Efficient Detection of Exchangeable Factors in Factor Graphs*

Malte Luttermann[1], Johann Machemer[2] and **Marcel Gehrke[2]**
[1] German Research Center for Artificial Intelligence (DFKI), Lübeck, Germany
[2] Institute of Information Systems, University of Lübeck, Lübeck, Germany
malte.luttermann@dfki.de, johann.machemer@student.uni-luebeck.de, gehrke@ifis.uni-luebeck.de

Abstract

To allow for tractable probabilistic inference with respect to domain sizes, lifted probabilistic inference exploits symmetries in probabilistic graphical models. However, checking whether two factors encode equivalent semantics and hence are exchangeable is computationally expensive. In this paper, we efficiently solve the problem of detecting exchangeable factors in a factor graph. In particular, we introduce the *detection of exchangeable factors (DEFT)* algorithm, which allows us to drastically reduce the computational effort for checking whether two factors are exchangeable in practice. While previous approaches iterate all $O(n!)$ permutations of a factor's argument list in the worst case (where n is the number of arguments of the factor), we prove that DEFT efficiently identifies restrictions to drastically reduce the number of permutations and validate the efficiency of DEFT in our empirical evaluation.

1 Introduction

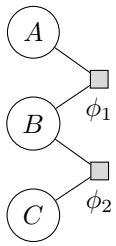
Probabilistic graphical models compactly encode a full joint probability distribution as a factorisation and provide a well-founded formalism to reason under uncertainty, e.g., when performing automated planning and acting. Reasoning under uncertainty, however, might become computationally expensive when using propositional probabilistic models such as Bayesian networks, Markov networks, or factor graphs (FGs). In general, probabilistic inference (i.e., the computation of marginal distributions of random variables (randvars) given observations for other randvars) scales exponentially with the number of randvars in the Bayesian network, Markov network, or FG, respectively, in the worst case. To allow for tractable probabilistic inference (e.g., probabilistic inference requiring polynomial time) with respect to domain sizes of logical variables, lifted probabilistic inference algorithms exploit symmetries in a probabilistic graphical model by using a representative of indistinguishable individuals for computations. However, to exploit symmetries in a probabilistic graphical model, these symmetries must be detected first and therefore, we investigate the problem of efficiently detecting exchangeable factors (i.e., factors that encode the same underlying function regardless of the order of their arguments) in FGs in this paper.

In previous work, Poole (2003) introduces parametric factor graphs (PFGs) and lifted variable elimination as a lifted inference algorithm operating on PFGs. Since then, lifted variable elimination has been refined by many researchers (De Salvo Braz, Amir, and Roth 2005; 2006; Milch et al. 2008; Kisyński and Poole 2009; Taghipour et al. 2013; Braun and Möller 2018). To perform lifted probabilistic inference, the lifted representation (e.g., the PFG) has to be obtained first. The commonly used colour passing (CP) algorithm (initially named as "CompressFactorGraph") can be used to obtain an equivalent lifted representation of an FG (Kersting, Ahmadi, and Natarajan 2009; Ahmadi et al. 2013). Among other refinements (Luttermann, Möller, and Gehrke 2023), the CP algorithm has been extended to *advanced colour passing (ACP)* (Luttermann et al. 2024), which transforms a given FG into a PFG entailing equivalent semantics as the initial FG and, in contrast to CP, does not require exchangeable factors to have their potential mappings specified in a specific order. However, the offline-step required by ACP to detect exchangeable factors is computationally expensive.

To efficiently detect exchangeable factors in FGs, we contribute the *detection of exchangeable factors (DEFT)* algorithm and show in an in-depth theoretical analysis that DEFT avoids the expensive computation of permutations in many practical settings, making the search for exchangeable factors feasible in practice. More specifically, we analyse both the complexity of previous approaches and the problem itself and then apply the theoretical insights to obtain DEFT as a practical algorithm. To tackle the problem of detecting exchangeable factors, we make use of so-called buckets that count the occurrences of specific range values in an assignment for a (sub)set of a factor's arguments. We show that using buckets, the exchangeability of factors can be detected in a highly efficient manner, thereby allowing us to drastically reduce the number of permutations to be checked.

The remaining part of this paper is structured as follows. We begin by introducing necessary background information and notations. Thereafter, we delve into the problem of detecting exchangeable factors in FGs and provide an in-depth

*Extended version of paper accepted to the Proceedings of the 37th International FLAIRS Conference (FLAIRS-24).



A	B	$\phi_1(A,B)$
true	true	φ_1
true	false	φ_2
false	true	φ_3
false	false	φ_4

C	B	$\phi_2(C,B)$
true	true	φ_1
true	false	φ_2
false	true	φ_3
false	false	φ_4

Figure 1: A toy example for an FG consisting of three Boolean randvars A, B, and C as well as two factors ϕ_1 and ϕ_2. The mappings of ϕ_1 and ϕ_2 are given in the respective tables on the right with $\varphi_1, \ldots, \varphi_4 \in \mathbb{R}^+$.

theoretical analysis of the problem. Finally, we present the DEFT algorithm to drastically reduce the required computational effort to detect exchangeable factors in practice and confirm its efficiency in an empirical evaluation.

2 Background

We begin by defining FGs as undirected propositional probabilistic models and then continue to introduce the notion of exchangeable factors within FGs. An FG compactly encodes a full joint probability distribution between randvars by decomposing it into a product of factors (Frey et al. 1997; Kschischang, Frey, and Loeliger 2001).

Definition 1 (Factor Graph)**.** *An* FG $G = (\boldsymbol{V}, \boldsymbol{E})$ *is an undirected bipartite graph with node set* $\boldsymbol{V} = \boldsymbol{R} \cup \boldsymbol{\Phi}$ *where* $\boldsymbol{R} = \{R_1, \ldots, R_n\}$ *is a set of variable nodes (randvars) and* $\boldsymbol{\Phi} = \{\phi_1, \ldots, \phi_m\}$ *is a set of factor nodes (functions). The term* $\mathcal{R}(R_i)$ *denotes the possible values (range) of a randvar* R_i*. There is an edge between a variable node* R_i *and a factor node* ϕ_j *in* $\boldsymbol{E} \subseteq \boldsymbol{R} \times \boldsymbol{\Phi}$ *if* R_i *appears in the argument list of* ϕ_j*. A factor is a function that maps its arguments to a positive real number, called potential. The semantics of* G *is given by*

$$P_G = \frac{1}{Z} \prod_{j=1}^{m} \phi_j(\mathcal{A}_j)$$

with Z *being the normalisation constant and* $\mathcal{A}_j$ *denoting the randvars connected to* ϕ_j *(i.e., the arguments of* ϕ_j*).*

Example 1. *Figure 1 shows a toy example for an FG consisting of three Boolean randvars* A*,* B*, and* C *as well as two factors* ϕ_1 *and* ϕ_2*. Note that in this example,* ϕ_1 *and* ϕ_2 *encode identical potentials (depicted in the tables on the right) and hence are exchangeable. More specifically, it holds that* $\phi_1(A, B) = \phi_2(C, B)$ *for all inputs where* A *and* C *are assigned the same value. Consequently, we can say that* A *and* C *are exchangeable, enabling us to treat them equally during probabilistic inference.*

In general, grouping together identically behaving objects and using only a single representative for each group is the core idea behind lifted inference (Niepert and Van den Broeck 2014). Lifted inference algorithms exploit symmetries in a probabilistic graphical model by operating on PFGs which consist of parameterised randvars and parametric factors, representing sets of randvars and factors, respectively (Poole 2003). Coming back to Ex. 1, recall that the semantics of the FG G depicted in Fig. 1 is given by $P_G = \frac{1}{Z} \cdot \phi_1(A, B) \cdot \phi_2(C, B)$. As A and C (and hence ϕ_1 and ϕ_2) are exchangeable, we can use a single representative factor that represents a group of identically behaving factors (here ϕ_1 and ϕ_2) and take it to the power of the group size instead of multiplying each factor separately.

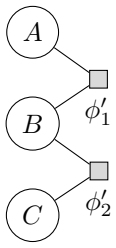
Symmetries in FGs occur not only in our toy example but are highly relevant in various real world domains such as an epidemic domain where each individual person influences the probability of an epidemic in the same way—because the probability of having an epidemic depends on the number of sick people and not on individual people being sick. More specifically, the probability for an epidemic is the same if there are three sick people and the remaining people in the universe are not sick, independent of whether $alice$, bob, and eve or $charlie$, $dave$, and $fred$ are sick, for example. Analogously, in a movie domain the popularity of an actor influences the success of a movie in the same way for each actor, in a research domain the quality of every publication influences the quality of a conference equally, and so on.

To detect symmetries in an FG and exploit them to speed up probabilistic inference, the ACP algorithm (Luttermann et al. 2024), which generalises the CP algorithm (Kersting, Ahmadi, and Natarajan 2009; Ahmadi et al. 2013), is the state of the art. ACP deploys a subroutine that checks whether two factors are exchangeable to find out which factors should be grouped together. Before we investigate how this subroutine can efficiently be realised, we give a formal definition of exchangeable factors.

Definition 2 (Exchangeable Factors)**.** *Let* $\phi_1(R_1, \ldots, R_n)$ *and* $\phi_2(R'_1, \ldots, R'_n)$ *denote two factors in an FG* G*. Then,* ϕ_1 *and* ϕ_2 *represent equivalent potentials if and only if there exists a permutation* π *of* $\{1, \ldots, n\}$ *such that for all* $r_1, \ldots, r_n \in \times_{i=1}^{n} \mathcal{R}(R_i)$ *it holds that* $\phi_1(r_1, \ldots, r_n) = \phi_2(r_{\pi(1)}, \ldots, r_{\pi(n)})$*. Factors that represent equivalent potentials are called* exchangeable*.*

Note that two factors must have the same number of arguments n to be able to be exchangeable because otherwise they cannot encode the same underlying function. Further, it must hold that there exists a bijection $\tau \colon \{R_1, \ldots, R_n\} \to \{R'_1, \ldots, R'_n\}$ which maps each R_i to an R'_j such that $\mathcal{R}(R_i) = \mathcal{R}(R'_j)$. In other words, the ranges of the arguments of two exchangeable factors must be the same to ensure that the two functions encoded by the factors are defined over the same function domain.

Example 2. *Take a look at the FG* G *illustrated in Fig. 2.* G *entails equivalent semantics as the FG depicted in Fig. 1 because* $\phi_1(A, B) = \phi'_1(A, B)$ *and* $\phi_2(C, B) = \phi'_2(B, C)$ *for all possible assignments of* A*,* B*, and* C*. More specifically, in* ϕ'_2*, the order of the arguments* B *and* C *is swapped compared to their order in* ϕ_2*. The potentials, however, are still the same as* $\phi_2(C = \text{true}, B = \text{false}) = \phi'_2(B =$



A	B	$\phi'_1(A,B)$
true	true	φ_1
true	false	φ_2
false	true	φ_3
false	false	φ_4

B	C	$\phi'_2(B,C)$
true	true	φ_1
true	false	φ_3
false	true	φ_2
false	false	φ_4

Figure 2: Another toy example for an FG consisting of three Boolean randvars A, B, and C as well as two factors ϕ_1 and ϕ_2. Observe that the full joint probability distribution encoded by the illustrated FG is the same as the probability distribution encoded by the FG depicted in Fig. 1.

$\text{false}, C = \text{true}) = \varphi_2$ *and* $\phi_2(C = \text{false}, B = \text{true}) = \phi'_2(B = \text{true}, C = \text{false}) = \varphi_3$. *Consequently, all of the four factors* ϕ_1*,* ϕ_2*,* ϕ'_1*, and* ϕ'_2 *are exchangeable.*

As we have seen in Ex. 2, it is not necessarily the case that the tables of the input-output mappings of two exchangeable factors are identical. For practical applications, it is far too strong an assumption to make that the tables of exchangeable factors are always ordered such that they read identical values from top to bottom. Therefore, we next briefly recap the previous approach to detect exchangeable factors and afterwards continue to provide an in-depth investigation of the problem of detecting exchangeable factors in FGs, allowing us to pour the gained insights into the DEFT algorithm to efficiently detect exchangeable factors in practice.

3 Efficient Detection of Exchangeable Factors Using Buckets

A straightforward approach to check whether two factors $\phi_1(R_1, \ldots, R_n)$ and $\phi_2(R'_1, \ldots, R'_n)$ are exchangeable is to loop over all permutations of one of the factors' argument lists, say the argument list of ϕ_2, and then check for each permutation whether ϕ_1 and ϕ_2 map to the same potential for all assignments of their arguments (i.e., whether both tables read identical values from top to bottom after the rearrangement of ϕ_2's arguments according to the permutation). Luttermann et al. (2024) give a more efficient approach where histograms (which we call *buckets* in this paper) entailed by so-called counting randvars are used to enforce a necessary condition for two factors to be exchangeable, thereby allowing to heavily prune the search space and check the permutations only after the initial filtering succeeds. Nevertheless, the approach still checks all $n!$ permutations after the initial filtering succeeds. To provide a theoretical analysis and then show how the approach can be drastically improved, we first introduce the notion of a bucket.

Definition 3 (Bucket)**.** *Let* $\phi(R_1, \ldots, R_n)$ *be a factor and let* $\boldsymbol{S} \subseteq \{R_1, \ldots, R_n\}$ *denote a subset of* ϕ*'s arguments such that* $\mathcal{R}(R_i) = \mathcal{R}(R_j)$ *for all* $R_i, R_j \in \boldsymbol{S}$*. Further, let* $\mathcal{V}$ *denote the range of the elements in* $\boldsymbol{S}$ *(identical for all* $R_i \in \boldsymbol{S}$*). Then, a* bucket b *entailed by* $\boldsymbol{S}$ *is a set of tuples* $\{(v_i, n_i)\}_{i=1}^{|\mathcal{V}|}$*,* $v_i \in \mathcal{V}$*,* $n_i \in \mathbb{N}$*, and* $\sum_i n_i = |\boldsymbol{S}|$*, such that* n_i *specifies the number of occurrences of value* v_i *in an assignment for all randvars in* $\boldsymbol{S}$*. A shorthand notation for* $\{(v_i, n_i)\}_{i=1}^{|\mathcal{V}|}$ *is* $[n_1, \ldots, n_{|\mathcal{V}|}]$*. In abuse of notation, we denote by* $\phi(b)$ *the multiset of potentials the assignments represented by* b *are mapped to by* ϕ*.*

Example 3. *Consider the factor* $\phi'_1(A, B)$ *from Fig. 2 and let* $\boldsymbol{S} = \{A, B\}$ *with* $\mathcal{R}(A) = \mathcal{R}(B) = \{\text{true}, \text{false}\}$*. Then,* $\boldsymbol{S}$ *entails the three buckets* $\{(\text{true}, 2), (\text{false}, 0)\}$*,* $\{(\text{true}, 1), (\text{false}, 1)\}$*, and* $\{(\text{true}, 0), (\text{false}, 2)\}$*—or* $[2, 0]$*,* $[1, 1]$*, and* $[0, 2]$ *in shorthand notation. According to the mappings depicted in the table shown in Fig. 2, it holds that* $\phi'_1([2, 0]) = \langle\varphi_1\rangle$*,* $\phi'_1([1, 1]) = \langle\varphi_2, \varphi_3\rangle$*, and* $\phi'_1([0, 2]) = \langle\varphi_4\rangle$*.*

In other words, buckets count the occurrences of specific range values in an assignment for a subset of a factor's arguments. Hence, every bucket may be mapped to multiple potentials while every potential (row in the table of mappings of a factor) belongs to exactly one bucket.

Buckets over Multiple Ranges

The reason we define buckets over a subset of a factor's arguments (instead of over the whole argument list) is that we are looking for a permutation of arguments having the same range such that the potential mappings of two factors are identical. The following example illustrates this point.

Example 4. *Consider* $\phi_1(R_1, R_2, R_3)$ *and* $\phi_2(R_4, R_5, R_6)$ *with* $\mathcal{R}(R_1) = \mathcal{R}(R_2) = \mathcal{R}(R_4) = \mathcal{R}(R_5) = \{\text{true}, \text{false}\}$ *and* $\mathcal{R}(R_3) = \mathcal{R}(R_6) = \{\text{low}, \text{medium}, \text{high}\}$*. To be able to obtain identical tables of potential mappings for* ϕ_1 *and* ϕ_2*,* R_3 *and* R_6 *must be located at the same position in the respective argument list of* ϕ_1 *and* ϕ_2 *because they are the only arguments having a non-Boolean range. Therefore, we compare the buckets entailed by* $\boldsymbol{S}_1 = \{R_1, R_2\}$ *and* $\boldsymbol{S}_2 = \{R_4, R_5\}$ *separately from the buckets entailed by* $\boldsymbol{S}_3 = \{R_3\}$ *and* $\boldsymbol{S}_4 = \{R_6\}$ *when checking whether* ϕ_1 *and* ϕ_2 *are exchangeable.*

As we are interested in using buckets to check whether arguments with the same range can be permuted such that the tables of two factors are identical, from now on we consider only factors with arguments having the same range. If this simplification does not hold, i.e., if a factor contains arguments with different ranges $\mathcal{R}_1, \ldots, \mathcal{R}_k$, the buckets for all arguments with range $\mathcal{R}_i$ must be compared separately for each $i = 1, \ldots, k$. Having introduced the notion of buckets, we are now able to investigate the complexity of detecting exchangeable factors with the help of buckets.

Properties of Buckets

Before we take a closer look at using buckets for detecting exchangeable factors, we briefly state the complexity of the problem of detecting exchangeable factors.

Theorem 1. *Let* $\phi_1(R_1, \ldots, R_n)$ *and* $\phi_2(R'_1, \ldots, R'_n)$ *denote two factors. The number of table comparisons needed to check whether* ϕ_1 *and* ϕ_2 *are exchangeable is in* $\mathcal{O}(n!)$*.*

Proof. According to Def. 2, there must exist a permutation of $\{1, \ldots, n\}$ such that ϕ_1 and ϕ_2 map to the same potential for all assignments of their arguments. As there are $n!$ permutations of $\{1, \ldots, n\}$ in total and we have to check every single permutation in the worst case, checking whether ϕ_1 and ϕ_2 are exchangeable is in $\mathcal{O}(n!)$. □

Even though we have to compare the two tables $n!$ times in the worst case, we can still drastically reduce the computational effort in many practical settings. The current state of the art, as presented by Luttermann et al. (2024), uses buckets as a necessary pre-condition that must be fulfilled for two factors to be exchangeable.

Proposition 1 (Luttermann et al., 2024)**.** *Let ϕ_1 and ϕ_2 denote two exchangeable factors. Then, ϕ_1 and ϕ_2 are defined over the same function domain and hence their arguments entail the same buckets.*

Proposition 2 (Luttermann et al., 2024)**.** *Let ϕ_1 and ϕ_2 denote two factors. If there exists a bucket b such that $\phi_1(b) \neq \phi_2(b)$, then ϕ_1 and ϕ_2 are not exchangeable.*

The approach of checking whether ϕ_1 and ϕ_2 are exchangeable is to compute all buckets entailed by ϕ_1 and ϕ_2, respectively, and compare them pairwise to check whether they are mapped to identical multisets of potentials. If this check fails, ϕ_1 and ϕ_2 are not exchangeable and hence no permutations are computed, otherwise it is checked whether ϕ_1 and ϕ_2 have identical tables of potential mappings for all permutations of the argument list of either ϕ_1 or ϕ_2.

The computation of the buckets has no impact on the worst-case complexity as the number of buckets is always smaller than the number of rows in the table of potential mappings, which must be compared anyway. As each potential value belongs to exactly one bucket, the effort for comparing all buckets to each other is identical to the effort of comparing the two tables to each other. Therefore, applying a pre-pruning strategy that uses buckets to check whether two factors might be exchangeable at all reduces the computational effort in many practical settings. We next show that the approach can be further improved to drastically speed up the detection of exchangeable factors in practice.

A crucial observation is that two factors are exchangeable if and only if their buckets are identical under consideration of the order of the values in the buckets. In particular, we denote by $\phi^{\succ}(b)$ the ordered multiset of potentials a bucket b is mapped to by ϕ (in order of their appearance in the table of ϕ) and then prove this insight in the following theorem.

Theorem 2. *Let ϕ_1 and ϕ_2 denote two factors. Then, ϕ_1 and ϕ_2 are exchangeable if and only if there exists a permutation of their arguments such that $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$ for all buckets b entailed by the arguments of ϕ_1 and ϕ_2.*

Proof. Let ϕ_1 and ϕ_2 denote two factors. For the first direction, it holds that ϕ_1 and ϕ_2 are exchangeable. According to Def. 2, there exists a permutation of ϕ_2's arguments such that ϕ_1 and ϕ_2 have identical tables of potential mappings. Then, we have $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$ for every bucket b as both tables read identical values from top to bottom.

For the second direction, it holds that $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$ for all buckets b entailed by the arguments of ϕ_1 and ϕ_2. As the order of the values in the buckets is identical for ϕ_1 and ϕ_2, converting the buckets back to tables of potential mappings results in identical tables for ϕ_1 and ϕ_2 and consequently, ϕ_1 and ϕ_2 are exchangeable. □

To make use of Thm. 2, the basic idea is that keeping an order for the elements in each bucket allows us to detect which arguments must be swapped for two buckets to be able to exactly match. The next example illustrates this idea.

Example 5. *Let us take a look at Fig. 3. When comparing the buckets for ϕ_1 and ϕ_2, we can for example observe that the bucket $[2, 1]$ is mapped to the same multiset of values, i.e., $\phi_1([2, 1]) = \phi_2([2, 1])$, but the values are ordered differently if we order them according to their appearance in the tables, i.e., $\phi_1^{\succ}([2, 1]) \neq \phi_2^{\succ}([2, 1])$. To obtain identical orders of values for both $\phi_1^{\succ}([2, 1])$ and $\phi_2^{\succ}([2, 1])$, e.g., φ_2 has to be swapped to the first position in $\phi_2^{\succ}([2, 1])$ (i.e., φ_2 and φ_3 have to be swapped). Consequently, we can swap the assignments of ϕ_2 that map to φ_2 and φ_3 to locate φ_2 at the first position in the bucket of ϕ_2. In particular, we know that φ_2 belongs to the assignment* $(\text{false}, \text{true}, \text{true})$ *and φ_3 belongs to the assignment* $(\text{true}, \text{true}, \text{false})$*, i.e., R_4 and R_6 must be swapped in the argument list of ϕ_2. This procedure of swapping two arguments can then be repeated until the buckets of ϕ_1 and ϕ_2 are identical (or until it is clear that the buckets cannot be made identical by swapping arguments).*

In this example, the order of the potentials in the bucket $[2, 1]$ uniquely determines which values and hence which arguments must be swapped to obtain identical buckets. As soon as the buckets contain duplicate values (such as it is the case for the bucket $[1, 2]$ in Fig. 3), however, there might be multiple candidates for the next swap. We formalise this observation in the following definition.

Definition 4 (Degree of Freedom)**.** *Let $\phi(R_1, \ldots, R_n)$ be a factor and let b be a bucket entailed by $\boldsymbol{S} \subseteq \{R_1, \ldots, R_n\}$. The* degree of freedom *of b is given by*

$$\mathcal{F}(b) = \prod_{\varphi \in \text{unique}(\phi(b))} \text{count}(\phi(b), \varphi)!$$

where $\text{unique}(\phi(b))$ denotes the set of unique potentials in $\phi(b)$ and $\text{count}(\phi(b), \varphi)$ denotes the number of occurrences of potential φ in $\phi(b)$.

Then, the degree of freedom of a factor ϕ is defined as $\mathcal{F}(\phi) = \min_{b \in \{b \mid b \in \mathcal{B}(\phi) \wedge |\phi(b)| > 1\}} \mathcal{F}(b)$, where $\mathcal{B}(\phi)$ denotes the set of all buckets entailed by ϕ's arguments. We consider only buckets that are mapped to at least two potential values because buckets which are mapped to a single potential correspond to all arguments being assigned the same value and hence, there is no need to swap arguments. Consequently, we define $\mathcal{F}(\phi) = 1$ if ϕ does not entail any buckets that are mapped to at least two potential values (which is only the case for factors having less than two arguments).

Example 6. *Consider the factor ϕ_1 given in Fig. 3. It holds that $\mathcal{B}(\phi_1) = \{[3, 0], [2, 1], [1, 2], [0, 3]\}$ and $\mathcal{F}([3, 0]) = 1!$, $\mathcal{F}([2, 1]) = 1! \cdot 1! \cdot 1!$, $\mathcal{F}([1, 2]) = 1! \cdot 2!$, and $\mathcal{F}([0, 3]) = 1!$. The buckets $[2, 1]$ and $[1, 2]$ are the only ones being mapped to a multiset consisting of more than one element and thus, it holds that $\mathcal{F}(\phi_1) = \min_{b \in \{[2,1],[1,2]\}} \mathcal{F}(b) = 1$.*

R_1	R_2	R_3	ϕ_1	b
true	true	true	φ_1	$[3,0]$
true	true	false	φ_2	$[2,1]$
true	false	true	φ_3	$[2,1]$
true	false	false	φ_4	$[1,2]$
false	true	true	φ_5	$[2,1]$
false	true	false	φ_6	$[1,2]$
false	false	true	φ_6	$[1,2]$
false	false	false	φ_7	$[0,3]$

b	$\phi_1^{\succ}(b)$	$\phi_2^{\succ}(b)$
$[3,0]$	$\langle\varphi_1\rangle$	$\langle\varphi_1\rangle$
$[2,1]$	$\langle\varphi_2,\varphi_3,\varphi_5\rangle$	$\langle\varphi_3,\varphi_5,\varphi_2\rangle$
$[1,2]$	$\langle\varphi_4,\varphi_6,\varphi_6\rangle$	$\langle\varphi_6,\varphi_4,\varphi_6\rangle$
$[0,3]$	$\langle\varphi_7\rangle$	$\langle\varphi_7\rangle$

R_4	R_5	R_6	ϕ_2	b
true	true	true	φ_1	$[3,0]$
true	true	false	φ_3	$[2,1]$
true	false	true	φ_5	$[2,1]$
true	false	false	φ_6	$[1,2]$
false	true	true	φ_2	$[2,1]$
false	true	false	φ_4	$[1,2]$
false	false	true	φ_6	$[1,2]$
false	false	false	φ_7	$[0,3]$

Figure 3: Two exchangeable factors $\phi_1(R_1, R_2, R_3)$ (abbreviated as ϕ_1) and $\phi_2(R_4, R_5, R_6)$ (abbreviated as ϕ_2) and their corresponding buckets. Rearranging, for example, the arguments of ϕ_2 such that they appear in order R_5, R_6, R_4 results in identical tables of potential mappings for ϕ_1 and ϕ_2.

Note that we take the minimum degree of freedom of the buckets as the degree of freedom of a factor. Let us again take a look at ϕ_1 from Fig. 3 to explain the reason for the minimum operation. Observe that even though the bucket $[1,2]$ contains a duplicate value and hence, we do not immediately know which of the two φ_6 values belongs to which position (and therefore, we do not uniquely know which arguments to swap), the bucket $[2,1]$ fixes the order of the arguments already. In other words, it is sufficient to swap the arguments according to the order induced by the bucket $[2,1]$ and afterwards check for all other buckets b whether $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$. We next make use of this observation to present our main result and give a more precise complexity analysis of the problem of detecting exchangeable factors.

Theorem 3. *Let $\phi_1(R_1, \ldots, R_n)$ and $\phi_2(R'_1, \ldots, R'_n)$ denote two factors and let $d = \min\{\mathcal{F}(\phi_1), \mathcal{F}(\phi_2)\}$. The number of table comparisons needed to check whether ϕ_1 and ϕ_2 are exchangeable is in $\mathcal{O}(d)$.*

Proof. From Prop. 2, we know that if $\mathcal{F}(\phi_1) \neq \mathcal{F}(\phi_2)$, ϕ_1 and ϕ_2 are not exchangeable and hence we do not need to try any permutations of arguments for table comparisons. Thus, we can assume that $\mathcal{F}(\phi_1) = \mathcal{F}(\phi_2)$ for the remaining part of this proof. Let $d = \mathcal{F}(\phi_1) = \mathcal{F}(\phi_2)$. Then, it holds that there exists a bucket b' with $\mathcal{F}(b') = d$. If $\phi_1(b') \neq \phi_2(b')$, ϕ_1 and ϕ_2 are not exchangeable and we are done, so let $\phi_1(b') = \phi_2(b')$. From Thm. 2, we know that for ϕ_1 and ϕ_2 to be able to be exchangeable, there must exist a permutation of their arguments such that $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$ for all buckets b, including b'. Hence, it is sufficient to try all permutations of arguments of, say ϕ_2, for table comparison that are possible according to bucket b' to find out whether ϕ_1 and ϕ_2 are exchangeable. Note that $\phi_1^{\succ}(b') = \langle\varphi_1, \ldots, \varphi_\ell\rangle$ restricts the possible permutations of ϕ_2's arguments as the potential φ_1 must be placed at the first position in $\phi_2^{\succ}(b')$, and analogously for the other potentials. Therefore, each potential φ_i can be placed at $\mathrm{count}(\phi_2(b'), \varphi_i)$ different positions in $\phi_2^{\succ}(b')$ and we have to try all permutations of these positions, resulting in $\mathrm{count}(\phi_2(b'), \varphi_i)!$ permutations that must be checked for each unique potential φ_i. Consequently, the number of permutations permitted by b' is given by d, which completes the proof. □

Intuitively, Thm. 3 tells us that the number of different potential values within the buckets of a factor determines the amount of permutations that must be iterated over in the worst case. Fortunately, for any factor $\phi(R_1, \ldots, R_n)$, its potential values are mostly not identical in practice. Thus, $\mathcal{F}(\phi)$ is significantly smaller than $n!$ in most practical settings. In particular, $\mathcal{F}(\phi)$ is upper-bounded by $n!$.

Corollary 1. *Let $\phi(R_1, \ldots, R_n)$. It holds that $\mathcal{F}(\phi) \leq n!$.*

Next, we exploit the theoretical insights from this section to efficiently detect exchangeable factors in practice.

Algorithm 1 Detection of Exchangeable Factors (DEFT)

Input: Factors $\phi_1(R_1, \ldots, R_n)$ and $\phi_2(R'_1, \ldots, R'_m)$.
Output: true if ϕ_1 and ϕ_2 are exchangeable, else false.

```
1:  if n ≠ m ∨ B(φ1) ≠ B(φ2) then
2:      return false
                          ▷ It holds that B(φ1) = B(φ2)
3:  for each b ∈ B(φ1) do     ▷ In ascending order of F(b)
4:      if φ1(b) ≠ φ2(b) then
5:          return false
6:      C_b ← Possible swaps to obtain φ1^≻(b) = φ2^≻(b)
7:  if there exists a swap in ⋂_{b∈B(φ1)} C_b s.t. φ1 = φ2 then
8:      return true
9:  else
10:     return false
```

4 The DEFT Algorithm

We now present the DEFT algorithm to efficiently detect exchangeable factors in practical applications. Algorithm 1 presents the entire DEFT algorithm, which proceeds as follows to check whether two given factors $\phi_1(R_1, \ldots, R_n)$ and $\phi_2(R'_1, \ldots, R'_m)$ are exchangeable. First, DEFT ensures that ϕ_1 and ϕ_2 are defined over the same function domain, i.e., that they have the same number of arguments and that their arguments entail the same set of buckets. Thereafter, DEFT iterates over the buckets (which are identical for ϕ_1 and ϕ_2) in ascending order of their degree of freedom and ensures that they are mapped to the same multiset of values by ϕ_1 and ϕ_2. If this initial check fails, ϕ_1 and ϕ_2 are not exchangeable and DEFT stops. Otherwise, DEFT checks for each bucket b whether the arguments of ϕ_2 can be rearranged such that $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$. More specifically, for each

position $p \in \{1, \ldots, |\phi_2^{\succ}(b)|\}$ in $\phi_2^{\succ}(b)$, DEFT looks up all positions p' in $\phi_1^{\succ}(b)$ that contain the same potential value as $\phi_2^{\succ}(b)$ at position p. Having found all positions p', DEFT knows that the potential values at position p and p' in $\phi_2^{\succ}(b)$ are candidates for swapping to obtain $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$. DEFT then looks up the assignments (rows in the table) of ϕ_2 that correspond to the potential values at positions p and p' and builds a dictionary C_b of possible rearrangements for ϕ_2's arguments. In particular, C_b is a dictionary that maps each argument position $i \in \{1, \ldots, m\}$ to a set of possible new argument positions at which the argument R_i can be placed. Let $\mathcal{A} = (a_1, \ldots, a_m)$ and $\mathcal{A}' = (a'_1, \ldots, a'_m)$ denote the assignments that ϕ_2 maps to the potential values at positions p and p' in $\phi_2^{\succ}(b)$. To rearrange the order of potential values in $\phi_2^{\succ}(b)$ such that $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$ holds, the arguments of ϕ_2 can be rearranged in any way such that $\mathcal{A}$ maps to φ' and $\mathcal{A}'$ maps to φ afterwards. Therefore, DEFT iterates over the assignment $\mathcal{A}$ and stores an entry in C_b for each position $i \in \{1, \ldots, m\}$ containing a set of possible rearrangements $\{p_1, \ldots, p_\ell\}$ such that $a_i = a'_{p_j}$ holds for all $j \in \{1, \ldots, \ell\}$. Finally, DEFT builds the intersection over the possible rearrangements of all positions in all buckets and checks whether there is a rearrangement left such that the tables of ϕ_1 and ϕ_2 are identical.

Example 7. *Take again a look at Fig. 3 and let* $b = [2, 1]$. *DEFT begins with position* $p = 1$ *in* $\phi_2^{\succ}(b)$*, which is assigned the value* φ_3*. The value* φ_3 *is located at position* $p' = 2$ *in* $\phi_1^{\succ}(b)$ *and hence, DEFT considers the assignments* $\mathcal{A} = (\text{true}, \text{true}, \text{false})$ *and* $\mathcal{A}' = (\text{true}, \text{false}, \text{true})$ *that belong to the potential values at positions* $p = 1$ *and* $p' = 2$ *in* $\phi_2^{\succ}(b)$*. The first position in* $\mathcal{A}$ *is assigned the value* true*. In* $\mathcal{A}'$*,* true *is assigned to the first and third position and thus, position* 1 *can be rearranged either at position* 1 *or* 3*, denoted as* $1 \mapsto \{1, 3\}$*. DEFT continues this step for the remaining values in* $\mathcal{A}$ *and obtains* $1 \mapsto \{1, 3\}$*,* $2 \mapsto \{1, 3\}$*, and* $3 \mapsto \{2\}$ *for* C_b *after handling position* $p = 1$ *in* $\phi_2^{\succ}(b)$*. The whole procedure is then repeated for the remaining positions in* $\phi_2^{\succ}(b)$ *and afterwards for the remaining buckets.*

The reason we iterate the buckets in ascending order of their degree of freedom is that we can already build the intersection of the sets in C_b after each iteration and immediately stop if the intersection becomes empty for at least one argument position. Further implementation details and a more comprehensive example can be found in Appendix C. We next validate the practical efficiency of DEFT empirically.

5 Experiments

We compare the practical performance of DEFT to the "naive" approach (i.e., iterating over all possible permutations) and the state of the art incorporated in the ACP algorithm (which uses buckets as a filtering condition before iterating over permutations, see Appendix B for more details). For our experiments, we generate factors with $n = 2, 4, 6, 8, 10, 12, 14, 16$ Boolean arguments and a proportion $p = 0.0, 0.1, 0.2, 0.5, 0.8, 0.9, 1.0$ of identical potentials. For each scenario, we generate exchangeable factors and non-exchangeable factors. The exchangeable factors are generated by creating two factors with identical tables and then randomly permuting the arguments of one factor (and rearranging its table accordingly to keep its semantics).

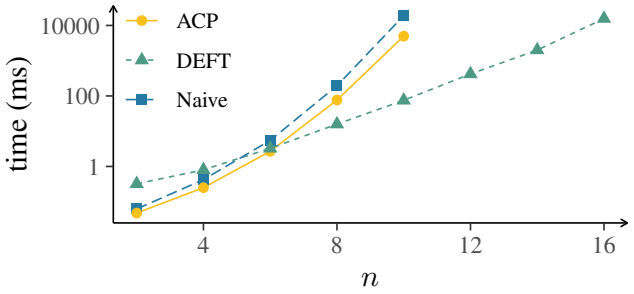


Figure 4: Average run times of the "naive" approach, the approach used in ACP, and DEFT for different numbers of arguments n. For each choice of n, the proportion of identical potentials is varied between 0.0 and 1.0 and both exchangeable as well as non-exchangeable factors are considered.

We run each algorithm with a timeout of 30 minutes per instance and report the average run time over all instances for each choice of n. Figure 4 displays the average run times on a logarithmic scale. While both the naive approach and ACP are fast on small instances with up to $n = 8$ arguments, they do not scale for larger instances. After $n = 10$, both the naive approach and ACP run into timeouts, which is not surprising as they both have to iterate over $O(n!)$ permutations. DEFT, on the other hand, is able to solve all instances within the specified timeout and is able to handle instances having nearly twice as many arguments as the instances that can be solved by the naive approach and ACP. For small values of n, DEFT is slightly slower than the naive approach and ACP due to additional pre-processing and with increasing n, DEFT greatly benefits from its pre-processing. In particular, DEFT solves instances with $n = 16$ in about ten seconds on average while the naive approach and ACP are in general not able to solve instances with $n = 12$ within 30 minutes. Finally, we remark that ACP is able to solve instances with $n > 10$ that are not exchangeable within the specified timeout but as ACP does not solve any instance of exchangeable factors with $n > 10$ within the timeout, it is not possible to compute a meaningful average run time. We thus provide further experimental results in Appendix D, where we give additional results specifically for individual scenarios.

6 Conclusion

We introduce the DEFT algorithm to efficiently detect exchangeable factors in FGs. DEFT uses buckets to drastically reduce the required computational effort in many practical settings. In particular, we prove that the number of table comparisons needed to check whether two factors are exchangeable is upper-bounded by the number of identical potential values within the buckets of the factors. By exploiting this upper bound, DEFT is able to significantly reduce the number of permutations that must be considered to check whether two factors are exchangeable.

Acknowledgements

This work is mainly funded by the BMBF project AnoMed 16KISA057 and 16KISA050K. The authors would like to thank Tanya Braun, Ralf Möller, and the anonymous reviewers for their valuable comments.

References

Ahmadi, B.; Kersting, K.; Mladenov, M.; and Natarajan, S. 2013. Exploiting Symmetries for Scaling Loopy Belief Propagation and Relational Training. *Machine Learning* 92:91–132.

Braun, T., and Möller, R. 2018. Parameterised Queries and Lifted Query Answering. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, 4980–4986. IJCAI Organization.

De Salvo Braz, R.; Amir, E.; and Roth, D. 2005. Lifted First-Order Probabilistic Inference. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, 1319–1325. Morgan Kaufmann Publishers Inc.

De Salvo Braz, R.; Amir, E.; and Roth, D. 2006. MPE and Partial Inversion in Lifted Probabilistic Variable Elimination. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, 1123–1130. AAAI Press.

Frey, B. J.; Kschischang, F. R.; Loeliger, H.-A.; and Wiberg, N. 1997. Factor Graphs and Algorithms. In *Proceedings of the Thirty-Fifth Annual Allerton Conference on Communication, Control, and Computing*, 666–680. Allerton House.

Kersting, K.; Ahmadi, B.; and Natarajan, S. 2009. Counting Belief Propagation. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-09)*, 277–284. AUAI Press.

Kisyński, J., and Poole, D. 2009. Constraint Processing in Lifted Probabilistic Inference. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-09)*, 293–302. AUAI Press.

Kschischang, F. R.; Frey, B. J.; and Loeliger, H.-A. 2001. Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory* 47:498–519.

Luttermann, M.; Braun, T.; Möller, R.; and Gehrke, M. 2024. Colour Passing Revisited: Lifted Model Construction with Commutative Factors. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)*, 20500–20507. AAAI Press.

Luttermann, M.; Möller, R.; and Gehrke, M. 2023. Lifting Factor Graphs with Some Unknown Factors. In *Proceedings of the Seventeenth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-23)*, 337–347. Springer.

Milch, B.; Zettlemoyer, L. S.; Kersting, K.; Haimes, M.; and Kaelbling, L. P. 2008. Lifted Probabilistic Inference with Counting Formulas. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*, 1062–1068. AAAI Press.

Niepert, M., and Van den Broeck, G. 2014. Tractability through Exchangeability: A New Perspective on Efficient Probabilistic Inference. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-14)*, 2467–2475. AAAI Press.

Poole, D. 2003. First-Order Probabilistic Inference. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 985–991. Morgan Kaufmann Publishers Inc.

Taghipour, N.; Fierens, D.; Davis, J.; and Blockeel, H. 2013. Lifted Variable Elimination: Decoupling the Operators from the Constraint Language. *Journal of Artificial Intelligence Research* 47:393–439.

A Missing Proofs

Corollary 1. *Let $\phi(R_1, \ldots, R_n)$. It holds that $\mathcal{F}(\phi) \leq n!$.*

Proof. Let $\{R'_1, \ldots, R'_k\}$ denote the largest subset of ϕ's arguments such that all arguments in $\{R'_1, \ldots, R'_k\}$ have the same range $\mathcal{V}$, i.e., $\mathcal{V} = \mathcal{R}(R'_1) = \cdots = \mathcal{R}(R'_k)$. If $k = 1$, $\mathcal{F}(\phi) = 1$ and thus our claim holds. Therefore, we assume that $k > 1$ for the remaining part of this proof. Then, there exists a bucket $b = [n_1, \ldots, n_{|\mathcal{V}|}]$ (with $\sum_i n_i = k$) such that one of the n_i is set to $k - 1$, another one is set to 1, and all remaining are set to 0. In other words, bucket b represents all assignments such that there is a $v_i \in \mathcal{V}$ appearing $k - 1$ times and there is a $v_j \in \mathcal{V}$ with $v_j \neq v_i$ appearing once. Therefore, b represents exactly k assignments because there are k possible positions for v_i to be located at. Consequently, $|\phi(b)| = k$ because there is a potential in b for each assignment represented by b. As $\mathcal{F}(b)$ is maximal when all potentials in $\phi(b)$ are identical, it holds that $\mathcal{F}(b) \leq k!$. Finally, we have $k \leq n$ (because ϕ has n arguments) and as $\mathcal{F}(\phi) = \min_{b \in \{b | b \in \mathcal{B}(\phi) \wedge |\phi(b)| > 1\}} \mathcal{F}(b)$, it holds that $\mathcal{F}(\phi) \leq \mathcal{F}(b) \leq k! \leq n!$. □

B Formal Description of the Advanced Colour Passing Algorithm

The advanced colour passing (ACP) algorithm introduced by Luttermann et al. (2024) builds on the colour passing algorithm (Kersting, Ahmadi, and Natarajan 2009; Ahmadi et al. 2013) and solves the problem of constructing a lifted representation (i.e., a so-called parametric factor graph) from a given factor graph (FG). The idea of ACP is to first find symmetries in a propositional FG and then group together symmetric subgraphs. ACP looks for symmetries based on potentials of factors, on ranges and evidence of random variables (randvars), as well as on the graph structure by passing around colours. Algorithm 2 provides a formal description of the ACP algorithm, which proceeds as follows.

ACP begins with the colour assignment to variable nodes, meaning that all randvars that have the same range and observed event are assigned the same colour. Thereafter, ACP assigns colours to factor nodes such that factors representing identical potentials are assigned the same colour. Two factors represent identical potentials if they are exchangeable according to Def. 2, i.e., if there exists a rearrangement of one of the factor's arguments such that both factors have identical tables of potentials when comparing them row by row. In its original form, ACP first checks whether two factors entail the same buckets and whether all buckets contain the same potential values. If this initial check fails, the two factors cannot be exchangeable and ACP assigns different colours to them, otherwise, ACP iterates over all possible permutations of the arguments of one factor to check whether there exists a permutation such that the tables of potential mappings are identical for both factors. The detection of exchangeable factors (DEFT) algorithm avoids this exhaustive search over all permutations of arguments by restricting the search space to a small subset of possible permutations. After the initial colour assignments, ACP passes the colours around. ACP first passes the colours from every variable node to its neighbouring factor nodes and afterwards, every factor node ϕ sends its colour plus the position $p(R, \phi)$ of R in ϕ's argument list to all of its neighbouring variable nodes R. For more details about the colour passing procedure and the grouping of nodes, we refer the reader to (Luttermann et al. 2024). The authors also provide extensive results about the benefits of constructing a lifted representation for probabilistic inference.

Algorithm 2 Advanced Colour Passing (as introduced by Luttermann et al., 2024)

Input: An FG G with randvars $\boldsymbol{R} = \{R_1, \ldots, R_n\}$, and factors $\boldsymbol{\Phi} = \{\phi_1, \ldots, \phi_m\}$, as well as a set of evidence $\boldsymbol{E} = \{R_1 = r_1, \ldots, R_k = r_k\}$.

Output: A lifted representation G' in form of a parametric factor graph (PFG) with equivalent semantics to G.

Assign each R_i a colour according to $\mathcal{R}(R_i)$ and $\boldsymbol{E}$
Assign each ϕ_i a colour according to order-independent potentials and rearrange arguments accordingly
repeat
  for each factor $\phi \in \boldsymbol{\Phi}$ **do**
    $signature_\phi \leftarrow [\,]$
    for each randvar $R \in neighbours(G, \phi)$ **do**
      ▷ In order of appearance in ϕ
      $append(signature_\phi, R.colour)$
    $append(signature_\phi, \phi.colour)$
  Group together all ϕs with the same signature
  Assign each such cluster a unique colour
  Set $\phi.colour$ correspondingly for all ϕs
  for each randvar $R \in \boldsymbol{R}$ **do**
    $signature_R \leftarrow [\,]$
    for each factor $\phi \in neighbours(G, R)$ **do**
      if ϕ is commutative w.r.t. $\boldsymbol{S}$ and $R \in \boldsymbol{S}$ **then**
        $append(signature_R, (\phi.colour, 0))$
      else
        $append(signature_R, (\phi.colour, p(R, \phi)))$
    Sort $signature_R$ according to colour
    $append(signature_R, R.colour)$
  Group together all Rs with the same signature
  Assign each such cluster a unique colour
  Set $R.colour$ correspondingly for all Rs
until grouping does not change
$G' \leftarrow$ construct PFG from groupings

C Implementation Details of DEFT

To complement the description of the DEFT algorithm given in Alg. 1, we provide implementation details on the computation of possible swaps (rearrangements) of ϕ_2's arguments to obtain identically ordered buckets for $\phi_1(R_1, \ldots, R_n)$ and $\phi_2(R'_1, \ldots, R'_m)$ (Line 6 in Alg. 1) in this section. To check whether it is possible to rearrange the arguments of ϕ_2 such that $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$ holds for a bucket b, DEFT proceeds as follows. For each position $p \in \{1, \ldots, |\phi_2^{\succ}(b)|\}$ in $\phi_2^{\succ}(b)$, DEFT looks up all positions p' in $\phi_1^{\succ}(b)$ that contain the same potential value as $\phi_2^{\succ}(b)$ at position p and then

builds sets of possible rearrangements of arguments for each position. Let φ and φ' denote the two potential values at positions p and p' in $\phi_2^{\succ}(b)$, respectively, which in turn correspond to two assignments (rows) in the table of ϕ_2. Note that it is also possible that the corresponding assignments are identical (i.e., they refer to the same row in the table of ϕ_2). Let $\mathcal{A} = (a_1, \ldots, a_m)$ and $\mathcal{A}' = (a'_1, \ldots, a'_m)$ denote the assignments that are mapped to φ and φ', respectively, by ϕ_2. To swap the positions of φ and φ' in $\phi_2^{\succ}(b)$ (i.e., to rearrange the order of potential values in $\phi_2^{\succ}(b)$ such that $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$), the arguments of ϕ_2 can be rearranged in any way such that $\mathcal{A}$ maps to φ' and $\mathcal{A}'$ maps to φ afterwards. Therefore, DEFT iterates over the assignment $\mathcal{A}$ and builds sets of possible rearrangements for each position $1, \ldots, m$ in $\mathcal{A}$. More specifically, DEFT stores for each position $i \in \{1, \ldots, m\}$ a set of positions $\{p_1, \ldots, p_\ell\}$ such that $a_i = a'_{p_j}$ for all $j \in \{1, \ldots, \ell\}$.

Example 8. *Consider again Fig. 3 and let* $b = [2, 1]$. *DEFT begins by considering the position* $p = 1$ *in* $\phi_2^{\succ}(b)$, *which is assigned the value* φ_3. *Then, DEFT looks up all positions in* $\phi_1^{\succ}(b)$ *that are assigned the value* φ_3 *and obtains a single position* $p' = 2$. $\phi_2^{\succ}(b)$ *contains the value* φ_5 *at position* $p' = 2$. *The corresponding assignments of* φ_3 *and* φ_5 *in the table of* ϕ_2 *are given by the assignments* $\mathcal{A} = (\text{true}, \text{true}, \text{false})$ *and* $\mathcal{A}' = (\text{true}, \text{false}, \text{true})$, *respectively. Then, DEFT builds sets of possible rearrangements of* ϕ_2*'s arguments as follows. The first position in the argument list is assigned the value* true *in* $\mathcal{A}$. *In* $\mathcal{A}'$, true *is assigned to the first and third position, i.e., position one can be rearranged either at position one or three, denoted as* $1 \mapsto \{1, 3\}$. *DEFT continues this step for the remaining values in* $\mathcal{A}$ *and obtains* $1 \mapsto \{1, 3\}$, $2 \mapsto \{1, 3\}$, *and* $3 \mapsto \{2\}$ *for position* $p = 1$ *in* $\phi_2^{\succ}(b)$.

Next, DEFT considers the position $p = 2$ *in* $\phi_2^{\succ}(b)$, *which is assigned the value* φ_5 *and looks up the position* $p' = 3$ *of* φ_5 *in* $\phi_1^{\succ}(b)$. $\phi_2^{\succ}(b)$ *contains the value* φ_2 *at position* $p' = 3$. *The corresponding assignments of* φ_5 *and* φ_2 *are given by* $\mathcal{A} = (\text{true}, \text{false}, \text{true})$ *and* $\mathcal{A}' = (\text{false}, \text{true}, \text{true})$, *respectively. Therefore, DEFT obtains* $1 \mapsto \{2, 3\}$, $2 \mapsto \{1\}$, *and* $3 \mapsto \{2, 3\}$ *as possible rearrangements of* ϕ_2*'s arguments for position* $p = 2$ *in* $\phi_2^{\succ}(b)$.

DEFT then repeats the procedure for the last position $p = 3$. $\phi_2^{\succ}(b)$ *contains the value* φ_2 *at position* $p = 3$ *and DEFT obtains* $p' = 1$ *as* φ_2 *occurs at position* $p' = 1$ *in* $\phi_1^{\succ}(b)$. $\phi_2^{\succ}(b)$ *contains the value* φ_3 *at position* $p' = 1$. *The corresponding assignments of* φ_2 *and* φ_3 *are given by* $\mathcal{A} = (\text{false}, \text{true}, \text{true})$ *and* $\mathcal{A}' = (\text{true}, \text{true}, \text{false})$, *respectively. In consequence, DEFT obtains the possible rearrangements* $1 \mapsto \{3\}$, $2 \mapsto \{1, 2\}$, *and* $3 \mapsto \{1, 2\}$ *for position* $p = 3$ *in* $\phi_2^{\succ}(b)$.

As every position $p \in \{1, 2, 3\}$ *must be arranged such that the potential value located at position* p *in* $\phi_1^{\succ}(b)$ *is identical to the potential value located at position* p *in* $\phi_2^{\succ}(b)$, *DEFT computes the intersection of all sets of possible rearrangements for each position to obtain a possible rearrangement of* ϕ_2*'s arguments that ensures* $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$. *Building the intersections for the sets of possible rearrangements, for position one it holds that* $1 \mapsto \{1, 3\} \cap \{2, 3\} \cap \{3\} = \{3\}$, *for position two we have* $2 \mapsto \{1, 3\} \cap \{1\} \cap \{1, 3\} = \{1\}$, *and for position three we obtain* $3 \mapsto \{2\} \cap \{2, 3\} \cap \{1, 2\} = \{2\}$.

As we have seen, all potential values must be located at the same position in their respective bucket for both ϕ_1 and ϕ_2 and in order to ensure this, DEFT computes the intersection of all sets of possible rearrangements for each position in the arguments of ϕ_2. Consequently, if there exists an argument position i such that $i \mapsto \emptyset$ after the intersection, there is no possibility to rearrange the arguments of ϕ_2 to ensure that $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$ holds and DEFT immediately stops. Further, as $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$ must hold for *all* buckets $b \in \mathcal{B}(\phi_1) = \mathcal{B}(\phi_2)$, DEFT builds sets of possible rearrangements for each bucket and then computes the intersection of them. If there is an argument position for which the intersection is empty, no rearrangement is possible to ensure that $\phi_1^{\succ}(b) = \phi_2^{\succ}(b)$ holds for all buckets.

Example 9. *To continue Ex. 8, DEFT computes the intersection of the sets of possible rearrangements for all remaining buckets* $b \in \{[3, 0], [0, 3], [1, 2]\}$. *Afterwards, DEFT obtains the possible rearrangements* $1 \mapsto \{3\}$, $2 \mapsto \{1\}$, *and* $3 \mapsto \{2\}$. *Finally, DEFT rearranges the arguments of* ϕ_2 *such that position one (*R_4*) is located at position three, position two (*R_5*) is located at position one, and position three (*R_6*) is located at position two, i.e., we obtain* R_5, R_6, R_4 *as the new argument list for* ϕ_2, *and DEFT verifies that the tables of* ϕ_1 *and* ϕ_2 *are identical after this rearrangement.*

Moreover, we remark that it is also conceivable to compute the intersection of possible rearrangements over a subset of the buckets only. In particular, each bucket further restricts the set of possible rearrangements, that is, the less buckets we consider, the more possible rearrangements are permitted, which, in turn, must all be considered for verification to determine whether ϕ_1 and ϕ_2 are exchangeable.

Example 10. *Consider again Fig. 3 and let* $b = [1, 2]$. *After iterating over all positions in* $\phi_2^{\succ}(b)$, *DEFT obtains the sets of possible rearrangements* $1 \mapsto \{2, 3\}$, $2 \mapsto \{1\}$, *and* $3 \mapsto \{2, 3\}$. *Consequently, there are multiple options for rearranging the arguments of* ϕ_2 *if we consider only the bucket* b. *In case there are multiple options for rearranging the arguments of* ϕ_2, *DEFT checks whether there is an option such that the tables of* ϕ_1 *and* ϕ_2 *are identical. The possible rearrangements of* ϕ_2*'s arguments are illustrated in Fig. 5. DEFT tries them in any order, so it might start with placing* R_4 *(currently position* 1*) at position* 2, *then places* R_5 *(currently position* 2*) at position* 1, *and finally places* R_6 *(currently position* 3*) at position* 3. *Note that we originally had* $3 \mapsto \{2, 3\}$, *i.e.,* R_6 *could potentially be placed at position* 2 *as well but as we already fixed position* 2 *(by placing* R_4 *there), the only remaining possible position for* R_6 *is position* 3. *After the rearrangement to* R_5, R_4, R_6, *the verification that the tables of* ϕ_1 *and* ϕ_2 *are identical fails and DEFT tries the next possible rearrangement. The next rearrangement is then* R_4, R_6, R_5 *and the verification succeeds. In case none of the possible rearrangements results in identical tables,* ϕ_1 *and* ϕ_2 *are not exchangeable.*

Note that the number of paths in the tree depicted in Fig. 5 depends on the number of identical potential values in the

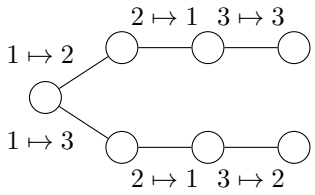


Figure 5: Possible rearrangements of the argument positions 1, 2, and 3 for the sets of possible rearrangements $1 \mapsto \{2, 3\}$, $2 \mapsto \{1\}$, and $3 \mapsto \{2, 3\}$.

bucket, which is mostly small in practical applications as in general different assignments map to different potential values. Therefore, DEFT is guaranteed to never consider more rearrangements than previous approaches.

In our implementation, we deploy a heuristic within DEFT such that only the five buckets with the lowest degree of freedom are considered to compute the intersection of possible rearrangements. The use of such an heuristic does not impact the correctness of DEFT while at the same time significantly reduces the overhead for computing the intersections over large buckets.

D Further Experimental Results

In addition to the experimental results provided in Sec. 5, we give further plots for individual scenarios in this section. Figure 6 shows the run times for instances with a proportion of $p = 0.0$ identical potentials, i.e., each input factor maps each assignment of its arguments to a different potential value. The plot on the left shows the results for factors that are not exchangeable and the plot on the right depicts results for exchangeable factors. For both exchangeable and non-exchangeable factors, the naive approach is able to handle instances with up to $n = 10$ arguments and fails to solve larger instances. While ACP exhibits a similar behaviour as the naive approach for exchangeable factors, ACP is able to solve all non-exchangeable instances. This is expected as all non-exchangeable instances are generated such that the buckets of the factors under consideration are not mapped to the same multiset of potential values and thus, ACP is able to return before iterating over any permutations. The left plot also indicates that DEFT induces some overhead when iterating over the buckets, which is in line with our expectation as DEFT performs additional pre-processing and computes intersections of possible argument rearrangements while ACP does not. At the same time, DEFT is able to solve all instances independent of whether the factors are exchangeable or not whereas ACP solves larger instances only for non-exchangeable factors within the specified timeout.

Figure 7 contains the average run times for instances with a proportion of $p \in \{0.1, 0.2, 0.5, 0.8, 0.9\}$ identical potentials. More specifically, every input factor maps each assignment of its arguments to the same potential value with a probability of p (i.e., out of all rows in the table, there is roughly a proportion of p rows that are mapped to the same potential value). We can see that both plots depicted in Fig. 7 exhibit the same patterns as the plots in Fig. 6.

Finally, Fig. 8 presents the run times for instances with a proportion of $p = 1.0$ identical potentials, that is, every input factor maps each assignment of its arguments to the same potential value. Even though this scenario is rather unrealistic for practical applications, we include it as an extreme case. The left plot again shows similar patterns as the plots for non-exchangeable factors in Figs. 6 and 7. The right plot illustrates that both the naive approach and ACP are now able to solve all instances of non-exchangeable factors within the specified timeout. All three approaches have a similar run time, which can be explained by the fact that only a single permutation must be considered as the arguments can be rearranged arbitrarily to obtain identical tables (in fact, the tables of the input factors are already identical).

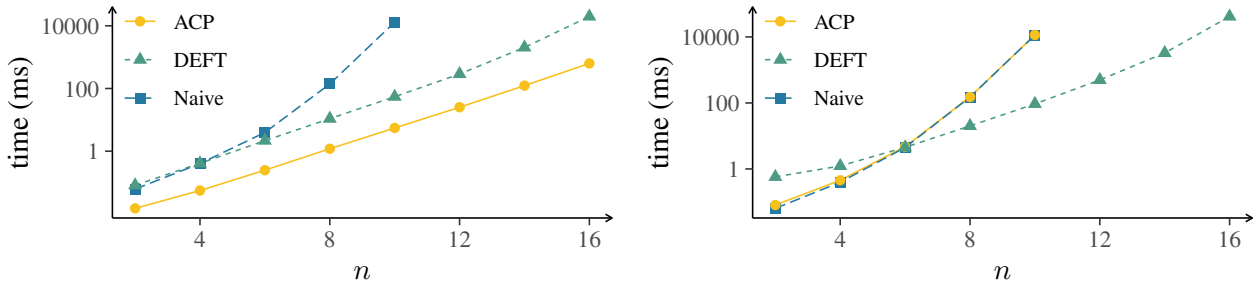


Figure 6: Run times of the "naive" approach, the approach used in ACP, and DEFT for different numbers of arguments n and a proportion $p = 0.0$ of identical potentials. The left plot shows results for non-exchangeable instances and the right plot presents results for exchangeable instances. Both plots use a logarithmic scale.

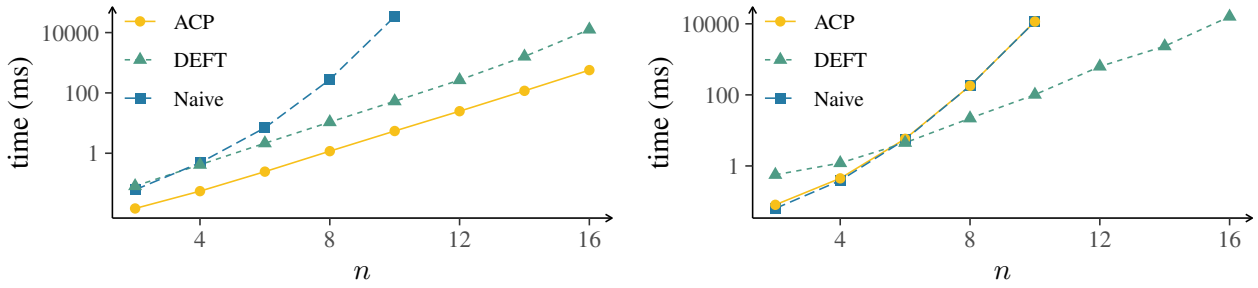


Figure 7: Run times of the "naive" approach, the approach used in ACP, and DEFT for different numbers of arguments n and different proportions $p \in \{0.1, 0.2, 0.5, 0.8, 0.9\}$ of identical potentials (averaged). The left plot shows results for non-exchangeable instances and the right plot presents results for exchangeable instances. Both plots use a logarithmic scale.

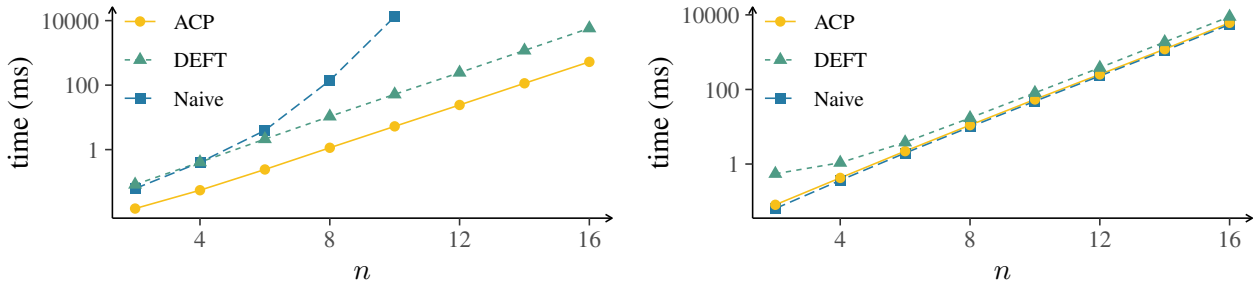


Figure 8: Run times of the "naive" approach, the approach used in ACP, and DEFT for different numbers of arguments n and a proportion $p = 1.0$ of identical potentials. The left plot shows results for non-exchangeable instances and the right plot presents results for exchangeable instances. Both plots use a logarithmic scale.