutative. (*b*) Proportion of the run time of DECOR+ spent in the bucket loop and the verification step for input factors with $k = 0$ commutative arguments.

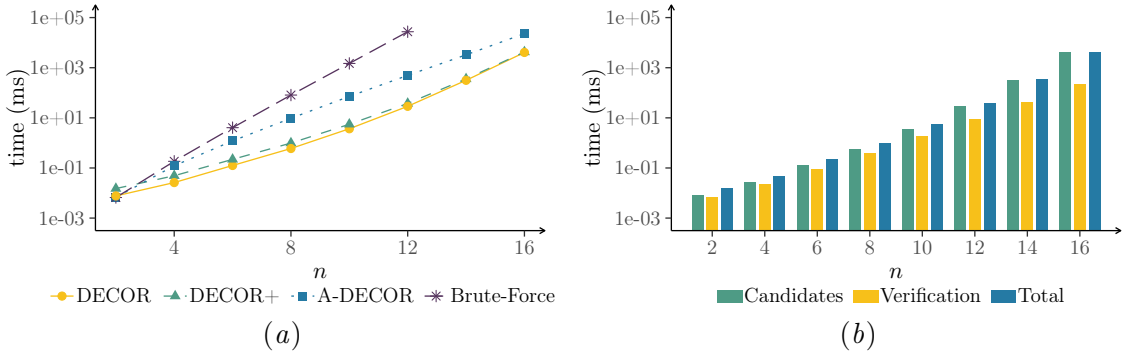


Figure 4: (*a*) Run times of DECOR, DECOR+, A-DECOR, and the brute-force algorithm for input factors with n arguments, of which $k = 2$ arguments are commutative. (*b*) Proportion of the run time of DECOR+ spent in the bucket loop and the verification step for input factors with $k = 2$ commutative arguments.

the performance on the algorithms. We again compare the run times of the original DECOR algorithm, DECOR+, A-DECOR, and the brute-force algorithm on factors with $n \in \{2, 4, 6, 8, 10, 12, 14, 16\}$ Boolean arguments and set a timeout of five minutes per instance.

The plots for individual choices of k in Figs. 3 to 8 confirm the general picture observed in Fig. 2 across all values of k: DECOR+ consistently outperforms A-DECOR and the brute-force algorithm runs into the timeout for $n > 12$—with the exception of $k = n - 1$ and $k = n$ (Figs. 7 and 8), where its decreasing-size search finds a commutative subset within a few iterations (which also explains its superior performance for the choice of $k = n$, where the brute-force algorithm finds the unique commutative subset spanning all arguments immediately). For $k = 0$ (Fig. 3), there exists no subset of commutative arguments at all, and the bucket loop of DECOR+ rules out every candidate subset immediately before even reaching the verification step, which explains the missing bars for the verification step in

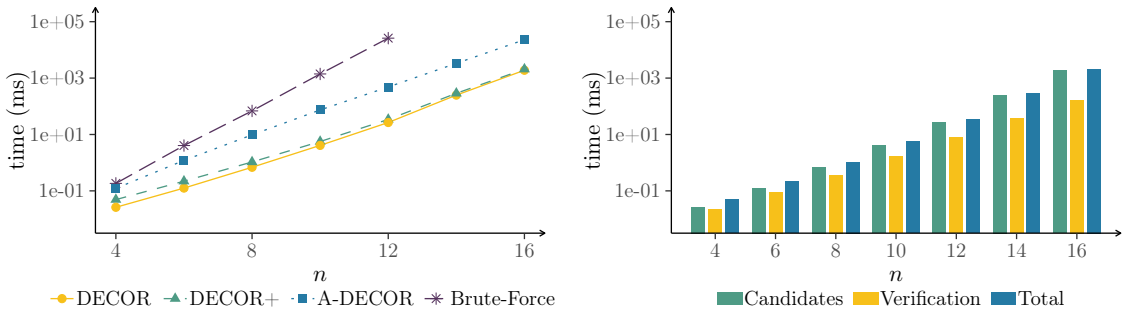


Figure 5: (*a*) Run times of DECOR, DECOR+, A-DECOR, and the brute-force algorithm for input factors with n arguments, of which $k = \lfloor \log_2 n \rfloor$ arguments are commutative. (*b*) Proportion of the run time of DECOR+ spent in the bucket loop and the verification step for input factors with $k = \lfloor \log_2 n \rfloor$ commutative arguments.

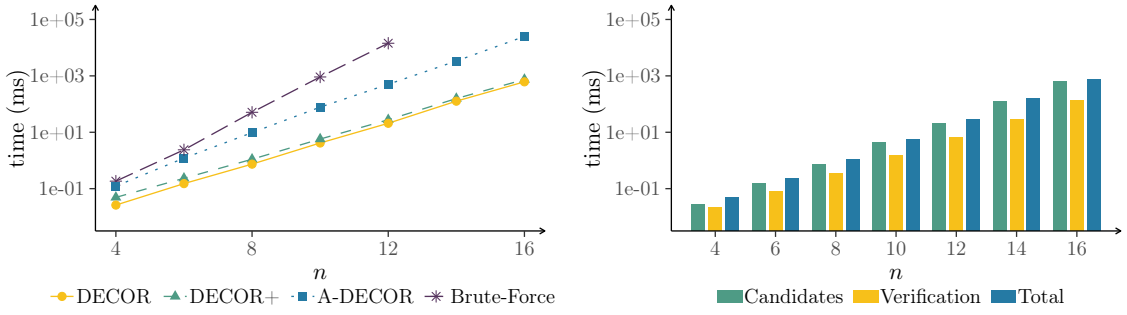


Figure 6: (*a*) Run times of DECOR, DECOR+, A-DECOR, and the brute-force algorithm for input factors with n arguments, of which $k = \lfloor n \,/\, 2 \rfloor$ arguments are commutative. (*b*) Proportion of the run time of DECOR+ spent in the bucket loop and the verification step for input factors with $k = \lfloor n \,/\, 2 \rfloor$ commutative arguments.

Fig. 3(*b*). For the remaining values of k, the verification step still contributes only a small fraction of the total run time of DECOR+, in line with the observation in Sec. 6 (again, note that due to the logarithmic scale on the y-axis, the "Candidates" and "Verification" bars do not visually add up to the "Total" bar).

So far, all considered input factors had a single subset of commutative arguments of size k. Next, we investigate how the algorithms behave when there are multiple disjoint subsets of commutative arguments (which is unlikely in practice—nevertheless an evaluation of this setting has been missing in the original DECOR paper (Luttermann et al., 2024b)). In particular, we now consider input factors with $n \in \{4, 6, 8, 10, 12, 14, 16\}$ Boolean arguments and partition the n arguments into g disjoint commutative groups of equal size $s = \lfloor n/g \rfloor$ for every divisor g of n satisfying $2 \leq g \leq n/2$ (that is, there are $k = g \cdot s$ commutative arguments in total), and we again set a timeout of five minutes per instance. Figure 9 examines how the

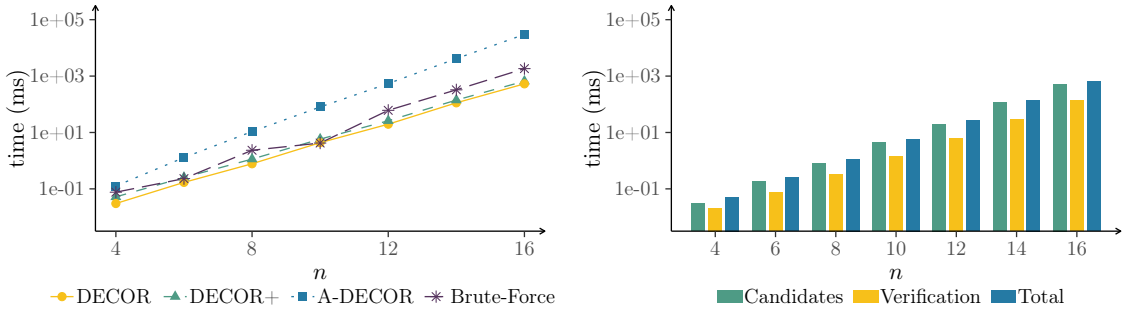


Figure 7: (*a*) Run times of DECOR, DECOR+, A-DECOR, and the brute-force algorithm for input factors with n arguments, of which $k = n-1$ arguments are commutative. (*b*) Proportion of the run time of DECOR+ spent in the bucket loop and the verification step for input factors with $k = n - 1$ commutative arguments.

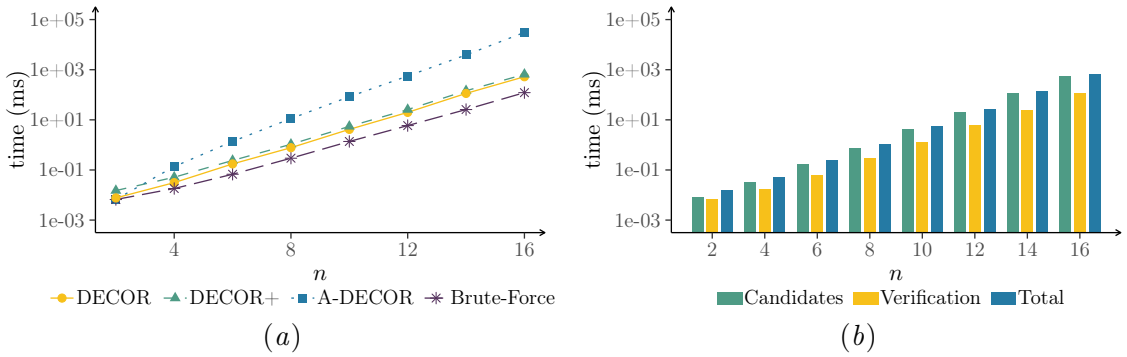


Figure 8: (*a*) Run times of DECOR, DECOR+, A-DECOR, and the brute-force algorithm for input factors with n arguments, of which $k = n$ arguments are commutative. (*b*) Proportion of the run time of DECOR+ spent in the bucket loop and the verification step for input factors with $k = n$ commutative arguments.

algorithms behave when the commutative arguments are split into g disjoint groups of size s each. The run times of DECOR+ in Fig. 9 are essentially indistinguishable from the run times observed for a single subset of commutative arguments in the previous plots, regardless of how the commutative arguments are distributed across groups: DECOR+ consistently outperforms A-DECOR for every individual choice of g and s, respectively, confirming that the bucket-based pruning of DECOR+ maintains its high effectiveness independent of the group structure of the commutative arguments.

Before we close this section, we briefly compare the A-DECOR algorithm to CC-DECOR and discuss a selection of heuristics that seem promising to improve the performance of DECOR+ in practice. We have seen that Prop. 9 provides an a-priori upper bound on the size of any commutative subset of ϕ, given by the smallest intermediate result of the candidate subsets. Since the candidate set $\boldsymbol{C}_{cand}$ is monotonically refined by intersection

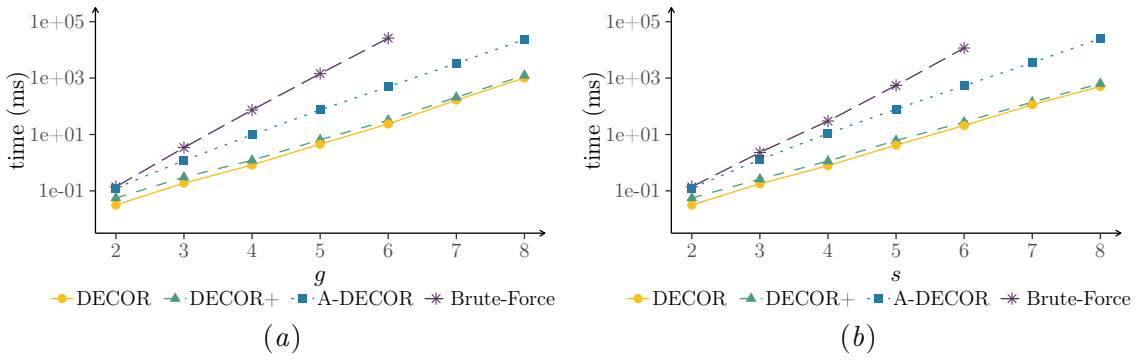


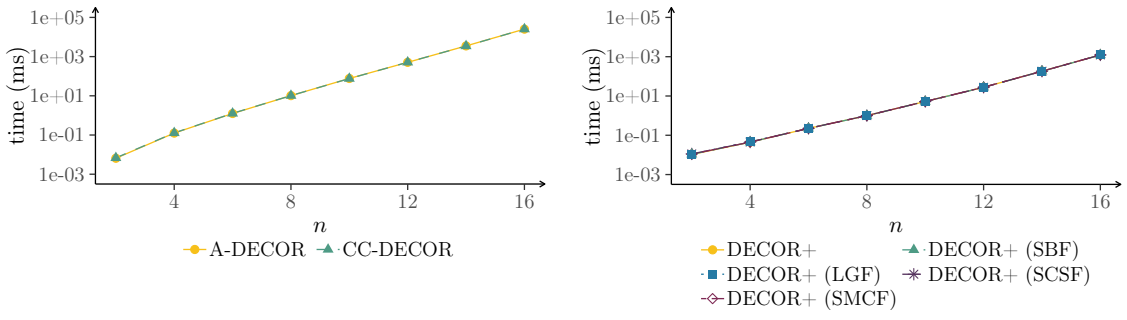
Figure 9: Run times of DECOR, DECOR+, A-DECOR, and the brute-force algorithm on input factors with n arguments partitioned into g disjoint commutative groups of size $s = n \,/\, g$ each. (a) varies g at the smallest available s for each n; (b) varies s at the smallest available g for each n.

with $\boldsymbol{C}'$ at every bucket, the order in which the buckets are processed does not affect the final result returned by DECOR+, but it might affect the size of the intermediate candidate set $\boldsymbol{C}_{cand}$ and hence the overall run time. Concretely, a bucket that produces a small, restrictive $\boldsymbol{C}'$ collapses $\boldsymbol{C}_{cand}$ early, which in turn makes every subsequent cross-intersection step cheap. Our goal hence is to find restrictive buckets early to keep intermediate results for the candidate subsets small.

To exploit this observation, we investigate the effect of a preliminary bucket-ordering pass that sorts the buckets entailed by ϕ before computing the intersections between the candidate subsets according to a heuristic that prioritises buckets that are likely to collapse $\boldsymbol{C}_{cand}$ early. We consider four heuristics, ordered by increasing pre-processing cost:

1. **Smallest bucket first (SBF).** Sort the buckets in $\mathcal{B}(\phi)$ by $|\phi^{\succ}(b)|$ in ascending order.
2. **Least groups first (LGF).** Sort the buckets by the number of maximal groups of identical potentials of size at least two in ascending order.
3. **Smallest candidate set first (SCSF).** Sort the buckets by the size of the candidate set $\boldsymbol{C}'$ in ascending order.
4. **Smallest minimal candidate first (SMCF).** Sort the buckets by the size of the smallest minimal candidate $\min_{\boldsymbol{C}_i \in \boldsymbol{C}'} |\boldsymbol{C}_i|$ in ascending order.

Figure 10(a) compares A-DECOR to CC-DECOR. The run times of both algorithms are essentially indistinguishable, indicating that the merging procedure plays a minor role since the dominant cost of both algorithms is the initial computation of all pairwise commutative arguments. Furthermore, Fig. 10(b) compares the run times of DECOR+ and its four bucket-ordering heuristics. Here, again, the four heuristic variants of DECOR+ achieve run times indistinguishable from the run time of plain DECOR+, showcasing that the bucket order does not have a significant influence on the overall run time of DECOR+ (probably because the pruning step produces intermediate results independent of the bucket order).



(a) (b)

Figure 10: (a) Average run times of A-DECOR and CC-DECOR for input factors with n arguments. (b) Average run times of DECOR+ and its four bucket-ordering heuristics for input factors with n arguments.